Formulation et terminologie

- Description générale d'un problème d'optimisation
  - Ω : **ensemble de configurations** (ou de solutions) du problème
    C : **fonction de coût** (ou objectif)
  - Trouver la configuration $x' \in \Omega$ qui minimise le coût minimal

$$\mathrm{C}\left(x'\right) = \min\left\{\mathrm{C}\left(x\right) \middle| x \in \Omega\right\}$$

  - x' n'est pas obligatoirement unique;
  - la **fonction de coût** est aussi appelée ***fitness function***
  - Ω est aussi appelé l'**espace de recherche**

Heuristiques et métaheuristiques

- Une **heuristique** est une méthode pour résoudre de manière approchée un type de problème particulier, dont la solution optimale exacte ne peut être obtenue (car, par exemple, son obtention nécessiterait un temps de calcul d'ordinateur, de plusieurs milliers d'années)
- Une **métaheuristique** est une méthode, ou plus précisément, un canevas de méthodes, pour résoudre de manière approchée tous les problèmes dont la solution optimale ne peut être obtenue. La méthode ne dépend donc plus du type de problème auquel on est confronté

Définir un problème de recherche

Espace d'états

- chaque état est une représentation abstraite de l'environnement
- l'espace d'état est discret, il peut être fini ou infini

État initial

- habituellement l'état courant
- parfois un ou plusieurs états hypothétiques

Fonction "successeur"

- fonction : [état -> sous-ensemble d'états]
- une représentation abstraite des actions possibles

Etat-solution

- habituellement une condition à satisfaire
- parfois la description explicite d'un état

Coût du chemin

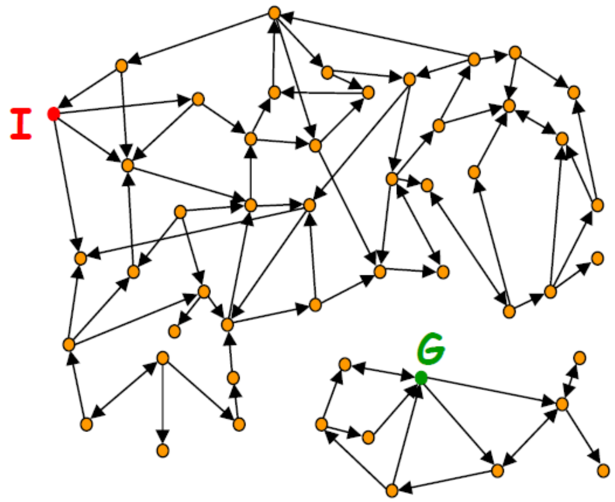
- fonction : [chemin -> nombre positif]
- habituellement: coût du chemin = somme des coûts de ses étapes
- Ex: # déplacements de la plaquette "vide"

Recherche heuristique: généralités

- Les algorithmes de recherche aveugle n'exploitent aucune information concernant la structure de l'arbre de recherche ou la présence potentielle de noeuds-solution pour optimiser la recherche.
- Recherche "rustique" à travers l'espace jusqu'à trouver une solution.
- La plupart des problèmes réels sont susceptibles de provoquer une explosion combinatoire du nombre d'états possibles.
- Un algorithme de recherche heuristique utilise toute l'information disponible pour rendre le processus de recherche plus efficace.
- Une information heuristique est une règle ou une méthode qui améliore presque toujours le processus de recherche.

Solution à un problème de recherche

- Une **solution** est un chemin reliant l'état initial I à un état solution G (n'importe lequel)
- Le **coût** d'un chemin est la somme des coûts des arcs qui le constituent
- Une solution **optimale** est un chemin-solution de coût minimum (la plupart du temps) ou maximum
- Il peut ne pas y avoir de solution !!!



Algorithme général de recherche (#1)

1. Si SOLUTION?(état-initial) alors retourner état-initial
2. INSERER(noeud-initial,FILE)
3. Répéter:
   a. Si vide(FILE) alors retourner échec
   b. n ← RETIRER(FILE) *Prolongement de n*
   c. s ← ETAT(n)
   d. Pour chaque état s' SUCCESSEUR(s)
      i. Créer un nouveau noeud n' comme "enfant" de n
      ii. Si SOLUTION?(s') alors retourner chemin ou état-solution
      iii. INSERER(n',FILE)

Les méthodes de recherche diffèrent les unes des autres selon l'ordre dans lequel les nœuds sont insérés et extraits de la file d'attente.

Algorithme de « backtracking »

PSC-BACKTRACKING(A)

1. Si assignement A est complet alors retourner A
2. X ← sélectionner une variable absente de A
3. D ← sélectionner un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
    a. Add (X←v) à A
    b. Si A est valide alors
        i. résultat ← PSC-BACKTRACKING(A)
        ii. Si résultat ≠ échec alors retourner résultat
5. Retourner échec

Appel: PSC-BACKTRACKING({})

Algorithme de backtracking modifié

PSC-BACKTRACKING(A, var-domaines)

1. Si assignement A est complet alors retourner A
2. X ← **sélectionner** une variable absente de A
3. D ← **sélectionner** un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
    a. Add (X←v) à A
    b. var-domaines ← forward checking(var-domaines, X, v, A)
    c. Si aucune variable a un domaine vide alors
        i. résultat ← PSC-BACKTRACKING(A, var-domaines)
        ii. Si résultat ≠ échec alors retourner résultat
    d. Enlever (X←v) de A
5. Retourner échec

Algorithme Backtracking modifié

PSC-BACKTRACKING(A, var-domaines)

1. Si assignement A est complet alors retourner A
2. X ← **sélectionner** une variable absente de A
3. D ← **sélectionner** un ordre sur le domaine de X
4. Pour chaque valeur v dans D faire
   a. Add (X←v) à A
   b. var-domaines ← forward checking(var-domaines, X, v, A)
   c. Si aucune variable a une domaine vide alors
      i. résultat ← PSC-BACKTRACKING(A, var-domaines)
      ii. Si résultat ≠ échec alors retourner résultat
5. Retourner échec

1) Heuristique variable-plus-contrainte
2) Heuristique variable-plus-contraignante

1) Heuristique valeur-moins-contraignante

1) Sélectionner la variable ayant le plus petit domaine restant
2) Sélectionner la variable apparaissant dans le plus grand nombre de contraintes sur des variables absentes de l'assignement courant

Résolution avec tableaux

- Maximiser

$$5x_1 + 6.5x_2 + 8x_3 + 9x_4 = z$$

- Sous contraintes

$$\begin{aligned} 3x_1 + 6x_2 + 3x_3 + 9x_4 + e_1 &= 240 \\ 12x_1 + 15x_2 + 9x_3 + 12x_4 + e_2 &= 150 \\ 0.1x_1 + 0.1x_2 + 0.1x_3 + 0.1x_4 + e_3 &= 2 \end{aligned}$$

$$e_1, e_2, e_3, x_1, x_2, x_3, x_4 \geq 0$$

Résolution avec tableaux

- Nouvelle représentation

variables hors base				variables de base			
x_1	x_2	x_3	x_4	e_1	e_2	e_3	
3	6	3	9	1	0	0	240
12	15	9	12	0	1	0	150
0,1	0,1	0,1	0,1	0	0	1	2
5	6,5	8	9	0	0	0	Z

Résolution avec tableaux

- Itération 1: choix de la variable entrante

x_1	x_2	x_3	x_4	e_1	e_2	e_3	
3	6	3	9	1	0	0	240
12	15	9	12	0	1	0	150
0,1	0,1	0,1	0,1	0	0	1	2
5	6,5	8	9	0	0	0	Z

- x_4 entre dans la base

Résolution avec tableaux

- Itération 1: choix de la variable sortante

x_1	x_2	x_3	x_4	e_1	e_2	e_3	
3	6	3	9	1	0	0	240
12	15	9	12	0	1	0	150
0,1	0,1	0,1	0,1	0	0	1	2
5	6,5	8	9	0	0	0	Z

- min(240/9, 150/12, 2/0.1) $\Rightarrow$ e_2 sort de la base

Résolution avec tableaux

- Itération 1: x_4 rentre et e_2 sort de la base
  $\Rightarrow$ on échange les colonnes de x_4 et e_2

$\underbrace{}_{}$ variables hors base				variables de base			
x_1	x_2	x_3	e_2	e_1	x_4	e_3	
3	6	3	0	1	9	0	240
12	15	9	1	0	12	0	150
0, 1	0, 1	0, 1	0	0	0, 1	1	2
5	6, 5	8	0	0	9	0	Z

Résolution avec tableaux

- Itération 1: reconstruction de la matrice identité
  - on normalise la seconde ligne par 12

x_1	x_2	x_3	e_2	e_1	x_4	e_3	
3	6	3	0	1	9	0	240
1	$\frac{15}{12}$	$\frac{9}{12}$	$\frac{1}{12}$	0	1	0	$\frac{150}{12}$
1	$\frac{5}{4}$	$\frac{3}{4}$	$\frac{1}{12}$	0	1	0	$\frac{25}{2}$
$0,1$	$0,1$	$0,1$	0	0	$0,1$	1	2
5	$6,5$	8	0	0	9	0	Z

  - puis on l'utilise comme **pivot** pour éliminer x_4 des autres lignes

Résolution avec tableaux

- À la fin de la première itération

x_1	x_2	x_3	e_2	e_1	x_4	e_3	
-6	$-\frac{21}{4}$	$-\frac{15}{4}$	$-\frac{3}{4}$	1	0	0	$\frac{255}{2}$
1	$\frac{5}{4}$	$\frac{3}{4}$	$\frac{1}{12}$	0	1	0	$\frac{25}{2}$
0	$-\frac{1}{40}$	$\frac{1}{40}$	$-\frac{1}{120}$	0	0	1	$\frac{3}{4}$
5	$6,5$	8	0	0	9	0	Z

Résolution avec tableaux

- Itération 2 : choix de la variable entrante

x_1	x_2	x_3	e_2	e_1	x_4	e_3	
-6	$-\frac{21}{4}$	$-\frac{15}{4}$	$-\frac{3}{4}$	1	0	0	$\frac{255}{2}$
1	$\frac{5}{4}$	$\frac{3}{4}$	$\frac{1}{12}$	0	1	0	$\frac{25}{2}$
0	$-\frac{1}{40}$	$\frac{1}{40}$	$-\frac{1}{120}$	0	0	1	$\frac{3}{4}$
-4	$-\frac{19}{4}$	$\frac{5}{4}$	$-\frac{3}{4}$	0	0	0	$Z-\frac{225}{2}$

- x_3 entre dans la base

Le recuit simulé

- Algorithme du recuit simulé (version Metropolis)

$T \leftarrow T_0$ -- Température initiale

$X \leftarrow X_0$ -- Configuration initiale

répéter

  répéter

    Tirer aléatoirement $Y \in$ Voisinage(X)

    si $\Delta E = E(Y)\text{-}E(X) < 0$ ou $\exp(-\Delta E / T) > \mu$, $\mu \in [0;1]$ aléatoire

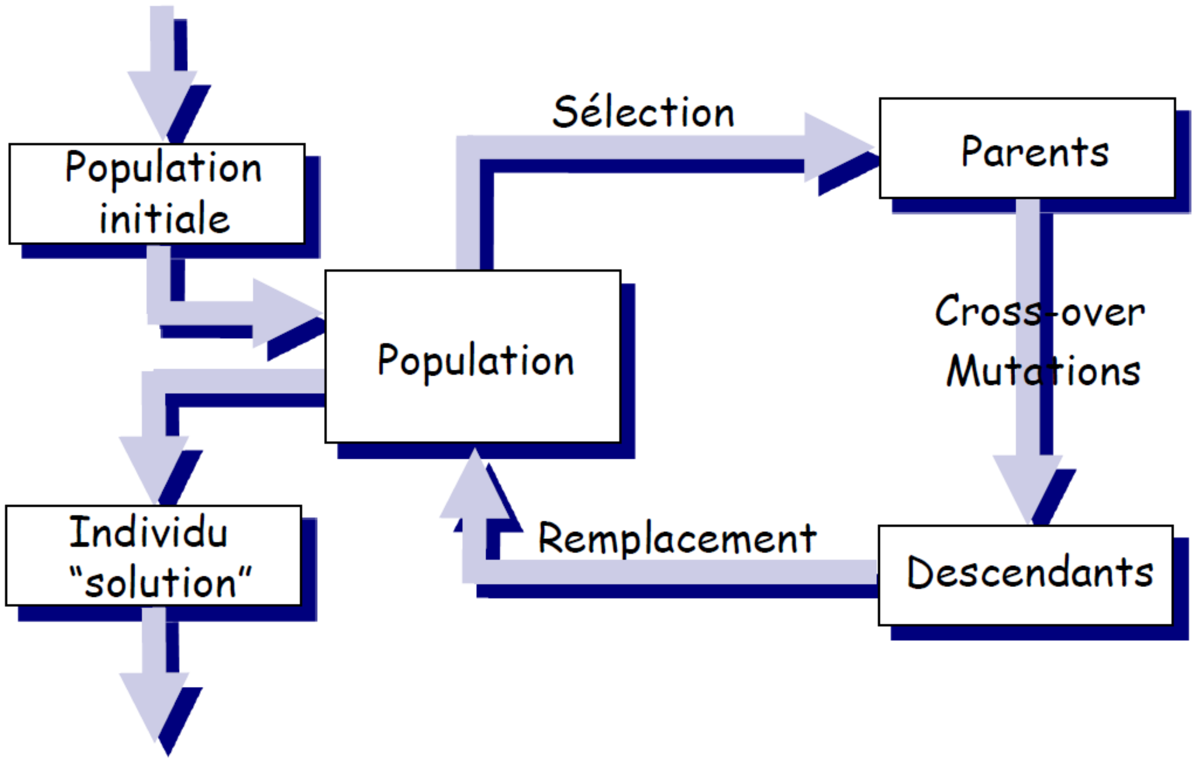
    alors $X \leftarrow Y$

  jusqu'à fin palier

  $T \leftarrow g(T)$ -- refroidissement : g strictement décroissante

jusqu'à critère d'arrêt vérifié

Principe du cycle d'évolution



Algorithme génétique

```
créer population initiale
faire
  évaluer tous les individus (avec fct fitness)

  faire
     sélectionner parents
     créer descendants par recombinaison
     faire muter descendants
  jusqu'à taille population

  remplacer tous les individus par les descendants
jusqu'à critère d'arrêt
```

Algos génétiques - Sélections

Naturelle

- Assurer que les meilleurs individus aient plus de chance d'avoir une descendance que les autres.
- Les moins bons individus doivent conserver une chance de se reproduire :
  - ils peuvent contenir des gènes intéressants.
  - la pression de sélection doit préserver une part de bio-diversité.

Proportionnelle (selon sa fonction fitness)

Désavantages : * Réduction rapide de la diversité * Risque de convergence prématurée * La pression de sélection reste faible lorsque les fitness sont très similaires

Par le rang

Sélection linéaire par le rang aligne la population selon le rang et donne une probabilité de sélection proportionnelle au rang La droite ne descend pas obligatoirement entre 2.0 et 0.0, par exemple entre 1.5 et 0.5 * Possibilité d'un rang non linéaire * Plus couramment utilisé: rang linéaire avec une pente de 2.0 à 0.0 * Ainsi le meilleur se reproduit 2 fois plus que le médian. Les solutions inférieures à la moyenne gardent une chance de se reproduire

Par compétition

- Sélection basée sur un remplacement partiel de la population
  - Sélection aléatoire de k individus (sans remplacement),
  - Compétition entre les k individus,
  - Seul le meilleur se reproduit
- Très en vogue

Intérêts relatifs de la mutation et du cross-over

- Cross-over
  - Faible part d'aléa, permet de ré-exprimer des caractères déjà présents dans la population,
  - L'effet du cross-over diminue lorsqu'on se rapproche de la convergence,
  - Opérateur d'***exploitation***
- Mutations
  - Indispensables pour éviter les maxima locaux,
  - Opérateur d'***exploration*** ...

⇘ *La recombinaison est souvent plus difficile à mettre en œuvre ...*

Critère d’arrêt ...

- Difficile à déterminer puisque la qualité de la solution s’améliore de façon quasi-continue
  - Plusieurs « solutions »
- Le maximum est atteint
  - Suppose qu’il soit connu !
  - Il s’agit souvent d’un maximum *utile ...*
- Limite de temps de calcul
  - On ne peut pas toujours évaluer autant d’individus qu’on le souhaite
- Limite de patience de l’utilisateur
  - Lorsque rien n’évolue pendant plusieurs générations

Point clef de l'algorithmique génétique

- Maintien de la diversité génétique au cours de l'évolution
  - Il faut maintenir des caractéristiques génétiques différentes dans la population (bio-diversité)
  - Lorsqu'on perd la diversité génétique, tous les individus deviennent semblables
  - Effet boule de neige
  - Convergence vers l'optimum local le plus proche

⇘ *En théorie, les mutations permettent de continuer à explorer l'espace … en pratique, la perte de diversité génétique est irréversible …*

Point clef de l'algorithmique génétique

- Dilemme Exploration / Exploitation
  - Exploration = tester les zones inconnues

    Trop d'exploration revient à une marche aléatoire et compromet la convergence
  - Exploitation = essayer d'améliorer le meilleur individu trouvé jusqu'ici

    Trop d'exploitation revient à une recherche locale et conduit à une convergence vers un optimum local

⇘ *Dilemme exploration / exploitation*

Avantages des AGs

- Bon rapport coût / résultat sur une grande classe de problèmes
- Parallélisme intrinsèque
- Robuste, tolérant aux fautes
- Applicable sans connaissance préalable du domaine d'application
- Simple à programmer
- Attention, « everything is problem dependent »
  - Les algorithmes génétiques ne sont pas **la** panacée universelle
- les AGs sont particulièrement adaptés lorsque le problème :
  - contient beaucoup de données / de paramètres
  - contient des paramètres interdépendants (problème complexe)
  - comporte des optimums locaux

Inconvénients des AGs

- Pas de garantie de convergence en un temps fini
- Faiblesse des fondements théoriques et mathématiques
- Souvent gourmands en calcul donc lents (mais aisément parallélisables)
- Chaque individu doit être évalué, même les individus inadaptés (utilisation offline, sur simulateur)
- Produit toujours des individus inadaptés

⇘ *Le comportement de l 'algorithme dépend d 'un grand nombre de paramètres qui peuvent être difficiles à fixer*

Interprétation géométrique

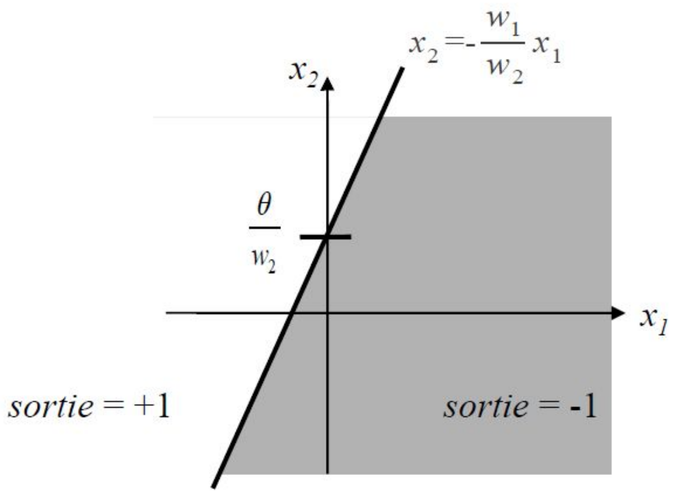
Soit un neurone formel de McCulloch&Pitts avec deux entrées x_1 et x_2

sa *sortie* est +1 si: $w_1x_1 + w_2x_2 > \theta$

sa *sortie* est -1 si: $w_1x_1 + w_2x_2 \leq \theta$

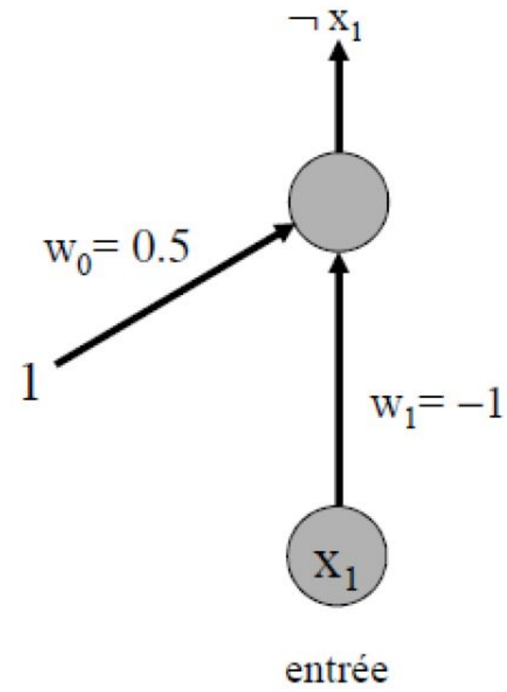
Géométriquement ces deux équations partitionnent le plan (x_1,x_2) par une droite définie par l'équation suivante:

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}$$



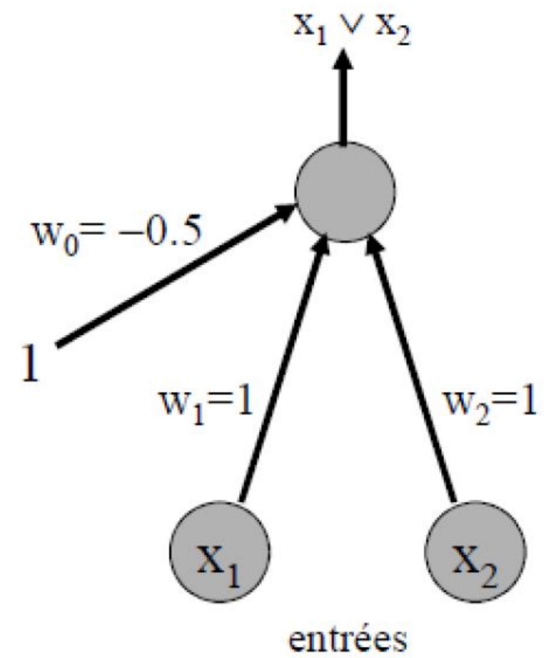
Exemple: négation logique

entrée x1	sortie
0	1
1	0



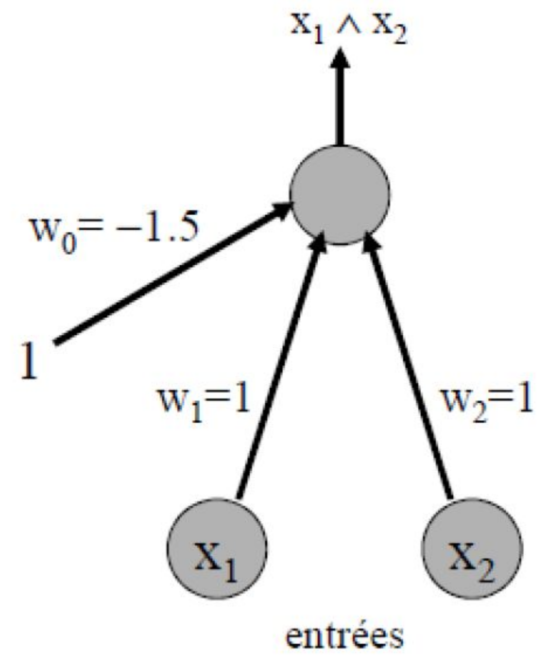
Exemple: "ou" logique

entrée x1	entrée x2	sortie
0	0	0
0	1	1
1	0	1
1	1	1

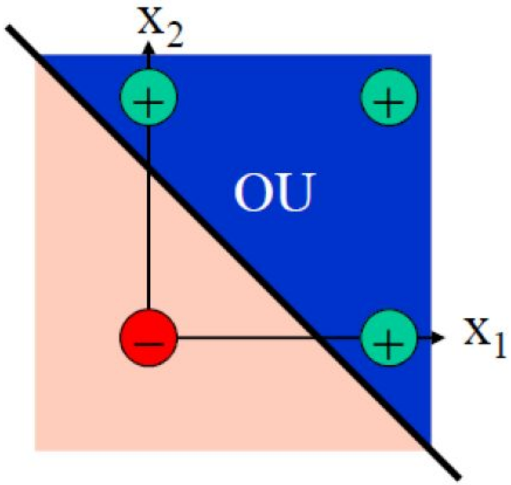


Exemple: "et" logique

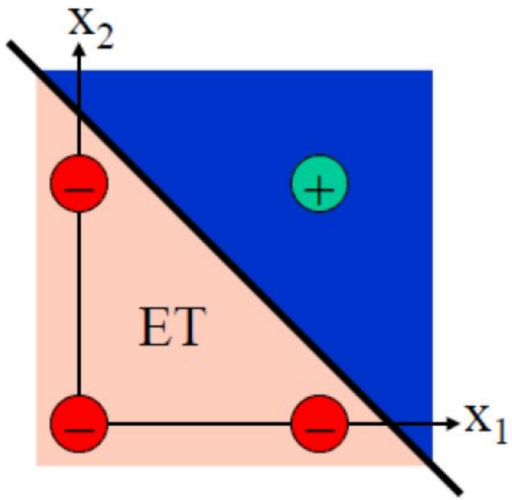
entrée x1	entrée x2	sortie
0	0	0
0	1	0
1	0	0
1	1	1



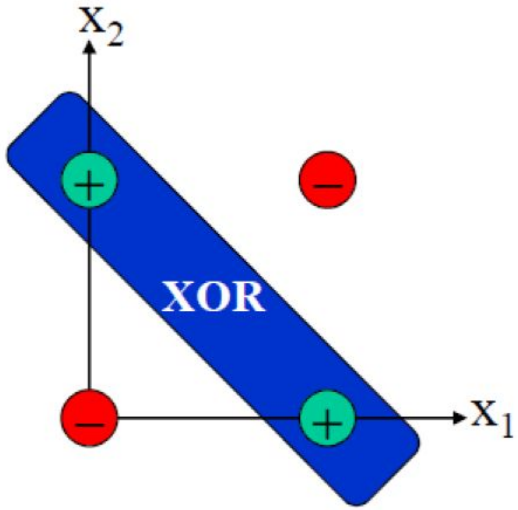
Séparabilité linéaire

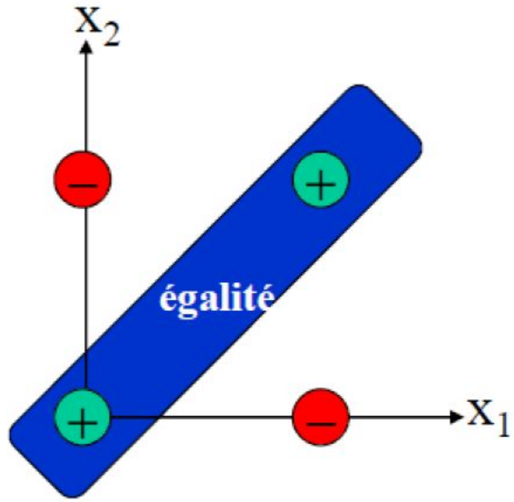


Séparabilité linéaire



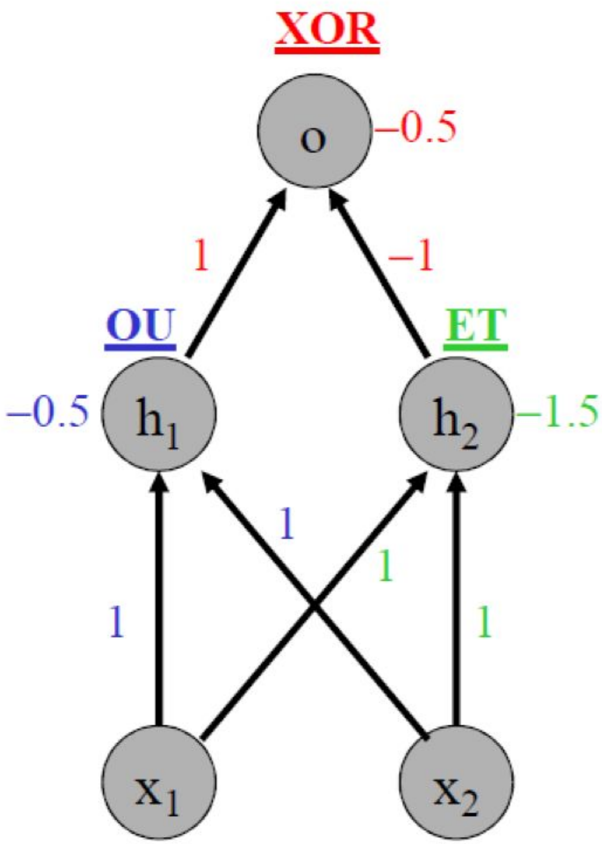
Non-séparabilité linéaire





Exemple: "xor" logique (revu)

entrée x1	entrée x2	sortie
0	0	0
0	1	1
1	0	1
1	1	0



Algorithme d'apprentissage du Perceptron

<u>Entrées</u>: ensemble d'apprentissage $\{(x_1, x_2, \ldots, x_n, t)\}$

<u>Méthode</u>

initialiser aléatoirement les poids w(i), 0<=i<=n

répéter jusqu'à convergence:

pour chaque exemple

calculer la valeur de sortie *o* du réseau.

ajuster les poids:

$$\Delta w_i = \eta\,(t - o)\, x_i$$

$$w_i \leftarrow w_i + \Delta w_i$$

Règle d'apprentissage du *Perceptron*

Algorithme d'apprentissage du Perceptron (résumé)

nouveau poids   ancien poids   modification

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta (t - o) x_i$$

modification   taux d'apprentissage   valeur attendue   valeur de sortie du Perceptron   entrée