Réseaux de neurones et Algorithmes génétiques : version 2013

MICHEL Philippe

19 janvier 2022

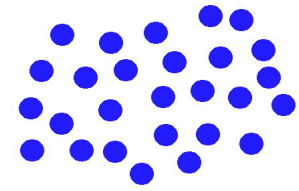
Table des matières

1 Algorithmes génétiques

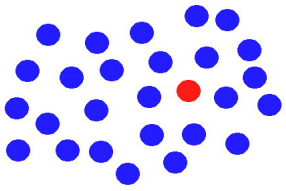
1.1 Vocabulaire

Définition 1.1 *Un individu a son trait génétique représenté par ses* **Chromosomes** *définissant son* **Génotype**. *Cela peut être une suite de lettres (ATGCCCGATT), des bits (010110), des réels ((1.5 ;0.66))... La traduction de cela est le* **Phénotype** *qui a pour conséquence une plus ou moins grande* **Adaptation (Fitness)** *au milieu. La* **Sélection** *naturelle favorise les individus les mieux adaptés qui se reproduisent de manière sexuée avec* **Recombinaison (Crossover)** *des gènes avec ou sans* **Mutation(s)** *ou asexuée par* **clonage** *et* **Mutation**.

Définition 1.2 *L'algorithme génétique (basique) :*
1. Créer une population initiale (aléatoirement) d'une taille N.
2. **Evaluer la 'fitness'** *de chaque membre de la population.*
3a. **Sélection naturelle** *: reproduction des plus adaptés.*
3b. **Générer les enfants par Crossover ou Clonage.**
3c. **Mutation** *de quelques individus.*
4. Arrêt si la fitness est assez bonne ou retour à l'étape 2.

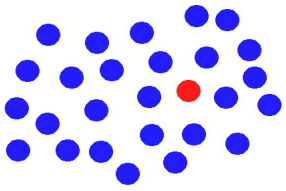


Population initiale : Fitness = 1



Population initiale : Fitness = 1

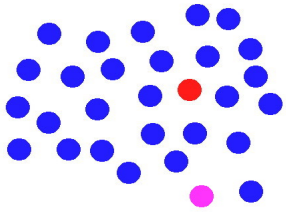
un mutant se creer : Fitness du mutant <1



Population initiale : Fitness = 1

un mutant se creer : Fitness du mutant <1

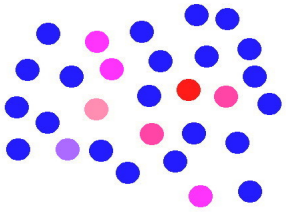
probabilité de se reproduire << => disparition



Population initiale : Fitness = 1

un mutant se creer : Fitness du mutant >1

le mutant survie, peut se reproduire :
(clonage ou recombinason)



Population initiale : Fitness = 1

un mutant se creer : Fitness du mutant >1

le mutant survie, peut se reproduire :
(clonage ou recombinason)

1.2 Compréhension

Les algorithmes génétiques sont utilisés pour résoudre des problèmes de minimisation (maximisation). Trouver le chemin le plus court, le minimum d'une fonction, des optimums de Pareto en théorie des jeux, améliorer un réseau de neurones. Dans tous les cas, on cherche

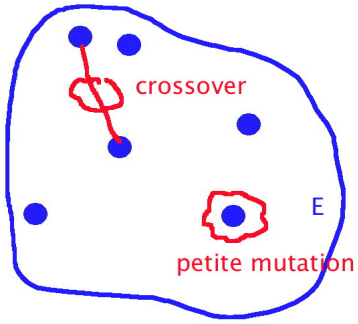
Définition 1.3 *Soit E l'espace des chromosomes (sous ensemble d'un espace vectoriel : convexe et compact) et*

$$f : E \mapsto \mathbb{R}$$

une fonction d'adaptation au milieu, on cherche

$$\min_{x\in E} f(x). \tag{1.1}$$

L'algorithme génétique permet d'explorer l'espace E pour trouver le minimum de f, pour cela



l'opérateur de mutation permet d'explorer les alentour d'une position, celui de recomposition permet d'explorer l'enveloppe convexe des positions préexistantes. Un exemple simple, sans crossover (juste mutation) donne

Proposition 1.4 *Soit E l'espace des chromosomes (sous ensemble de $\mathbb{R}^n$: convexe et compact), $f : E \mapsto \mathbb{R}$ une application continue et $T_n(\omega) = \{x_1(n,\omega); x_2(n,\omega)\}$ avec $x_i(n) \in E$ construit de la manière suivante :*

$$x_1(1); x_2(1) \sim U(E)$$

et

$$x_1(n+1,\omega) = \begin{cases} x_1(n,\omega), & si\ \ f(x_1(n,\omega)) \le f(x_2(n,\omega)) \\ x_2(n,\omega), & si\ \ f(x_2(n,\omega)) < f(x_1(n,\omega)) \end{cases}$$

$$x_2(n+1,\omega) \sim U(E)$$

alors $f(x_1(n+1,\omega)) \le f(x_1(n,\omega))$ et $d(x_1(n,\omega), \{\bar{x} \ : \ f(\bar{x}) = \min_{x\in E} f(x)\}) \to_{n\to\infty} 0$. De plus

$$N_\epsilon(\omega) := \min\{n \ : \ x_2(n,\omega) \in B(\bar{x},\epsilon))\}, \quad \bar{x} \in \{\bar{x} \ : \ f(\bar{x}) = \min_{x\in E} f(x)\}$$

vérifie

$$E(N_\epsilon) = s/(1-s)^2, \quad s := \int_{B(\bar{x},\epsilon)} dx / \int_E dx.$$

Preuve On a directement $f(x_1(n+1,\omega)) \le f(x_1(n,\omega))$. Par compacité, on peut extraire une sous suite convergente

$$x_1(n_k(\omega),\omega) \to y$$

et par continuité $f(x_1(n_k,\omega)) \to f(y)$ avec $f(y) \le f(x_1(n,\omega))$ pour tout n (sous suite et décroissance de la suite $(f(x_1(n,\omega)))_n$). Puisque f est continue et E compact, $\{\bar{x} \ : \ f(\bar{x}) = \min_{x\in E} f(x)\} \neq \emptyset$ et $P(x_2(n,\omega) \in B(\bar{x},\epsilon)) = \int_{B(\bar{x},\epsilon)} dx / \int_E dx = \alpha_\epsilon > 0$ et $P(\exists n_\epsilon \ : \ x_2(n_\epsilon,\omega) \in B(\bar{x},\epsilon)) = 1 - \prod_1^\infty (1-\alpha_\epsilon) = 1$. Si $y \notin$

$\{\bar{x} \ : \ f(\bar{x}) = \min_{x\in E} f(x)\}$ alors $f(y) > f(\bar{x})$ et il existe un voisinage de $\bar{x}$ tel que $f(y) > B(\bar{x},\epsilon)$, or p.s. $\exists n_\epsilon \ : \ x_2(n_\epsilon,\omega) \in B(\bar{x},\epsilon)$ donc $f(x_2(n_\epsilon,\omega)) < f(y)$ mais comme $f(y) \leq f(x_1(n,\omega))$ pour tout n, on aurait $f(x_2(n_\epsilon,\omega)) < f(x_1(n_\epsilon - 1,\omega))$ ce qui est absurde. Le calcul de l'espérance du premier temps d'atteinte d'un élément $\bar{x} \in \{\bar{x} \ : \ f(\bar{x}) = \min_{x\in E} f(x)\}$ est donné par

$$E(N_\epsilon) := \sum_{k=1}^{\infty} k s^k$$

et vérifie

$$E(N_\epsilon) = s/(1-s)^2, \quad s := \int_{B(\bar{x},\epsilon)} dx / \int_E dx.$$

□

Un exemple simple, sans mutation (juste crossover) donne

Proposition 1.5 *Soit E l'espace des chromosomes (= $[a,b]$ intervalle fermé borné de $\mathbb{R}$), $f : E \mapsto \mathbb{R}$ une application C^1 telle que $f'(a) < 0$ et $f'(b) > 0$ et $T_n = \{x_1(n); x_2(n)\}$ avec $x_i(n) \in E$ construit de la manière suivante :*

$$x_1(1) = a; \ x_2(1) = b,$$

et

$$x_1(n+1) = \begin{cases} x_1(n), & si \ \ f'(\frac{x_1(n)+x_2(n)}{2}) \geq 0 \\ \frac{x_1(n)+x_2(n)}{2}, & si \ \ f'(\frac{x_1(n)+x_2(n)}{2}) < 0 \end{cases}$$

$$x_2(n+1) = \begin{cases} \frac{x_1(n)+x_2(n)}{2}, & si \ \ f'(\frac{x_1(n)+x_2(n)}{2}) > 0 \\ x_2(n), & si \ \ f'(\frac{x_1(n)+x_2(n)}{2}) \leq 0 \end{cases}$$

alors $f'(x_1(n)) \to_{n\to\infty} 0$ et $d(x_1(n), \{\bar{x} \ : \ f'(\bar{x}) = 0\}) \to_{n\to\infty} 0$. De plus

$$N_\epsilon := \min\{n \ : \ x_i(n,\omega) \in B(\bar{x},\epsilon))\}, \quad \bar{x} \in \{\bar{x} \ : \ f'(\bar{x}) = 0\}$$

vérifie

$$N_\epsilon \leq |\log_2(\epsilon)| + \log_2(b-a) = \log_2(\int_{B(\bar{x},\epsilon)} dx / \int_E dx).$$

Preuve On a directement que $|x_2(n+1) - x_1(n+1)| = |x_2(n) - x_1(n)|/2 = (b-a)/2^n$, et $f'(x_i(n)) \to_{n\to\infty} 0$ donc

$$d(\bar{x}, x_1) \leq (b-a)/2^n$$

et $N_\epsilon \leq |\log_2(\epsilon)| + \log_2(b-a)$. □

Le résultat se généralise en dimension n

Proposition 1.6 *Soit E l'espace des chromosomes (= $\prod_{i=1}^{l}[a_i,b_i]$ sous espace fermé borné convexe de $\mathbb{R}^l$), $f : E \mapsto \mathbb{R}$ une application C^1 telle que $\nabla f(x).\eta(x) < 0$ avec $x \in \partial E$ et $\eta(x)$ le vecteur normal sortant à la frontière. Soit $T_n = \{(x_1^1(n), 0..0); (x_2^1(n), 0..0)), \quad (0, x_1^2(n), 0..0); (0, x_2^2(n), 0..0))\}$ avec $x_i^j(n) \in E$ construit de la manière suivante :*

$$x_1^i(1) = a_i; \ x_2^i(1) = b_i,$$

et

$$x_1^j(n+1) = \begin{cases} x_1^j(n), & si \ \ \frac{\partial}{\partial t_j} f(\frac{(0,\ldots,x_1^j(n)+x_2^j(n),0\ldots 0)}{2}) \geq 0 \\ \frac{x_1^j(n)+x_2^j(n)}{2}, & si \ \ \frac{\partial}{\partial t_j} f(\frac{(0,\ldots,x_1^j(n)+x_2^j(n),0\ldots 0)}{2}) < 0 \end{cases}$$

$$x_2^j(n+1) = \begin{cases} x_2^j(n), & si \ \frac{\partial}{\partial t_j} f(\frac{(0,...,x_1^j(n)+x_2^j(n),0...0)}{2}) \leq 0 \\ \frac{x_1^j(n)+x_2^j(n)}{2}, & si \ \frac{\partial}{\partial t_j} f(\frac{(0,...,x_1^j(n)+x_2^j(n),0...0)}{2}) > 0 \end{cases}$$

alors $\nabla f'((x_1^1(n),...x^l(n))) \to_{n\to\infty} 0$ *et* $d((x_1^1(n),...x^l(n)), \{\bar{x} \ : \ \nabla f(\bar{x}) = 0\}) \to_{n\to\infty} 0$. *De plus*

$$N_\epsilon := \min\{n \ : \ (x_1^1(n),...x^l(n)) \in B(\bar{x},\epsilon))\}, \quad \bar{x} \in \{\bar{x} \ : \ \nabla f(\bar{x}) = 0\}$$

vérifie

$$N_\epsilon \leq |\log_2(\epsilon)| + \log_2(longueur(E)) = \frac{1}{l}\log_2(\int_{B(\bar{x},\epsilon)} dx / \int_E dx).$$

En résumé

Opérateur	Avantage	Inconvénient
Mutation	Trouve avec certitude le min global	lent $O(1/\epsilon^l)$
Crossover	rapide $O(\|\log_2(\epsilon)\|)$	Ne trouve que des min locaux

1.3 Avantages des algorithmes génétiques

1. Optimise avec paramètres discrets ou continus (en dimension grande).

2. Pas besoin de régularité (Newton, Gradient...).

3. Marche lorsqu'il y a un grand nombre de variables.

4. Parallélisable...

1.4 Inconvénients des algorithmes génétiques

1- On ne sait rien démontrer théoriquement (pas tout à fait vrai - chaîne de Markov, processus stochastiques)

2- A chaque problème il y a une façon de concevoir un algorithme génétique (balance entre recherche locale et globale). Les paramètres de mutations et de créations d'individus jouent sur la convergence

1.5 Mise en oeuvre : BE1

Problème 0 : On cherche à maximiser la fonction

$$f : x \in [-1, 1] \mapsto -(x^2(2 + sin(10x))^2) \in \mathbb{R}$$

sur $[-1, 1]$.
1) Quel est le maximum ? Tracer sous matlab la fonction f entre $[-1, 1]$. A t'on un seul maximum local ? global ?
2) On va tester l'efficacité (c'est-à-dire le temps moyen pour atteindre une erreur donnée) de l'algorithme génétique suivant.
Soit :
- $prec$ la précision voulue
- $Npop$ la taille de la population
- p_M la proportion de grande mutation : $individu \leftarrow \mathbb{U}([-1, 1])$
- p_m la proportion de petite mutation de taille e_m autour du meilleur : $individu \leftarrow individu_{max} + U([-e_m, e_m])$
- p_s la proportion de sexe : $individu \leftarrow (individu_{max} + individu)/2$
- on conserve le meilleur à chaque tour ($p_M + p_m + p_s < 1$).

1) Initialisation aléatoire $\mathbb{U}([-1, 1])$ de la population
2) Calcul de $f(Pop)$, trier de la valeur la plus petite à la plus grande.
3a) Mutation, Mutation petite et crossover
3b) Calcul de l'erreur (la fitness)
4) retour à 2) tant que $erreur > prec$.

A) Implanter l'algorithme ci dessus sous la forme d'une fonction (faire une approx. : moyenne th. $\sim$ moyenne empirique)
$(Npop, p_M, p_m, p_s, prec) \mapsto (TempsMoyen, Meilleur_x)$ avec $TempsMoyen$ une approximation du temps moyen d'atteinte à la précision $prec$.

B1) A $Npop = 10$, $p_M = .8$ et $p_m = p_s = 0$. Tracer pour $prec = (10^{-8\frac{1}{M}}, 10^{-8\frac{2}{M}}, ..., 10^{-8\frac{j}{M}}, ..., 10^{-8\frac{M}{M}})$ le logarithme du temps moyen correspondant.
B2) A précision $prec = 10^{-8}$, $Npop = 10$, $p_M = .8$ ajouter des petites mutations $p_m > 0$ et $e_m \in]0, .1[$. Faire quelques tests avec différents (p_m, e_m), observer le résultat (A t'on amélioration de la convergence/$p_m = 0$).
B3) A précision $prec = 10^{-8}$, $p_M = p_m = p_s = (M - 1)/3$, $e_m = .01$ faire varier la taille de la population $Npop = (10^1, 10^2, ...10^5)$. A t'on décroissance du temps moyen lorsque l'on augmente la population ? Est ce que c'est rentable de le faire ?

Problème 0b : On pose $A =^t BB$ (avec $B = rand(n, n)$) une matrice définie positive, minimiser (resp. maximiser) $f(x) = ({}^t xAx)/({}^t xx)$ avec $x \neq 0$.

Problème 1 : On considère un jeu à deux joueurs dont les règles sont les suivantes :
- en même temps le joueur 1 et le joueur 2 annonce "pair" ou "impair"
- si les deux joueurs ont dit "pair" OU si les deux joueurs ont dit "impair", le joueur 1 gagne 1 euro et le joueur 2 perd 1 euro
- si l'un a dit "pair" et l'autre impair, le joueur 2 gagne 1 euro et le joueur 1 perd 1 euro

Une stratégie pour le joueur 1 est de jouer avec probabilité p_1^1 "pair" et p_2^1 "impair"

Une stratégie pour le joueur 2 est de jouer avec probabilité p_1^2 "pair" et p_2^2 "impair"

Le but est de trouver p_1^1, p_2^1, p_1^2 et p_2^2 qui maximise l'espérance de gain pour chacun des joueurs.

1) Calculer l'espérance de gain pour le joueur 1 lorsque :
- le joueur 1 joue avec probabilité p_1^1 "pair" et p_2^1 "impair"
- le joueur 2 joue avec probabilité p_1^2 "pair" et p_2^2 "impair"

Calculer l'espérance de gain pour le joueur 1 lorsque :
- le joueur 1 joue avec probabilité p_1^1 "pair" et p_2^1 "impair"
- contre une population de $s = 1..N_2$ où le joueur $\{2, s\}$ joue avec probabilité $p_1^{2,s}$ "pair" et $p_2^{2,s}$ "impair"

2) On considère l'algorithme génétique suivant :
α) Créer une population 1 (respectivement 2) de taille N_1 (resp. N_2) ayant des stratégies choisit au hasard
β) Trouver $rang_1^{max,0}$ (resp. $rang_2^{max,0}$) l'individu de la population 1 (resp. 2) ayant la meilleure espérance de gain face à la population 2 (resp. 1).

γ) "Au temps k"
- Trouver $rang_1^{max,k}$ (resp. $rang_1^{min,k}$) l'individu de la population 1 ayant la meilleure (resp. la pire) espérance de gain face à l'individu $rang_2^{max,k-1}$ de la population 2.
- Trouver $rang_2^{max,k}$ (resp. $rang_2^{min,k}$) l'individu de la population 2 ayant la meilleure (resp. la pire) espérance de gain face à l'individu $rang_1^{max,k-1}$ de la population 1.
- Dans les deux populations, on élimine le pire individu et on le remplace par une mutation du meilleur.
- On incrémente k de 1 et on recommence γ tant que $k < temps_{max}$

Implanter cet algorithme sur ce Problème, et étudier sa convergence pour différentes valeurs de $N_1 = N_2$ et $temps_{max}$. Qu'observe t'on (voir les graphes de convergence) ? Peut on l'expliquer ?

3) Faire la même chose pour le jeu de pierre papier ciseaux. Quel est la meilleure façon de jouer ?

Problème 2 : Jeu de Poker Le jeu comporte deux joueurs. Chacun tire un nombre entre 0 et 1 de manière aléatoire (suivant une loi uniforme) de manière indépendante.

Les stratégies possibles du joueur 1 sont $\{parier,\ ne\ pas\ parier\}$.
Les stratégies possibles du joueur 2 sont $\{appeler,\ se\ coucher\}$.

α) Si le joueur 1 parie et le joueur 2 se couche ALORS le joueur 1 gagne 1 euro et le joueur 2 perds 1 euro.

β) Si le joueur 1 parie et le joueur 2 appelle ALORS :
- si $X > Y$ alors le joueur 1 gagne $1 + B$ euros et le joueur 2 perds $1 + B$ euros (B est le montant du pari).

- si $X < Y$ alors le joueur 2 gagne $1 + B$ euros et le joueur 1 perds $1 + B$ euros (B est le montant du pari).

- si $X = Y$ alors le jeu est nul (ni gain, ni perte).

γ_{Borel}) Si le joueur 1 ne parie pas ALORS le joueur 2 gagne 1 euro et le joueur 1 perds 1 euro

Pour simplifier le Problème :

- on considère qu'une stratégie pour le joueur 1 est une variable aléatoire $S1(X)$ qui renvoie *parier* avec probabilité $p(X)$ et *ne pas parier* avec probabilité $1 - p(X)$

- on considère qu'une stratégie pour le joueur 2 est une variable aléatoire $S2(Y)$ qui renvoie *appeler* avec probabilité $q(Y)$ et *se coucher* avec probabilité $1 - q(Y)$

p et q sont des fonctions en escaliers (soit des vecteurs en matlab), i.e.

$$p(X) = p_i, \quad si \;\; X \in [ih, (i+1)h]$$

$$q(Y) = q_i, \quad si \;\; Y \in [ih, (i+1)h]$$

avec $h = 1/10$ par exemple.

1) Trouver le vecteur de meilleure stratégie pour le joueur 1 (et le joueur 2).

2) Même question pour le jeu avec la condition,

γ_{VN}) Si le joueur 1 ne parie pas ALORS :
- si $X > Y$ alors le joueur 1 gagne 1 euro et le joueur 2 perds 1 euro.

- si $X < Y$ alors le joueur 2 gagne 1 euro et le joueur 1 perds 1 euro.

- si $X = Y$ alors le jeu est nul (ni gain, ni perte).

Problème 3 : Un texte contenant des mots fraçais (non accentués et de longueur de 5 lettres au plus) a été codé en permutant les 26 lettres de l'alphabet. On se donne cinq dictionnaires de mots de 1 à 5 lettres respectivement.

Le script Matlab : "LectureFichierTexte.m" permet de charger en mémoire les dictionnaires de 1 à 5 lettre en code ASCII (c'est à dire $a \mapsto 97$... $z \mapsto 122$ et "espace=32") et le texte codé dans la $TexteCodeEnASCII$ de taille (nombre de mots $\times$ 5).

Par soucis de simplicité on crée 5 matrices contenant les mots de 1 à 5 lettres du texte codé.

On pourra utiliser/modifier la fonction $distanceTextePourUnePermutation(P)$ qui a une permutation des lettre (c'est à dire P est une permutation du vecteur [97 : 122] (représentant le code ASCII)) renvoie la distance du texte codé dont on a permuté les lettres aux dictionnaires des mots existant.

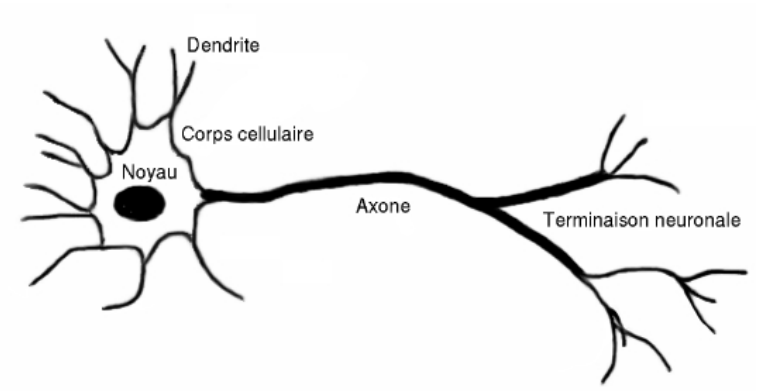
On pourra utiliser/modifier la fonction $AffichageTexte(P, Texte)$ permet d'afficher un texte (texte codé) ayant subit une permutation de lettre P.

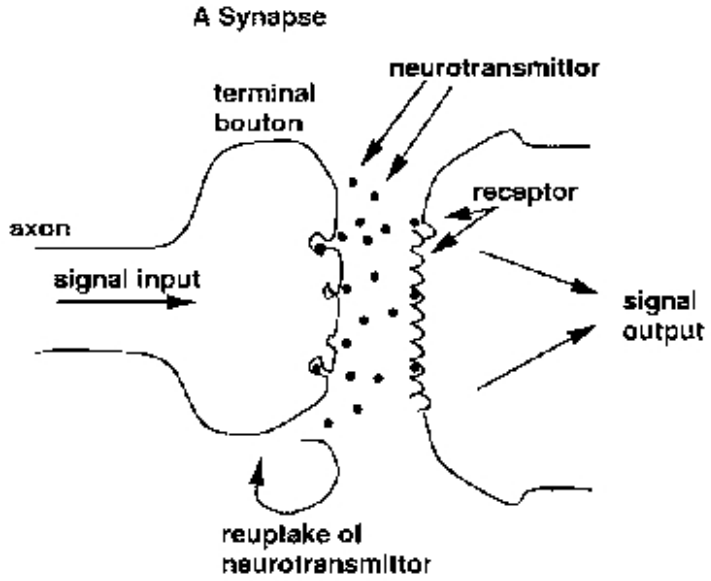
On pourra utiliser/modifier la fonction $[P] = PermuationAleatoire(W)$ qui échange deux éléments de W pour créer la nouvelle permutation P.

1) Essayer de retrouver la phrase avant codage.

2 Réseaux de Neurones

2.1 Principe "Biologique"





Définition 2.1 *Un* **réseau de neurones** *est constitué de* **neurones** *ayant un certain* **niveau d'activation** *: cela peut être un bit* $\{0,1\}$*, un réel* $[0,1]$*... Les neurones sont* **connectés** *entre eux et le neurone* i *a une influence sur le neurone* j *avec* **taux d'influence** $a_{ij} \in \mathbb{R}$*.*

Définition 2.2 *Un* **réseau de neurones** *est* **synchrone** *si au temps* $t+1$ *le niveau d'activation de tous les neurones est calculé à partir des niveau d'activation et des taux d'influence du temps* t *:* $(p_{ij})_{i,j}$*, c'est-à-dire, pour le neurone* i *:* e_i*, son niveau d'activation est de la forme*

$$e_i(t+1) = f_i\Big((p_{ij})_{i,j}; (e_j(t))_j\Big).$$

avec f_i *sa* **fonction d'activation**.

Définition 2.3 *Un* **réseau de neurones** *est déterministe ou probabiliste suivant que* e_i *est une variable aléatoire ou non.*

2.2 McCulloch and Pitts (1943)

Le modèle mathématique de réseaux de neurones de McCulloch and Pitts (1943) est à base logique. Un neurone prend la valeur 0 ou 1 (vrai/faux)

$$e_i = \begin{cases} 1, & si \ \sum_{j \in \mathcal{C}_i^{excitateur}} e_j > \theta_i \quad et \quad \sum_{j \in \mathcal{C}_i^{inhibiteur}} e_j = 0 \\ 0 & sinon \end{cases}$$

Le paramètre θ_i est le seuil d'activation du neurone i. Les inhibiteurs permettent d'avoir un "NON" logique. On note que les neurones activateurs ont la même influence sur le neurone i.

2.3 Donald Hebb (1940) - Franck Rosenblatt (1957)

- Donald Hebb (1940), sur l'apprentissage non supervisé (adaptation neuronale pendant le processus d'apprentissage) est pris en compte par la modification des coefficients de pondération.

- Franck Rosenblatt (1957) : le modèle du perceptron. C'est le premier système artificiel capable d'apprendre par expérience, y compris lorsque son instructeur commet quelques erreurs.

$$\tilde{e}_i = \begin{cases} 1, & si \ \sum_{j \in \mathcal{C}_i} p_{ji} e_j - \theta_i \geq 0 \\ 0 & sinon \end{cases}$$

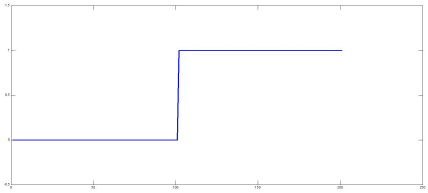
$$p'_{ji} = p_{ji} + \text{Pas d'apprentissage}((\tilde{e}_i)^{\text{théorique}} - \tilde{e}_i).e_i$$

2.4 Marvin Lee Minsky et Seymour Papert (1969)

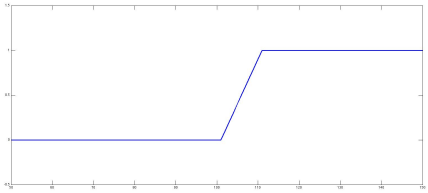
Timitations théoriques du Perceptron, et plus généralement des classifieurs linéaires, notamment l'impossibilité de traiter des problèmes non linéaires ou de connexité.

2.5 Fonctions d'activations

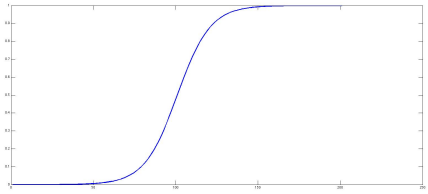
Type de réponse possible : **SEUIL** :
- Si la quantité est trop faible il n'y pas de réponse ($y = 0$)
- Si elle est trop forte il y a saturation ($y = 1$)

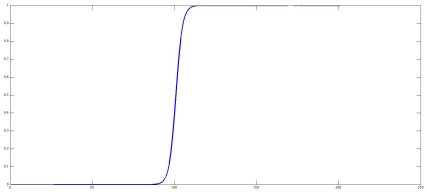


Type de réponse possible : **LINEAIRE** saturée
- Si la quantité est trop faible il n'y pas de réponse ($y = 0$)
- Si elle est trop forte il y a saturation ($y = 1$)
- Entre les deux, il y a une réponse proportionnelle à la quantité de messagers ($y = C(w1x1 + w2x2)$)



Type de réponse possible : **SIGMOIDE**
- Si la quantité est trop faible il n'y *presque* pas de réponse ($y \sim 0$)
- Si elle est trop forte il y a saturation ($y \sim 1$)
- Entre les deux, il y a une réponse plus ou moins marquée (paramètre de la sigmoïde)





Type de réponse possible : **PROBABILISTE**
- $y = 1$ avec une probabilité p
- $y = 0$ avec une probabilité $1 - p$

La probabilité dépend de la quantité de messagers

2.6 John Joseph Hopfield (1982)

John Joseph Hopfield (1982), modèle récurrent (temps n)

$$e_i^{n+1} = \begin{cases} 1, & si \ \sum_{j\in\mathcal{C}_i} p_{ji} e_j^n \geq \theta_i \\ -1 & sinon \end{cases}$$

$(p_{ij})_{i,j}$ symétrique, positive !

permet de définir une énergie (fonction de Lyapunov)

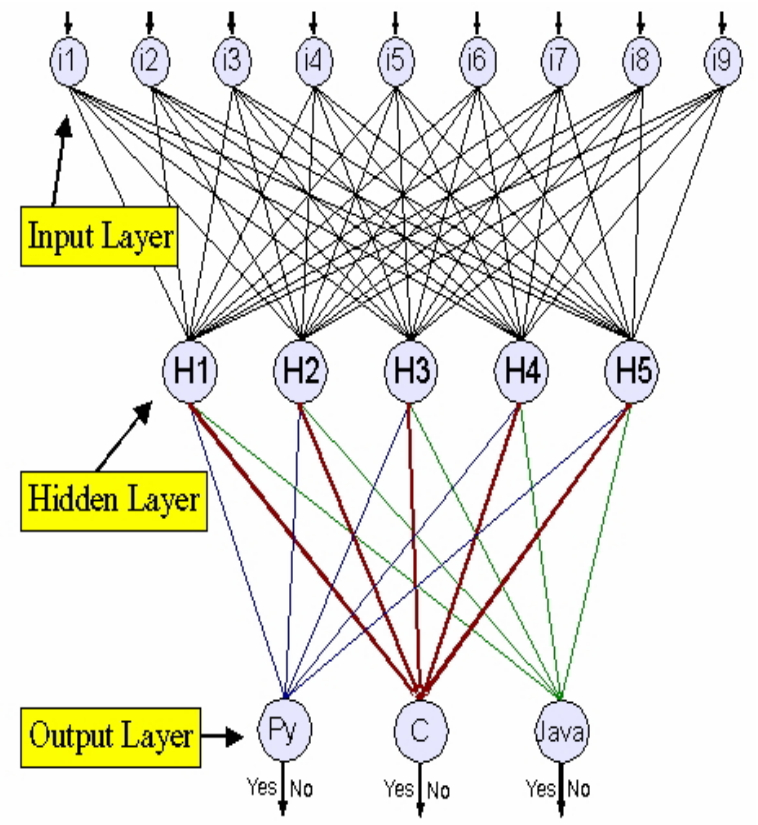
$$E = -\frac{1}{2}\sum_{i,j} p_{ij} e_i^n e_j^n + \sum_i e_i^n \theta_i$$

CONVERGENCE du réseau

2.7 Perceptron Multi-Couche

Epoque :fin-20ème : Werbos, Rumelhart 1986, le Perceptron Multi-Couche rétropropagation du gradient de l'erreur dans des systèmes à plusieurs couches, chacune proche du Perceptron.

Dans un réseau de neurones, il y a "beaucoup" de neurones interconnectés.
i) Les neurones qui reçoivent l'information de l'extérieur
ii) Des neurones "cachés" qui traitent l'information
iii) Les neurones qui renvoient la réponse finale



Un réseau de neurones est caractérisé par :

- une temporisation des événements : ex : temps discret - sans retard
- N neurones
- e_i charge électrique du neurone i
- p_{ij} la pondération de la quantité de messager allant du neurone i vers le neurone j
- ϕ le type de fonction réponse : Seuil ; Linéaire ; sigmoïde ; Probabiliste.

$$e_k(temps\ +1) = \phi(\sum_j p_{jk} e_j(temps)).$$

- apprentissage : les poids w_{ij} sont modifiés si la réponse en sortie est "mauvaise"

$$e_k(temps\ n) = \phi(\sum_j p_{jk}(RESULTAT)\rceil_k(temps\ n-1)).$$

2.8 Applications

a) Reconnaissance de Forme

b) Traitement d'image

c) Transformations mathématiques (ex : transfo de Fourier, traitement du signal)

d) Prédiction et contrôle

e) psychologie, linguistique...