CHAPITRE 14

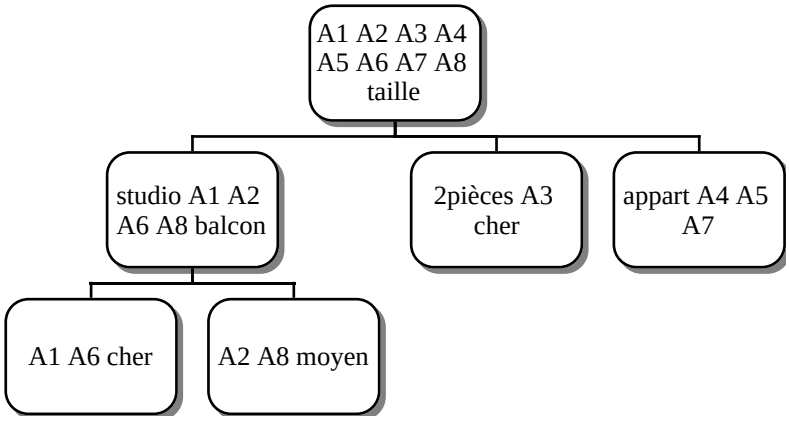
PROBLEMES D'APPRENTISSAGE

L'apprentissage est bien sûr une des grandes questions de l'intelligence artificielle, comment trouver des "bonne règles" à partir d'exemples et de contre-exemples ? Comment reconnaître un motif dans une chaîne en vue de classer des codes ? Comment enseigner une fonction mathématique à l'aide de quelques exemples seulement ? Comment un programme pourrait-il lui même évaluer ses performances afin de se corriger ? Autant de questions qui ont déjà donné lieu à plusieurs approches notamment la classification automatique, le connexionisme et les algorithmes génétiques.

Aux origines de l'apprentissage : le dichotomiseur itératif ID3

Le principe de cet algorithme [Quinlan 79], est de créer un arbre de classification à partir d'un ensemble d'exemples dont chacun est une liste de descripteurs. Prenons par exemple 8 appartements caractérisés par :

Exemple	Ensoleillement	Taille	Balcon	Prix
A1	non	studio	oui	cher
A2	oui	studio	non	moyen
A3	oui	2 pièces	oui	moyen
A4	non	appartement	oui	moyen
A5	oui	appartement	oui	moyen
A6	oui	studio	oui	cher
A7	oui	appartement	non	moyen
A8	non	studio	non	moyen

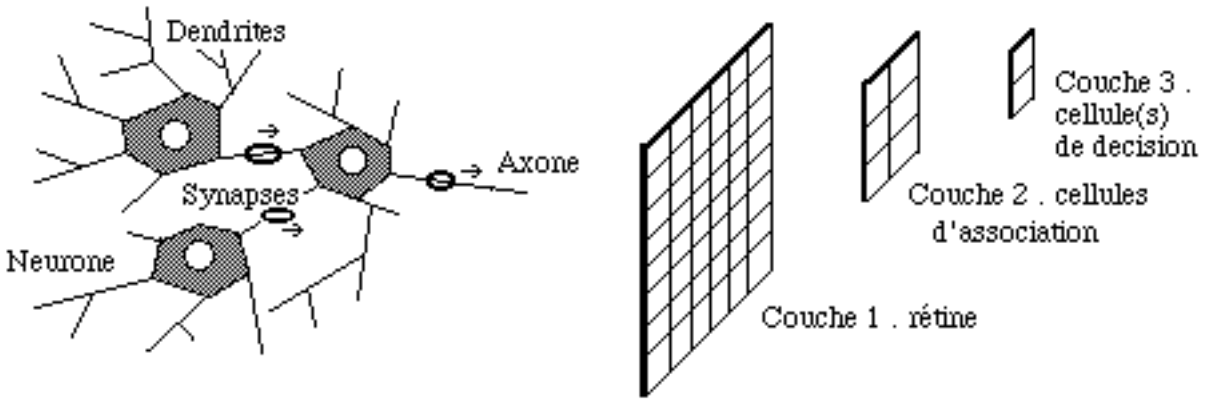


On crée un arbre de décision en choisissant l'un des attributs (par exemple la taille) avec autant de fils qu'il y a de valeurs (ici studio, 2pièces, appart) et on recommence pour chaque noeud tant que la partie des exemples concernés par ce noeud n'est pas uniforme du point de vue d'un attribut fixé à l'avance, (par exemple on veut des règles déterminant des tranches de prix). L'inconvénient est que la forme de l'arbre dépend des choix successifs et reste limité à des problèmes de classification assez simples.

En fait il faut calculer la séquence d'attributs commune à tous les chemins partant de la racine, permettant d'obtenir l'arbre minimal. Pour des données symboliques, l'algorithme est facile à implémenter et donne de bons résultats. On peut décider à chaque noeud de l'arbre, de sélectionner le prochain attribut comme celui qui apporte le plus d'information (entropie minimale) I(a) = - log p(a), donc l'attribut "a" qui a la probabilité minimale.

Le connexionisme avec les réseaux de neurones formels multicouches

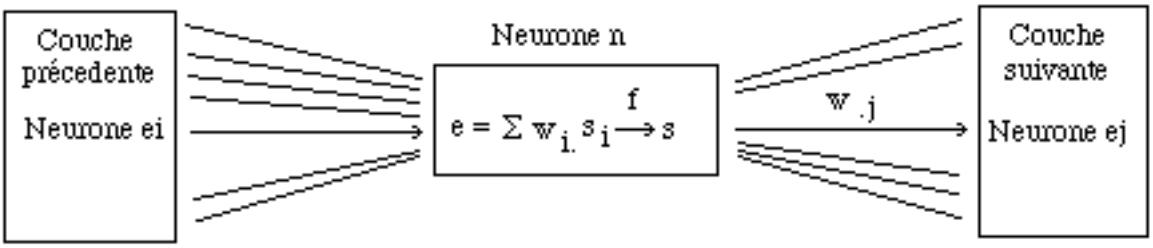
Les réseaux de neurones sont des schématisations de cerveaux simplifiés, dont on postule qu'ils sont constitués d'un très grand nombre d'unités simples en interaction. L'un des modèles les plus séduisants et les plus utilisés pour la reconnaissance des formes, est celui du réseau à couches. Dans chacune des couches, chaque neurone est relié à ceux de la couche précédente dont il reçoit les informations, et à chaque neurone de la couche suivante à qui il transmet des informations, mais il n'est pas relié aux autres neurones de sa propre couche. On considère que chaque neurone reçoit par ses "dendrites" une certaine activation électrique dont une somme pondérée constitue l'entrée e (un certain poids $w_{i,j}$ sera affectée à la liaison entre les neurones i et j). Par suite, le neurone passe à un certain "état" s = f(e) qui sera la mesure de sa sortie portée par son "axone" (f peut être modelisée par une simple fonction de seuil ou par une fonction du type arctan ou th).
L'axone transmet ensuite cette valeur par l'intermédiaire de "synapses" aux dendrites d'autres neurones. (Les poids mesurent en fait l'efficacité des synapses.)
L'apprentissage du reseau, consiste en une modification de ces poids au fur et à mesure des expériences, c'est à dire de la confrontation entre le vecteur sortant de la dernière couche et celui qui est attendu en fonction d'un vecteur d'entrée.



Algorithme de rétropropagation [Rumelhart 86] [Le Cun 87]

Dans un réseau multicouche, chaque neurone n, à l'intérieur, ayant $e = \sum w_{i,n} s_i$ pour entrée, où i parcourt les neurones de la couche précédente, aura une sortie s = f (e). Lorsque l'on propose le vecteur $X= (x_1, x_2, ...,x_m)$, à la première couche, la dernière restitue le vecteur $S = (s_1,..., s_p)$ alors qu'on attend $Y = (y_1,...,y_p)$ comme réponse.
Le but de cet algorithme est d'exprimer l'erreur quadratique $E = \sum_{1 \le i \le p} (y_i - s_i)^2$ et de chercher à la minimiser en modifiant chaque poids w, suivant l'influence qu'il a sur E : $w(t+dt) - w(t) = -\mu . \partial E/\partial w$ où μ est un coefficient positif, le "pas" du gradient. En effet, si cette dérivée est nulle ou faible, cela veut dire que l'erreur E dépend peu de ce poids là w et donc qu'il n'y a pas lieu de le modifier. Si, elle est positive et importante, cela signifie au contraire que E croît avec w, donc on diminue w d'autant.
On cherche donc à exprimer ce gradient qui est le vecteur formé par tous les $\partial E/\partial w$
Plaçons nous entre le neurone i d'une couche et un neurone n fixé :
Pour la commodité de la lecture, on notera i l'indice parcourant la couche précédente, et j celui de la couche suivante, si d_n est la dérivée partielle de E par rapport à e, on a :

$$\frac{\partial E}{\partial w_{i,n}} = \frac{\partial E}{\partial e}.\frac{\partial e}{\partial w_{i,n}} = \frac{\partial E}{\partial e}.\frac{\partial \sum_j w_{j,n}.s_j}{\partial w_{i,n}} = \frac{\partial E}{\partial e}.s_i = d_n.s_i$$



Car tous les termes de la sommation pour $j \neq i$ ne dépendent pas de $w_{i,n}$.
Si le neurone n'est pas en sortie, alors on calcule :

$$d_n = \frac{\partial E}{\partial e} = \sum_{j \text{ suivant}} \frac{\partial E}{\partial e_j}.\frac{\partial e_j}{\partial e} = \sum d_j \frac{\partial e_j}{\partial s}.\frac{\partial s}{\partial e} = \sum d_j . w_{n,j}.f'(e)$$

Et si n est en sortie :

$$d_n = \frac{\partial E}{\partial e} = \frac{\partial \sum_{j=1}^{p} (Y_j - S_j)^2}{\partial e} = 2(S_n - Y_n).f'(e)$$

La règle d'apprentissage est donc à chaque présentation d'exemple X, Y, de mesurer la sortie S, l'erreur E, et de modifier chaque poids (de i à j) $w_{i,j}$ en le remplaçant par $w_{i,j} - \mu d_j s_i$ avec $d_n = (\sum d_j w_{n,j}) f'(e_n)$ pour les indices j en aval de n, sauf si n est en sortie auquel cas on a plutôt $d_n = 2(s_n - Y_n) f'(e_n)$. Il y a rétro-propagation de l'erreur commise dans la mesure où les modifications vont devoir être effectuées de la dernière couche vers la première.
Ce processus est répété sur plusieurs exemples jusqu'à ce qu'il y ait convergence suivant un seuil fixé.

Résultats : Ils sont encore très expérimentaux, il se peut que des exemples présentés au début influencent trop le réseau, et que des exemples présentés tardivement ne puissent être appris facilement. Mais ce qui arrive aussi souvent, est que le réseau a tendance à se conformer au dernier exemple présenté. C'est pourquoi on peut modifier la procédure d'apprentissage en présentant successivement tous les exemples avec une seule rétropropagation à chaque fois et si l'erreur est supérieure au seuil au moins une fois au cours de ce balayage, on refait un balayage complet.
Des résultats intéressants ont été obtenus surtout en reconnaissance des formes, prédiction de consomnes ou voyelles dans un mot, etc ... mais le problème du choix des "bons" exemples et des paramètres du réseau (nombre de neurones par couches) reste entier.

Algorithmes génétiques

Les algorithmes génétiques sont inspirés par les lois de l'évolution en biologie [Goldberg 89], [Holland 93], [Michalewicz 92], pour résoudre des problèmes d'optimisation, les solutions vont être codées sous forme de chaînes de caractères appelées "chromosomes", les caractères étant les "gènes". L'algorithme consiste alors à reproduire par étapes une "population" P de chromosomes.
Au départ on prendra donc des chromosomes plus ou moins arbitraires, générés aléatoirement ou réalisant déjà un semblant de solution intuitive, ou bien de réelles solutions mais non optimales.
De génération en génération, la population doit s'améliorer suivant une fonction d'évaluation dictée par le problème, on classe la population suivant cette fonction pour permettre aux meilleurs chromosomes de se reproduire.

Les transitions entre générations se font par des "opérateurs génétiques" qui sont des fonctions $P \rightarrow P$ ou $P^2 \rightarrow P^2$, ainsi :

La mutation : un gène est remplacé par un autre aléatoirement choisi AB**C**BBAC → AB**E**BBAC
La transposition : deux gènes aléatoirement choisis sont permutés AB**C**EB**A**CD → AB**A**EB**C**CD
L'inversion : on permute autour d'une coupure choisie ABCB**DEA** → **DEA**ABCB
L'inversion partielle : une partie du chromosome est inversée ABCD**EFD**CD → ABCD**DFE**CD
Le cross-over, deux parents engendrent deux enfants en échangeant une partie choisie aléatoirement : AA**BED**ABC, DA**CBC**DAE → AA**CBC**ABC, DA**BED**DAE
C'est la plus puissante car intuitivement cela correspond à deux solutions comportant chacune une bonne partie de la solution, aussi en regroupant ces parties peut-on améliorer la solution globale.
D'autres opérateurs peuvent être imaginés mais ils doivent respecter la cohérence dans la représentation du problème en chromosomes, (exemple : respecter un aspect injectif ou bijectif...)
A chaque étape ces opérateurs sont tirés et appliqués au hasard suivant des probabilités à convenir, on peut alors prendre les chromosomes et leurs images et éliminer les plus mauvais (ce qui fait par exemple deux suppressions pour quatre individus dans le cas du cross-over) ou alors trier toute la population et sa descendance et ne conserver que les meilleurs en assurant à P un cardinal constant.

Exemple, dans le problème classique du voyageur de commerce, une solution est une permutation de toutes les villes, ABCDEF par exemple s'il y en a 6. La fonction d'évaluation à minimiser est alors la somme des distances, mais les transitions devront être adaptées de façon à ce que les "solutions" restent toujours bijectives sur l'ensemble des villes.

14-1° Programmer l'algorithme ID3

14-2° L'algorithme des k plus proches voisins, il s'agit de classer des points donnés dans R^n suivant des classes déjà données (il ne s'agit donc pas de trouver les classes, ce qui est un vrai problème d'apprentissage, mais de classer de nouveaux candidats). La méthode est très simple mais lente, elle consiste à chercher pour le nouveau point à classer, quelle est la classe la plus représentée parmi les k plus proches voisins de ce nouveau point. Le problème est de déterminer un k suffisamment petit, on pourra constater sur un exemple dans R^2, que si on prend 3 ou 5 pour k, les résultats peuvent être différents.

14-3° Méthode des centres mobiles, on donne un ensemble de points de R^n et le nombre k de classes voulues. On choisit aléatoirement des centres G_1, G_2, ... G_k, puis on place dans la classe i, tous les points plus proches de G_i que des G_j pour $j \neq i$.
On recalcule le centre G_i obtenus pour cette partition.
On recherche à nouveau la partition autour de ces nouveaux centres et ainsi de suite jusqu'à stabilisation relativement à un certain seuil, des G_i.
Prendre un exemple d'une cinquantaine de points du plan et programmer la classification en en 5, 8 ou 10 classes par exemple en les séparant.

14-4° Classification automatique, on donne un ensemble de points de R^n et la matrice de leurs distances deux à deux. En procédant par itérations avec un incrémént d_0, on regroupe en classes tous les éléments p, q tels que $d(p, q) < d$ ($d = d_0$ initialement puis d incrémenté de d_0 à chaque étape). Le nombre de classes diminue donc à chaque fois.
Si, à chaque étape, on cherche les centres de gravités G_i de ces classes et $I = \sum d^2(G_i, G)$ si G est le centre de tout le nuage.
On peut chercher l'étape (donc le nombre de classes) minimisant l'inertie I, car si I minimum $I' = \sum d_i d'_i = 0$ signifie que chaque centre est stabilisé par rapport à G. Programmer cette méthode et l'expérimenter sur un nuage de points du plan.

14-5° Un générateur de règles. Cette méthode est issu d'une technique d'apprentissage exposée dans [Ganascia 87] le but est d'obtenir des règles du type (H_1 et H_2 et H_3 ...) ou (H'_1 et H'_2 ...) ou ... --> C_1 et C_2 et... à partir d'exemples donnés par des descripteurs de caractères donnés toujours dans le même ordre.
Soit l'exemple où n = 3 individus sont donnés avec des valeurs simples aux attributs respectifs : taille, couleur de cheveux, couleur des yeux, (nature de peau et rhésus sanguin par la suite). A = {e_1 = petit & brun & bleus, e_2 = petit & roux & bleus, e_3 = grand & blond & bleus}
Cube de Hilbert : c'est l'ensemble $\{0, 1\}^n$ muni de l'ordre partiel (a_1, a_2, ... a_n) ≤ (b_1, b_2, ... bn) ⇔ Pour tout i, $a_i \leq b_i$. C'est trivialement un treillis de Boole. On associe à chaque exemple un sommet sur l'un des axes. Le dessiner pour l'exemple donné.
Soit Cn le cube des exemples de dimension n, et si D est l'ensemble des d descripteurs (ici d = 6), Cd est le cube des descripteurs. Un ensemble d'apprentissage peut être représenté par A, D et une fonction de A vers P(D), mais on va donner une série de fonctions traduisant le problème et permettant surtout d'obtenir les règles induites par les exemples.
1) A chaque sommet du cube Cn, on associe par ß sa description réduite faites des descripteurs non présents dans un sur-sommet, c'est à dire dont il ne pourrait hériter, mais dont il fait hériter ses fils. A partir de n exemples on va donc constituer moins de d sommets où la fonction ß n'est pas vide. Ainsi ß(0 0 1) = grand & blond, ß(1 1 0) = petit.
2) Si chacun des sommets de Cn est représenté par la liste de ses rangs de coordonnées non nulles à partir du rang 0, on aura par exemple (0 2 5) pour le point de coordonnées (1 0 1 0 0 1 0 0 0) ou (2 7) pour (0 0 1 0 0 0 0 0 1). On peut montrer que pour tout descripteur, il existe un et un seul sommet de Cn tel que ce descripteur appartienne à l'image de ce sommet par ß. On note alors ß' cette fonction "réciproque", ainsi : ß'(brun) = (0), ß'(roux) = (1), ß'(petit) = (0 1), ß'(grand) = ß'(blond) = (2) et ß'(bleus) = (0 1 2)

3) On définit à présent la fonction s ∈ Cd → γ(s) = ∧ {ß'(d) / d présent dans s} ∈ Cn (l'intersection). Par exemple si les descripteurs sont ordonnés suivant leur ordre d'apparition dans l'énoncé on aura petit, brun, bleus, roux, grand, blond pour les sommets 100000, 010000, ... , 000001 de Cd avec d = 6. Le sommet 001110 réunit donc les descripteurs petit, bleus et roux dont les images par ß' sont (0 1 2), (0 1) et (1) dont l'inf dans le treillis est (1).
4) On souhaite donner des règles du type d1 & d2 ... & dk → dp, ce qui signifierait que chaque exemple contenant les d1... dk contient dp, il faut donc calculer l'ensemble des dp tels que γ({d1 ... dk}) ≤ ß'(dp). On définit donc pour un sommet s de Cn, Ψ(s) le sommet de Cd correspondant à l'ensemble des descripteurs dont l'image par ß' est supérieure à s. Ainsi avec un certain nombre d'abus d'écriture en utilisant les deux repérages Ψ(001) = Ψ(2) = {blond, bleus, grand ont un ß' ≥ 2} = (001011) = (2 4 5). On calcule Ψ en explorant les descripteurs.

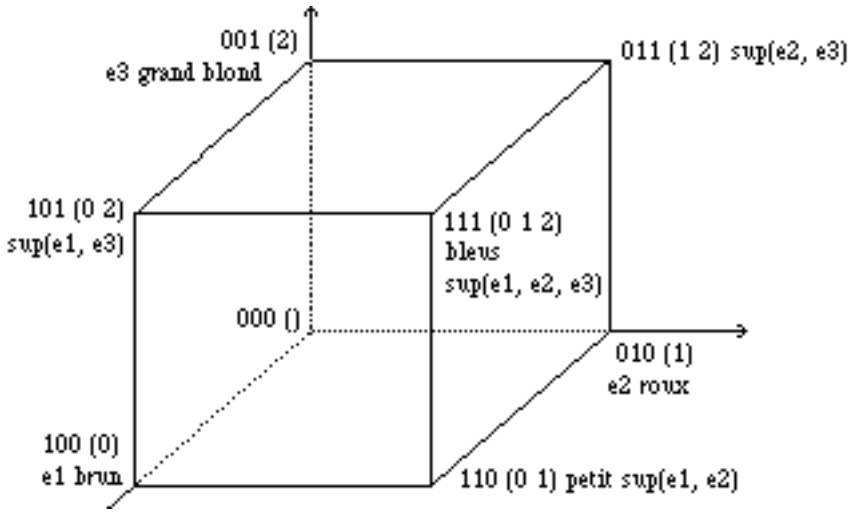
5) On note maintenant la composée ϕ = Ψ °γ qui est donc une fonction de Cd dans lui-même.

ϕ (involutive et croissante) va donner les règles, mais afin d'éviter les redondances on pose Ω

défini par : Ω(s) = ϕ(s) - V{phi (s') / s' < s} = ϕ(s) - V{Ω(s') / s' < s}.
Les règles seront grossièrement : "si descripteurs de s alors ceux de Ω(s)"

6) Algorithme de construction de Ω : comme Ω est presque partout vide, et que Ω(s) = ϕ(s) - V{Ω(s') / s' < s}, on a intérêt à construire Ω suivant un parcours de Cd (simulé par le paramètre F) qui va aller dans un ordre compatible avec <.
On explore le cube Cd en partant des sommets les plus proches de l'origine :
Initialisation de F à (())
Tant que F non vide faire s reçoit la tête de F F modifié en queue de F
i_0 = si s vide alors 0 sinon 1 + dernier (s)
pour i = i_0 à card(D) - 1
faire ss est s concaténé avec la liste (i) {on a toujours s < ss}
si γ(ss) ≠ () alors calcul de phi (ss) puis de Ω(ss) et si celui-ci n'est pas vide le ranger dans l'ensemble des règles, enfin, mettre ss en queue de F

Programmer β, φ, ϕ, ω et l'appliquer sur les exemples



Petites fonctions

```
(de inter (M N) (cond     ((null M) nil )
                          ((member (car M) N) (cons (car M) (inter (cdr M) N)))
                          (t (inter (cdr M) N))  ))
(de inclu (L M) (cond     ((null L) t)
                          ((member (car L) M) (inclu (cdr L) M))
                          (t nil)  ))
(de diff (L M) (cond      ((null L) L)
                          ((member (car L) M) (diff (cdr L) M))
                          (t (cons (car L) (diff (cdr L) M))) ))
```

Un ensemble LE d'apprentissage est une liste de listes de descripteurs. On constitue d'abord la liste LD des descripteurs présents grâce à "descrip" :

```
(de descripbis (L LD) (cond
        ((null L) (reverse LD))
        ((null (car L)) (descripbis (cdr L) LD))
        ((member (caar L) LD) (descripbis (cons (cdar L) (cdr L)) LD))
        (t (descripbis (cons (cdar L) (cdr L)) (cons (caar L) LD)))  ))
(de descrip (LE) (descripbis LE nil))
```

La fonction ß' réalisée par "beta" pour un descripteur d et un ensemble d'exemples LE, donne le sommet maximal ayant ce descripteur. On parcourt les exemples en notant (à partir de 0) les indices des exemples contenant d.

```
(de betabis (d L k CO) (cond      ((null L) (reverse CO))
                                  ((member d (car L)) (betabis d (cdr L) (+ 1 k) (cons k CO)))
                                  (t (betabis d (cdr L) (+ 1 k) CO))  ))
(de beta (d LE) (betabis d LE 0 nil))

;La fonction gamma pour une liste de numéros de descripteurs donne l'intersection des ß' de ces descripteurs
(de gam2 (CO LD LR LE) (if (null CO) LR (gam2 (cdr CO) LD (inter (beta (nth (car CO) LD) LE)LR) LE)))
(de gam1 (CO LD LE) (gam2 (cdr CO) LD (beta (nth (car CO) LD) LE) LE))
(de gam (CO LE) (gam1 CO (descrip LE) LE))
```

On peut montrer que γ est un homomorphisme échangeant inf et sup.
Psi pour un sommet SE du cube des exemples est le sommet du cube des descripteurs union des descripteurs ayant un beta supérieur à SE :

```
(de psibis (SE L R k LE) (cond
        ((null L) (reverse R))
        ((inclu SE (beta (car L) LE)) (psibis SE (cdr L) (cons k R) (+ 1 k) LE))
        (t (psibis SE (cdr L) R (+ 1 k) LE))  ))
```

```
(de psi (SE LE) (psibis SE (descrip LE) nil 0 LE))
(de phi (SD LE) (psi (gam SD LE) LE))
(de omega (LE) ;  La fonction principale omega construit pour l'instant une liste de règles
(let ((LD (descrip LE))  (F '(())) (R nil))     (set 'd (length LD))
     (while (not (null F))  (set 'S (car F))  (set 'F (cdr F))  (set 'i (if (null S) 0 (1+ (car (last S))))) (set 'k i)
          (while (< k d)     (set 'SS (append S (list k))) (set 'GM (gam1 SS LD LE))
                             (if (null GM) nil (set 'OM (reduire SS (diff (psibis GM LD nil 0 LE) SS) R))
                                               (if (null OM) nil (set 'R (cons (cons SS OM) R))
                                                                 (vue SS LD) (prin '" -> ") (vue om LD) (terpri))
                                               (set 'F (append F (list SS))) )
                             (set 'k (1+ k)) )) R))         ; valeur retournée, fin des deux boucles
(de vue (L LD) (if (null (cdr L)) (prin (nth (car L) LD)) (prin (nth (car L) LD)) (prin '" et ") (vue (cdr L) LD) ))
(de reduire (hyp conc L) (cond ; L est une liste de règles
          ((null L) conc)
          ((inclu (caar L) hyp) (reduire hyp (diff conc (cdar L)) (cdr L)))
          (t (reduire hyp conc (cdr L)))  ))
```

On pourra chercher des améliorations tant sur la représentation informatique de Ω, que sur la complexité du parcours de Cd (élaguer les sommets où Ω sera vide), que sur l'élimination de règles superflues, le but étant d'avoir un système de règles équivalent, mais minimal.

14-6° Tester sur les deux exemples suivants :

Exemple LE1:

e1 = petit & blond & bleus & clair & positif	
e2 = grand & blond & marrons & mat & négatif	e3 = grand & roux & bleus & clair & positif
e4 = petit & brun & bleus & mat & négatif	e5 = grand & brun & bleus & clair & négatif
e6 = grand & blond & bleus & clair & positif	e7 = grand & brun & marrons & mat & négatif
e8 = petit & blond & marrons & mat & négatif	e9 = grand & blond & marrons & clair & négatif

Exemple LE2 : Sur la planète Zographos de l'étoile Goukouni Houêdeï de la galaxie d'Arne Saknussem, les habitants se signalent par des couleurs, tailles, calvitie ou non, présences ou non de fourrure, poche ventrale ou queue. En fait quelques espèces issues de nombreux croisements et de millénaires d'exterminations mutuelles cohabitent (les mauves à fourrure mordorée et poche ventrale n'ont hélas pas survécu). Une mission d'observation a pu décrire partiellement une vingtaine de sujets dans la table ci-dessous, quelles règles peut-on en tirer et pourrait-on caractériser les différentes espèces ?

(vert petit poils oreilles chauve)	(bleu oreilles grand chauve poche)
(bleu petit oreilles)	(mauve gros queue poils)
(bleu poils gros)	(vert long queue)
(vert oreilles poils)	(queue mauve poche poils gros)
(petit chauve oreilles poils)	(vert chauve long queue)
(poche bleu poils)	(poche bleu grand chauve oreilles)
(vert petit chauve oreilles poils queue)	(bleu poils gros queue)
(vert queue long poils)	(mauve queue poche petit)
(long vert chauve oreilles queue poche)	(long bleu poche)
(bleu grand oreilles chauve poche queue)	(long vert poche queue)

Solution pour LE1, obtenue par (omega LE1), remis en forme :

positif -> bleus et clair	blond et mat -> marrons
marrons -> negatif	blond et negatif -> marrons
mat -> negatif	bleus et grand -> clair
roux -> bleus et clair et positif et grand	bleus et mat -> petit et brun
brun -> negatif	bleus et negatif -> brun
petit et clair -> blond et bleus et positif	clair et marrons -> blond et grand
petit et positif -> blond	clair et negatif -> grand
petit et marrons -> blond et mat	clair et brun -> bleus et grand
petit et negatif -> mat	grand et mat -> marrons
petit et brun -> bleus et mat	marrons et brun -> grand et mat
blond et bleus -> clair et positif	

(omega LE2)

grand -> oreilles et chauve et bleu et poche	chauve et bleu -> oreilles et grand et poche
mauve -> queue	chauve et poche -> oreilles
gros -> poils	chauve et long -> vert et queue
vert et petit -> poils et oreilles et chauve	bleu et long -> poche
vert et poche -> queue et long	poche et gros -> mauve et queue
vert et long -> queue	queue et long -> vert
petit et poils -> oreilles et chauve	petit et poils et queue -> vert
petit et chauve -> poils et oreilles	petit et oreilles et queue -> vert et poils
petit et bleu -> oreilles	petit et chauve et queue -> vert
petit et poche -> mauve et queue	poils et oreilles et queue -> vert et petit
petit et mauve -> poche	poils et chauve et queue -> vert
poils et chauve -> petit et oreilles	poils et bleu et queue -> gros
poils et mauve -> gros	poils et poche et queue -> mauve et gros
poils et long -> vert et queue	oreilles et bleu et poche -> grand
oreilles et poche -> chauve	oreilles et bleu et queue -> grand et poche
oreilles et queue -> chauve	bleu et poche et queue ->oreilles et chauve et grand
oreilles et long -> vert et chauve et poche et queue	

= (((5 7 10) 3 4 6) ((3 5 10) 6 7) ((3 5 7) 6) ((2 7 10) 8 9) ((2 5 10) 9) ((2 4 10) 0) ((2 3 10) 0 1) ((1 4 10) 0) ((1 3 10) 0 2) ((1 2 10) 0) ((10 11) 0) ((7 9) 8 10) ((5 11) 7) ((4 11) 0 10) ((4 7) 3) ((4 5) 3 6 7) ((3 11) 0 4 7 10) ((3 10) 4) ((3 7) 4) ((2 11) 0 10) ((2 8) 9) ((2 4) 1 3) ((1 8) 7) ((1 7) 8 10) ((1 5) 3) ((1 4) 2 3) ((1 2) 3 4) ((0 11) 10) ((0 7) 10 11) ((0 1) 2 3 4) ((9) 2) ((8) 10) ((6) 3 4 5 7))

14-7° Apprentissage d'une loi empirique.
Dans le cas où une loi empirique est donnée par une courbe et où celle-ci est suffisamment régulière, on propose une méthode de recherche d'une expression F(x) par ajustement.
Si les ordonnées sont positives, lorsque la courbe est croissante convexe, ou décroissante concave, on cherchera les exposants réalisant le meilleur ajustement linéaire entre les abscisses X et les ordonnées Y^{α} plutôt avec $0<\alpha<1$ ou bien $Log^{\alpha}(Y)$. Si la courbe est croissante concave, ou décroissante convexe, on essayera $\exp(\alpha Y)$ avec α positif faible puis Y^{α} avec $\alpha > 1$.
A chaque essai le coefficient de corrélation est modifié, il est alors tentant de conjecturer que si celui-ci a augmenté, on se trouve dans la bonne voie. C'est sur cette idée que repose l'algorithme suivant, où f désigne une fonction concave, passant par un unique maximum, et ε un réel positif.

On pose alors E (a,b) = a si $|f(a) - f(b)| < \varepsilon$
E (b, 2b-a) si f(a) < f(b)
E ((a+b)/2, a) sinon

La deuxième clause signifie une "avancée" dans le même sens que (a, b), la troisième, un recul à "mi-chemin". Faire un schéma.
Cette fonction E est définie à condition de rester dans le domaine de f, et elle donne l'abscisse s du sommet de f à ε près. En effet, tant que a et b sont de même côté de s la distance |a-b| reste invariante à chaque appel de E, au bout d'un nombre fini d'appels on obtiendra nécéssairement un encadrement de s. En ce cas après au plus un appel supplémentaire, |a-b| sera divisé par deux. La suite des longueurs de ces intervalles est donc stationnaire pendant un nombre fini de termes consécutifs, puis divisée par deux et à nouveau stationnaire un nombre fini etc, on a donc lim|a-b| = 0 car une suite extraite converge vers 0. Ceci prouve que E est définie à la condition que sa première clause soit vérifiée, mais cette dernière l'est si f est uniformément continue, ce qui constitue une hypothèse raisonnable.

Appliquer cet algorithme à la fonction $\alpha \rightarrow |$ cor (X, Y^{α}) | calculant un coefficient de corrélation linéaire, dont on suppose qu'elle passe par un maximum. Le but étant de trouver le meilleur exposant a tel que Y^{a} soit fonction du premier degré de X.

Fonctions de statistique :

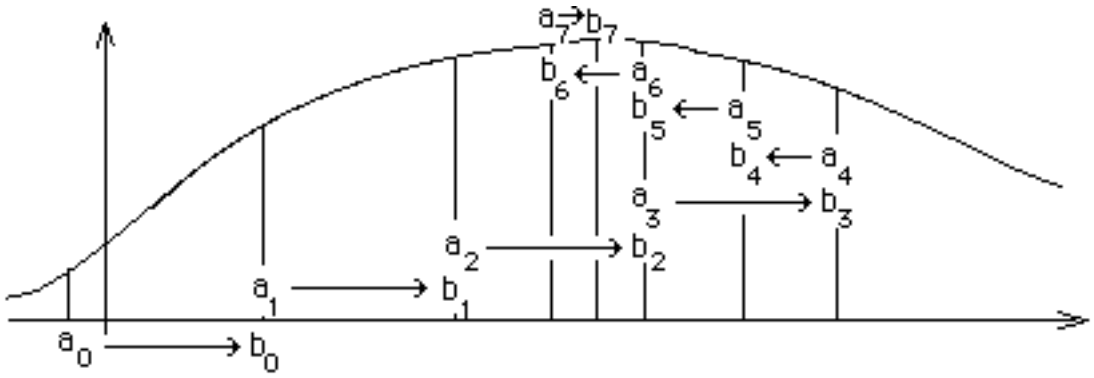
```
(de mbis (L N S) (if (nullL) (// S N) (mbis (cdr L) (1+ N) (+ S (car L))))))
(de moy (L) (mbis L 0 0)) ; donne la moyenne des éléments de la liste L

(de vbis (X Y N S) (if (null X) (// S N) (vbis (cdr X) (cdr Y) (1+ N) (+ S (* (car X) (car Y)))))))

(de cov (X Y) (- (vbis X Y 0 0) (* (moy X) (moy Y))))
; définit la covariance de deux listes non vides et de même longueur
(de var (X) (cov X X))
(de reg (X Y) (D1 X Y (moy X) (moy Y))) ; définit le coefficient de corrélation et les paramètres a et b de Dy/x
(de D1 (X Y MX MY) (D2 (cov X Y) (varX) (var Y) MX MY))
(de D2 (COV VX VY MX MY)
        (D3(// COV (sqrt (* VX VY))) (// COV VX) MX MY))
(de D3 (RO A MX MY) (list RO A (- MY (* A MX))))
(de cor (X Y) (car (reg X Y))) ; ne donne que le coefficient de corrélation des listes X et Y.
```

Fonctions de lissage :

Les sept premiers couples (a, b) sont figurés sur cet exemple :




```
(de ex (A) (subst A 'A (lambda (X) (power X A)))) ; est la fonctionnelle d'exponentiation à la puissance A

(de ptex (X) (power 1.01 X))                    ; est une "petite exponentielle"
(de E1 (X Y A B RA RB) (cond                    ; epsilon est choisi à 0.00001
                (( < (abs (- (car RA) (car RB))) 1e-5 ) (cons A RA))
                (( < (car RA) (car RB)) (E1 X Y B (- (* 2 B) A) RB
                                (reg X (mapcar (EX (- (* 2 B) A)) Y))))
                (t (E1 X Y B (// (+ A B) 2) RB
                                (reg X (mapcar (EX (// (+ A B) 2)) Y))))))
```

Ra Rb désignent des listes (corr a b) A, B sont deux exposants successifs, la première clause signifie que A est un bon exposant.

```
(de E2 (X Y A B) ; donne la liste (exposant corrélation a b)
                (E1 X Y A B (reg X (mapcar (EX A) Y)) (reg X (mapcar (EX B) Y))))

(de E4 (F L M) (if (< (cadr M) (cadr L)) (cons F (E5 L)) (E5 M))) ; test sur les deux coef de corrélation

(de E5 (M) (list  'Y 'puissance (car M) '= (caddr M) 'X+ (cadddr M) <coef= (cadr M) '> ))
```

Fait la comparaison entre les meilleurs résultats entre F(y) à une puissance, et y seul à une puissance et délivre un message en clair.

```
(de cronv (L) (< 0 (* (- (car (last L)) (car L)) (- (// (+ (car (last L)) (carL))2) (nth (div (length L) 2)L)))))
```

Cronv est le prédicat croissant-convexe ou bien décroissant-concave, pour une fonction monotone et convexe ou concave.

```
(de lis (X Y) (if (cronv Y) (E4 'ln (E2  X (mapcar 'log Y)1 2) (E2  X Y .4 .6))
                (E4 'exp (E2 X (mapcar 'exp Y) .1 .2) (E2 X Y 1 2))))
```

"lis" sera donc utilisé pour une liste d'abscisse X, et une liste d'ordonnées Y. Comme dans les recherches empiriques qui suivent on aura surtout affaire à des courbes positives monotones, on peut distinguer les cas suivants :
Si la courbe est croissante et convexe, on peut la rectifier par des transformations du type :

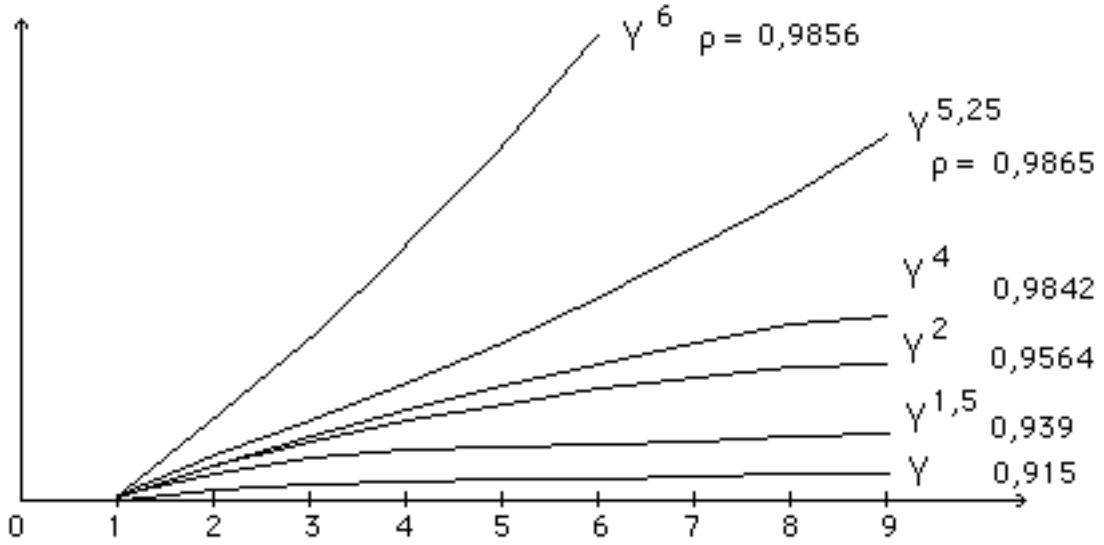
$Y \rightarrow Y^{\alpha}$ avec $0 < \alpha < 1$. Si la courbe est croissante concave, on tentera alors $Y \rightarrow Y^{\alpha}$ avec

$1 < \alpha$, mais en fait il suffit de tenter dans les deux cas, une comparaison des corrélations de Y et Y par exemple, l'algorithme expliqué dans l'énoncé se chargeant d'inventorier des exposants de plus en plus grands ou bien de plus en plus petits.
Si la courbe est décroissante, on testera de même avec des exposants de départ tels que -0,8 et -1,2 par exemple.
Il faut cependant remarquer que le problème de trouver une équation à une courbe représentée par une succession de points peut recevoir plusieurs solutions y compris du même type. Dans le cas de fontions positives croissantes, on peut chercher à opérer d'abord une transformation

$Y \rightarrow (\text{Log } Y)^{\alpha}$ si elle présente une convexité, et du type $Y \rightarrow (\text{Exp } Y)^{\alpha}$ dans le cas contraire. Une même courbe peut alors recevoir plusieurs équations, voire de chaque type, aussi le mieux est-il de laisser le choix, ce que fait la fonction principale.

```
(de clis (X Y)
        (print (cons 'ln (E5 (E2  X (mapcar 'log Y)1 2))))
        (print (cons 'exp (E5 (E2 X (mapcar 'exp Y) .1 .2))))
        (print (cons 'expbase1.01 (E5 (E2 X (mapcar 'ptex Y) 1 2))))
        (print (E5 (E2 X Y .5 1)))
        (if (< (car Y) (car (last Y))) (print (E5 (E2  X Y .8 1.2)))
                (print (E5 (E2 X Y -0.8 -1.2))))) 'choisissez)
```

Exemple, pour X la liste (1 2 3 4 5 6 7 8 9) et Y la liste des ordonnées (1 2 2,8 3,3 3,7 3,9 4 4,1 4,15), on cherche à rectifier la courbe par des puissances de plus en plus grandes, mais pour Y^6, on s'aperçoit que la courbe devient convexe, l'algorithme précédent donnera un exposant 5,25 optimum (pour le coefficient de corrélation).



14-8° Trouver une représentation d'un réseau multicouches et une fonction "init" qui pour une liste de nombres par exemple (64 21 7 2) va renvoyer un réseau qui aura 4 couches de respectivement 64, 21, 7 et 2 neurones, avec tous les poids initialement fixés à 1.

Cette solution constitue un exercice de style, afin de montrer une programmation purement fonctionnelle d'un réseau à couches. Quite à augmenter la taille de la pile, tout marche bien sur de petits exemples. On s'orientera cependant plutôt vers des réseaux à une seule couche cachée.
La représentation naturelle d'un réseau à couches, est une liste des couches, chaque couche étant une liste représentant un neurone et des poids. Ici, on peut choisir de lui associer, soit la liste des poids qui le relient à la couche précédente, soit celle des poids le reliant à la couche suivante. Pour des raisons qui n'apparaissent véritablement que lors de la rétropropagation, nous avons choisi la seconde option. Par ailleurs, le neurone est représenté par la valeur numérique de son entrée, et non par son "état" ou valeur de sortie, mais là, on peut dire que les deux choix sont équivalents. Ainsi par exemple pour 4 couches comportant respectivement 64, 21, 7 et 2 neurones :

$$R = \left(\begin{pmatrix} (e_{1,1}\ w_{1,1,1}\ \dots\ w_{1,1,21}) \\ (e_{1,2}\ w_{1,2,1}\ \dots\ w_{1,2,21}) \\ \dots \\ (e_{1,64}\ w_{1,64,1}\ \dots\ w_{1,64,21}) \end{pmatrix} \begin{pmatrix} (e_{2,1}\ w_{2,1,1}\ \dots\ w_{2,1,7}) \\ \dots \\ (e_{2,21}\ w_{2,21,1}\ \dots\ w_{2,21,7}) \end{pmatrix} \begin{pmatrix} (e_{3,1}\ w_{3,1,1}\ w_{3,1,2}) \\ \dots \\ (e_{3,7}\ w_{3,7,1}\ w_{3,7,2}) \end{pmatrix} \begin{pmatrix} (e_{4,1}) \\ (e_{4,2}) \end{pmatrix} \right)$$

Mais naturellement, on ne fera pas ces numérotations.

```
(de init1 (n) (if (eq n 0) nil (cons 1 (init1 (- n 1)))))  ;produit une liste de n entiers égaux à 1

Exemple :              (init1 7) = (1 1 1 1 1 1 1)

(de init2 (n L) (if (eq n 0) nil (cons (cons 0 L) (init2 (- n 1) L))))
                       ; donne la couche obtenue avec ((0 L) (0 L) ... (0 L)) n fois

Exemple :              (init2 3 '(a b c)) = ((0 a b c) (0 a b c) (0 a b c))

(de init3 (n1 n2 RC) (cons (init2 n1 (init1 n2)) RC))
            ; renvoie le réseau à l'envers où on a rajouté une couche, RC étant le réseau déjà construit
(de init4 (LN RC) (cond
            ((null LN) (reverse RC))
            ((null (cdr LN)) (init4 nil (init3 (car LN) 0 RC)))
            (t (init4 (cdr LN) (init3 (car LN) (cadr LN) RC))) ))
            ; donne le réseau où LN est la liste des nombres de neurones par couches
(de init (LN) (init4 LN nil)) ; est la fonction principale.

Exemple :
(init  '(5 3 1)) = (((0 1 1 1)  (0 1 1 1)  (0 1 1 1)  (0 1 1 1)  (0 1 1 1))   ((0 1)  (0 1)  (0 1))   ((0)) )

(init '(12 6 3 2 1))
= (((0 1 1 1 1 1 1) (0 1 1 1 1 1 1) (0 1 1 1 1 1 1) (0 1 1 1 1 1 1)
   (0 1 1 1 1 1 1) (0 1 1 1 1 1 1) (0 1 1 1 1 1 1) (0 1 1 1 1 1 1)
   (0 1 1 1 1 1 1) (0 1 1 1 1 1 1) (0 1 1 1 1 1 1) (0 1 1 1 1 1 1))
   ((0 1 1 1) (0 1 1 1) (0 1 1 1) (0 1 1 1) (0 1 1 1) (0 1 1 1))
   ((0 1 1) (0 1 1) (0 1 1)) ((0 1) (0 1))
 ((0)))
```

14-9° Construire une fonction "prg" qui à un exemple X et un réseau R, fait correspondre le réseau R où toutes les entrées sont calculées et considérées comme états des neurones.

Calcul des entrées d'une couche C2 à partir de la couche précédente C1, une première version placait les entrées comme états des neurones (dans les exemples), mais ici on range les sorties comme "états", ce qui diminue le nombre d'appels de f.

```
(de prg1 (E C1V C1R C2V C2R) (cond
        ((null C2R) (reverse C2V))
        ((null C1R) (prg1 0 nil (reverse C1V)  (cons (cons (f E) (cdar C2R)) C2V) (cdr C2R)))
        (t (prg1 (+ E (* (caar C1R) (cadar C1R))) (cons (cons (caar C1R) (cddar C1R)) C1V)
                                                  (cdr C1R) C2V C2R)) ))
```

E est la somme déjà faite, C1V partie de la couche d'avant vue et allégée des poids déjà utilisés, C1R ce qu'il en reste à parcourir, C2V les étages de la couche suivante déjà calculés C2R ceux qui restent, prg1, pour chaque neurone de C2, balaye C1, il est lancée par :

```
(de prg2 (C1 C2) (prg1 0 nil C1 nil C2))
(de prg3 (RV C RR) (if (null RR) ; RV couches déjà vue du réseau, C couche vue, RR couches restantes
            (cons C RV)    (prg3 (cons C RV) (prg2 C (car RR)) (cdr RR))))

(de mcar (L1 L2) (mcar1 L1 L2 nil))
        ; remplace les e'i de L2 = ((e'1 w1 ...wn)...(e'k u1...up)) par ei  si L1 = (e1 e2 ... ek)
(de mcar1 (L1 L2 R) (if (null L2) (reverse R)
        (mcar1 (cdr L1) (cdr L2) (cons (cons (car L1) (cdar L2)) R))))

Exemple : (mcar '(1 2 3) '((a b)(b c d)(c e)))  = ((1 b) (2 c d) (3 e))
(de prg (EX R) (prg3 nil (mcar (mapcar 'f EX) (car R)) (cdr R)))  ; est la fonction principale
```

Si EX est un exemple, soit une liste d'entrées, prg donne un nouvel état du réseau, mais renversé.

```
(de f(x) (- 1 (/ 2 (+ 1 (exp (* 2 x))))))              ; c'est la fonction tangente hyperbolique
(de derf (x) (- 1 (sqr (f x))))
(de sqr (x) (* x x))
```

Derf est la dérivée de f, cette fonction devient inutile par la suite, car il faudra avoir f'(x) en fonction de la sortie s = f(x). Exemple :

```
(prg '(4 1 0 2 3) (init '(5 3 1)))
= ((( 2.996479E0))
   (( 3.720006E0 1) ( 3.720006E0 1) ( 3.720006E0 1))
   ((4 1 1 1) (1 1 1 1) (0 1 1 1) (2 1 1 1) (3 1 1 1)))
```

14-10° Construire une fonction "rpg", qui à un coefficient μ, une sortie attendue Y et un réseau R, fait correspondre le réseau R où les poids sont modifiés suivant l'algorithme de rétropropagation.

On note D1 la liste des di de la couche aval C1, et C2V la partie transformée de la couche amont, (e LW) le neurone de sortie s dont on modifie la liste LW de ses poids vers l'aval, C2R est la partie restante de la couche amont.

```
(de rpg1 (LE LY LD) (if (null LE)
            ; rgp1 ne sert que pour la dernière couche LE avec la liste LY des sorties attendues
            (reverse LD) ; fini, d'ailleurs LE, LY en général très courtes souvent un seul neurone en sortie
            (rpg1 (cdr LE) (cdr LY) (cons (* 2 (- (caar LE) (car LY)) (dd (caar LE))) LD))))
            ; calcule la liste LD des premiers di = 2(f(ei)-yi)f'(ei)
```

```
(de rpg2 (mu D1 LD C2V D2V s d WV WR C2R) (if (null WR)
                (rpg3 mu D1 (cons (cons s (reverse WV)) C2V)  (cons (* d (dd s)) D2V) C2R ); fin du calcul de e
                (rpg4 mu D1 (cdr LD) C2V D2V s d (car LD)  WV (car WR) (cdr WR) C2R)))
```

A l'étape s, sortie du neurone, WV liste des poids vus w_i étant le poids courant. WR est la liste des poids restants, d_i le "d" courant pour la couche suivante, d la somme partielle du "d" de ce neurone.
Dans le même temps on modifie les poids $w_i := w_i - \mu * d_i * f(e) = w_i - \mu * d_i * s$ et on calcule $d = (\sum d_i * w_i) f'(e_i) = s \sum d_i * w_i$ (un seul balayage de W).

```
(de rpg3 (mu D1 C2V D2V C2R) (if (null C2R)
                (list (reverse C2V) (reverse D2V))          ; fin de la couche C2
                (rpg2 mu D1 D1 C2V D2V (caar C2R) 0 nil (cdar C2R) (cdr C2R)) ))
                ; début d'un nouveau neurone de C2R dans le second cas
                ; rpg3 renvoie une liste double formée de la couche C2 et de celle de ses "d"
(de rpg4 (mu D1 LDR C2V D2V s d di WV wi WR C2R)
                (rpg5 mu D1 LDR C2V D2V s d di WV (- wi (* mu di s)) WR C2R))
(de rpg5 (mu D1 LDR C2V D2V s d di WV wn WR C2R)
                (rpg2 mu D1 LDR C2V D2V s (+ d (* di wn)) (cons wn WV) WR C2R))
(de rpg6 (mu LD RV RR)   (if (null RR) RV    ; c'est fini, mais seuls les poids seront intéressants
                (rpg7 mu RV  (rpg2 mu LD LD nil nil (caaar RR) 0 nil (cdaar RR) (cdar RR) )  (cdr RR))))
```

Les arguments de rpg7 sont pour le réseau à l'envers, RV couches déjà vues, CD liste de la couche courante et des "d" associés, RR couches restantes

```
(de rpg7 (mu RV CD RR)      (rpg6 mu (cadr CD) (cons (car CD) RV) RR))
(de rpg (mu Y R)       (rpg6 mu (rpg1 (car R) Y nil) (list (car R)) (cdr R))) ; (car R) est la 1° couche à voir
```

Rpg est la fonction principale où Y est la sortie attendue et R le réseau renversé.
On va maintenant pouvoir enchainer des essais :

```
(set 'r0 '( ( (0 0.5 0.6 0.9) (0 0.8 0.7 0.5) (0 0.1 0.5 0.9) (0 0.8 0.5 0.6) )  ( (0 0.9) (0 0.8) (0 0.7) )  ( (0) ) ) )
(set 'ex '(1 0 2 3))
(prg ex r0) ; on l'appelera r1
= (  (( 2.793619E0))
     (( 1.643167E0  9.000000E-1) ( 2.203792E0 8.000000E-1) ( 2.609910E0  7.000000E-1))
     ((1  5.000000E-1  6.000000E-1  9.000000E-1) (0  8.000000E-1  7.000000E-1  5.000000E-1)
      (2 1.000000E-1  5.000000E-1  9.000000E-1) (3  8.000000E-1  5.000000E-1  6.000000E-1))))
```

Exemple de rétropropagation, on poursuit avec r1 par (set 'r2 (rpg 0.1 '(1) r1)) qui donne le résultat r2 du tableau :

Après rétropropagation	Après propagation (le résultat est inversé)
r0 = (((0 5.000000E-1 6.000000E-1 9.000000E-1) (0 8.000000E-1 7.000000E-1 5.000000E-1) (0 1.000000E-1 5.000000E-1 9.000000E-1) (0 8.000000E-1 5.000000E-1 6.000000E-1)) ((0 9.000000E-1) (0 8.000000E-1) (0 7.000000E-1)) ((0)))	r1 = (((2.793619E0)) ((1.643167E0 9.000000E-1) (2.203792E0 8.000000E-1) (2.609910E0 7.000000E-1)) ((1 5.000000E-1 6.000000E-1 9.000000E-1) (0 8.000000E-1 7.000000E-1 5.000000E-1) (2 1.000000E-1 5.000000E-1 9.000000E-1) (3 8.000000E-1 5.000000E-1 6.000000E-1)))
r2 = (((1 5.000021E-1 6.000007E-1 9.000002E-1) (0 8.000000E-1 7.000000E-1 5.000000E-1) (2 1.000027E-1 5.000008E-1 9.000003E-1) (3 8.000028E-1 5.000008E-1 6.000004E-1)) ((1.643167E0 9.000206E-1) (2.203792E0 8.000216E-1) (2.609910E0 7.000219E-1)) ((2.793619E0)))	r3 = (((2.793637E0)) ((1.643174E0 9.000206E-1) (2.203794E0 8.000216E-1) (2.609910E0 7.000219E-1)) ((1 5.000021E-1 6.000007E-1 9.000002E-1) (0 8.000000E-1 7.000000E-1 5.000000E-1) (2 1.000027E-1 5.000008E-1 9.000003E-1) (3 8.000028E-1 5.000008E-1 6.000004E-1)))

```
r4
= ((   (1  5.000042E-1  6.000013E-1  9.000005E-1)
       (0  8.000000E-1  7.000000E-1  5.000000E-1)
       (2  1.000054E-1  5.000016E-1  9.000006E-1)
       (3  8.000056E-1  5.000016E-1  6.000007E-1))
    (  ( 1.643174E0  9.000412E-1)
       ( 2.203794E0  8.000432E-1)
       ( 2.609910E0  7.000439E-1))
   (( 2.793637E0)))
```

```
r5
= (   ( ( 2.793656E0))
      (( 1.643182E0  9.000412E-1)
       ( 2.203795E0  8.000432E-1)
       ( 2.609910E0  7.000439E-1))
     ((1  5.000042E-1  6.000013E-1  9.000005E-1)
      (0  8.000000E-1  7.000000E-1  5.000000E-1)
      (2  1.000054E-1  5.000016E-1  9.000006E-1)
      (3  8.000056E-1  5.000016E-1  6.000007E-1)))
```

```
r6
= ((   (1  5.000063E-1  6.000020E-1  9.000007E-1)
       (0  8.000000E-1  7.000000E-1  5.000000E-1)
       (2  1.000080E-1  5.000024E-1  9.000009E-1)
       (3  8.000083E-1  5.000024E-1  6.000011E-1))
    (  ( 1.643182E0  9.000617E-1)
       ( 2.203795E0  8.000649E-1)
       ( 2.609910E0  7.000658E-1))
   (( 2.793656E0)))
```

```
r7
= (  (( 2.793675E0))
     (( 1.643188E0  9.000617E-1)
      ( 2.203797E0  8.000649E-1)
      ( 2.609911E0  7.000658E-1))
    ((1  5.000063E-1  6.000020E-1 9.000007E-1)
     (0  8.000000E-1  7.000000E-1  5.000000E-1)
     (2  1.000080E-1  5.000024E-1  9.000009E-1)
     (3  8.000083E-1  5.000024E-1  6.000011E-1)))
```

```
r8
= (   ((1  5.000083E-1  6.000026E-1  9.000009E-1)
       (0  8.000000E-1  7.000000E-1  5.000000E-1)
       (2  1.000107E-1  5.000033E-1  9.000012E-1)
       (3  8.000110E-1  5.000033E-1  6.000015E-1))
    (( 1.643188E0  9.000823E-1)
       ( 2.203797E0  8.000865E-1)
       ( 2.609911E0  7.000877E-1))
   (( 2.793675E0)))
```

```
r9
= (   (( 2.793693E0))
      ( ( 1.643196E0  9.000823E-1)
        ( 2.203798E0  8.000865E-1)
        ( 2.609911E0  7.000877E-1))
      ((1  5.000083E-1  6.000026E-1 9.000009E-1)
       (0  8.000000E-1  7.000000E-1  5.000000E-1)
       (2  1.000107E-1  5.000033E-1  9.000012E-1)
       (3  8.000110E-1  5.000033E-1  6.000015E-1)))
```

Comme on le voit, les poids, se modifient très lentement, ce qui nous engage à augmenter le gradient $\mu = 0{,}1$. "Sortie" donne la liste des valeurs de la couche la plus en aval, exemple :

```
(de sortie (R) (ifn (null (car R)) (cons (caaar R) (sortie (list (cdar R)))) ))
(mapcar 'sortie (list r1 r3 r5 r7 r9))
= (( 9.925373E-1) ( 9.925375E-1) ( 9.925378E-1) ( 9.925381E-1) (9.925383E-1))
```

14-11° Construire une fonction "app" qui pour μ, s, une liste d'exemples LX, LY et un réseau R, fait correspondre le réseau R répondant sur tous les exemples de LX des sorties voisines de celles de LY à s près.

```
(de erreur (R Y); erreur quadratique du réseau inversé R par rapport à la liste Y de sorties désirées
        (err1 (car R) Y 0))
(de err1 (LE Y er) (if (null Y) er  (err1 (cdr LE) (cdr Y) (+ er (sqr (- (car Y) (caar LE))))) ))
(de app1 (mu s R LX LY X Y toutbon n) ;n = nombre d'aller-retours
        (if (null X) (if  toutbon  R ; on a vu les exemples autant de fois qu'il fallait, R à l'endroit
                (print n "' aller-retours ") (vue R)
                (print "'On refait un tour de tous les exemples. ")
                (app1 mu s R LX LY LX LY t n))                  ; retour au début des exemples
        (app2 mu s (prg (car X) R) LX LY X Y toutbon n)))       ; sinon on entre le premier exemple
        ; toutbon n'est vrai que si un tour complet des exemples a été fait sans rétropropagation
(de app2 (mu s R LX LY X Y toutbon n)                           ; ici R à l'envers
        (vue (reverse R)) (print "'entrée " (car X) "' sortie " (mlis (sortie R)) "' cible " (mlis (car Y)) )
        (if (< s (erreur  R (car Y)))
                (app1 mu s (rpg mu (car Y) R) LX LY X Y nil (+ 1 n)) ; résultat insuffisant sur le premier ex.
                (app1 mu s (reverse R) LX LY (cdr X) (cdr Y) toutbon n)))
                        ; sinon, on passe à l'exemple suivant

(de app (mu s R  LX LY) (app1 mu s R LX LY  LX LY t 0))         ; fonction principale
(setq LX '((0 0 0 1) (0 0 1 0) (1 0 0 1) (1 0 1 0) (0 1 1 1) (1 1 0 1))     ; cet exemple compte en binaire les "1"
        LY '((0 1) (0 1) (1 0) (1 0) (1 1) (1 1))) R (init '(4 2)))
```

Pour essayer, on lance : (app 0.5 0.3 r lx ly) avec un seuil de 0.3 puis de 0.1 dans le tableau suivant.

```
(app 0.5 0.3 R LX LY)
entrée (0 0 0 1) sortie (6.420150E-1  6.420150E-1) cible (0 1)
entrée (0 0 0 1) sortie (5.457404E-1  6.756818E-1) cible (0 1)
entrée (0 0 0 1) sortie (3.778541E-1 6.964117E-1) cible (0 1)
entrée (0 0 1 0) sortie ( 6.420150E-1 6.420150E-1) cible (0 1)
entrée (0 0 1 0) sortie ( 5.457404E-1 6.756818E-1) cible (0 1)
entrée (0 0 1 0) sortie ( 3.778541E-1 6.964117E-1) cible (0 1)
entrée (1 0 0 1) sortie ( 8.207621E-1 9.248984E-1) cible (1 0)
entrée (1 0 0 1) sortie ( 8.378942E-1 9.132575E-1) cible (1 0)
entrée (1 0 0 1) sortie ( 8.503560E-1 8.949881E-1) cible (1 0)
entrée (1 0 0 1) sortie ( 8.599705E-1 8.631753E-1) cible (1 0)
entrée (1 0 0 1) sortie ( 8.676904E-1 7.987345E-1) cible (1 0)
entrée (1 0 0 1) sortie ( 8.740718E-1 6.392961E-1) cible (1 0)
entrée (1 0 0 1) sortie ( 8.794646E-1 2.271653E-1) cible (1 0)
entrée (1 0 1 0) sortie ( 8.418740E-1 6.813887E-1) cible (1 0)
entrée (1 0 1 0) sortie ( 8.530599E-1 4.390771E-1) cible (1 0)
entrée (0 1 1 1) sortie ( 9.392496E-1 9.457420E-1) cible (1 1)
entrée (1 1 0 1) sortie ( 9.729922E-1 6.194073E-1) cible (1 1)
On refait un tour de tous les exemples.
entrée (0 0 0 1) sortie ( 4.955423E-1 2.541046E-1) cible (0 1)
entrée (0 0 0 1) sortie (3.034464E-1  5.800333E-1) cible (0 1)
entrée (0 0 1 0) sortie (4.025322E-1  6.460180E-1) cible (0 1)
entrée (1 0 0 1) sortie (8.190513E-1  3.497561E-1) cible (1 0)
entrée (1 0 1 0) sortie (8.530599E-1  4.390771E-1) cible (1 0)
entrée (0 1 1 1) sortie (9.054320E-1  9.753830E-1) cible (1 1)
entrée (1 1 0 1) sortie (9.575468E-1  8.099074E-1) cible (1 1)
On refait un tour de tous les exemples.
entrée (0 0 0 1) sortie (3.034464E-1 5.800333E-1) cible (0 1)
entrée (0 0 1 0) sortie (4.025322E-1  6.460180E-1) cible (0 1)
entrée (1 0 0 1) sortie (8.190513E-1 3.497561E-1) cible (1 0)
entrée (1 0 1 0) sortie (8.530599E-1  4.390771E-1) cible (1 0)
entrée (0 1 1 1) sortie ( 9.054320E-1 9.753830E-1) cible (1 1)
entrée (1 1 0 1) sortie (9.575468E-1  8.099074E-1) cible (1 1)
= (((1  1.223273E0 -3.066070E-1)
      (1  1.000000E0  1.000000E0)
      (0  4.558148E-1  1.016493E0)
      (1  3.242128E-1  7.972792E-1))
  (( 1.915522E0)  ( 1.126760E0)))
```

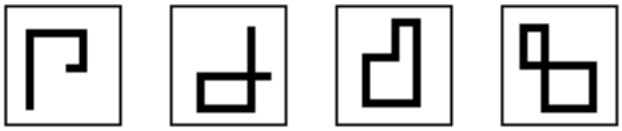
```
On essaie maintenant un seuil plus fin  (app 0.5 0.1 R LX LY)
entrée (0 0 0 1) sortie (6.420150E-1 6.420150E-1) cible (0 1)
entrée (0 0 0 1) sortie ( 5.457404E-1 6.756818E-1) cible (0 1)
entrée (0 0 0 1) sortie (3.778541E-1 6.964117E-1) cible (0 1)
entrée (0 0 0 1) sortie (1.706330E-1 7.105283E-1) cible (0 1)
entrée (0 0 0 1) sortie (4.781270E-2  7.207348E-1) cible (0 1)
entrée (0 0 1 0) sortie ( 6.420150E-1 6.420150E-1) cible (0 1)
entrée (0 0 1 0) sortie ( 5.457404E-1 6.756818E-1) cible (0 1)
entrée (0 0 1 0) sortie (3.778541E-1  6.964117E-1) cible (0 1)
entrée (0 0 1 0) sortie (1.706330E-1  7.105283E-1) cible (0 1)
entrée (0 0 1 0) sortie (4.781270E-2  7.207348E-1) cible (0 1)
entrée (1 0 0 1) sortie (6.692830E-1  9.316530E-1) cible (1 0)
entrée (1 0 0 1) sortie (7.611235E-1  9.232835E-1) cible (1 0)
entrée (1 0 0 1) sortie (7.990158E-1  9.110336E-1) cible (1 0)
entrée (1 0 0 1) sortie (8.218478E-1  8.917460E-1) cible (1 0)
entrée (1 0 0 1) sortie  (8.376518E-1 8.580659E-1) cible (1 0)
entrée (1 0 0 1) sortie (8.494617E-1  7.898667E-1) cible (1 0)
entrée (1 0 0 1) sortie (8.587334E-1 6.234581E-1) cible (1 0)
entrée (1 0 0 1) sortie  (8.662692E-1 2.163629E-1) cible (1 0)
entrée (1 0 1 0) sortie (7.332738E-1 6.450762E-1) cible (1 0)
entrée (1 0 1 0) sortie (7.806647E-1 4.126553E-1) cible (1 0)
entrée (1 0 1 0) sortie (8.080357E-1 1.511945E-1) cible (1 0)
entrée (0 1 1 1) sortie (8.844264E-1  9.540211E-1) cible (1 1)
entrée (1 1 0 1) sortie (9.709610E-1  4.817043E-1) cible (1 1)
entrée (1 1 0 1) sortie (9.710617E-1 7.955009E-1) cible (1 1)
On refait un tour de tous les exemples.
entrée (0 0 0 1) sortie (4.061916E-1 5.321716E-1) cible (0 1)
entrée (0 0 0 1) sortie (1.974957E-1 6.256714E-1) cible (0 1)
entrée (0 0 0 1) sortie (5.818635E-2 6.665801E-1) cible (0 1)
entrée (0 0 0 1) sortie (1.415205E-2 6.905408E-1) cible (0 1)
entrée (0 0 1 0) sortie (2.013419E-1 6.357130E-1) cible (0 1)
entrée (0 0 1 0) sortie (5.977178E-2 6.720990E-1) cible  (0 1)
entrée (0 0 1 0) sortie (1.455343E-2 6.940769E-1) cible  (0 1)
entrée (1 0 0 1) sortie (7.313234E-1 4.458654E-1) cible  (1 0)
entrée (1 0 0 1) sortie (7.779437E-1 1.732270E-1) cible (1 0)
entrée (1 0 1 0) sortie (7.378719E-1 2.705970E-1) cible (1 0)
entrée (1 0 1 0) sortie (7.808137E-1  1.036567E-1) cible (1 0)
entrée (0 1 1 1) sortie (7.513890E-1  9.840380E-1) cible (1 1)
entrée (1 1 0 1) sortie (9.481734E-1  7.279136E-1) cible (1 1)
On refait un tour de tous les exemples.
entrée (0 0 0 1) sortie (1.084751E-1  6.370918E-1) cible (0 1)
entrée (0 0 0 1) sortie (2.768123E-2  6.728750E-1) cible (0 1)
entrée (0 0 0 1) sortie (6.629586E-3  6.945800E-1) cible (0 1)
entrée (0 0 1 0) sortie (1.047233E-1  6.615402E-1) cible (0 1)
entrée (0 0 1 0) sortie (2.660376E-2  6.873754E-1) cible (0 1)
entrée (1 0 0 1) sortie (7.393176E-1  1.637465E-1) cible (1 0)
entrée (1 0 1 0) sortie (7.482439E-1  1.502987E-1) cible (1 0)
entrée (0 1 1 1) sortie (6.614375E-1 9.881718E-1) cible (1 1)
entrée (0 1 1 1) sortie (8.134123E-1 9.881759E-1) cible (1 1)
entrée (1 1 0 1) sortie (9.570338E-1  7.730540E-1) cible (1 1)
On refait un tour de tous les exemples.
entrée (0 0 0 1) sortie (1.493900E-1 6.946092E-1) cible (0 1)
entrée (0 0 0 1) sortie (4.038948E-2 7.092595E-1) cible (0 1)
entrée (0 0 1 0) sortie (1.683705E-1 6.874077E-1) cible (0 1)
entrée (0 0 1 0) sortie  (4.699123E-2 7.042693E-1) cible (0 1)
entrée (1 0 0 1) sortie (7.542559E-1 1.917711E-1) cible (1 0)
entrée (1 0 1 0) sortie  (7.570931E-1 1.821486E-1) cible (1 0)
entrée (0 1 1 1) sortie  (7.182224E-1 9.895389E-1) cible (1 1)
entrée (1 1 0 1) sortie  (9.467315E-1 7.844200E-1) cible (1 1)
On refait un tour de tous les exemples.
entrée (0 0 0 1) sortie  (4.038948E-2 7.092595E-1) cible (0 1)
entrée (0 0 1 0) sortie (4.699123E-2 7.042693E-1) cible  (0 1)
entrée (1 0 0 1) sortie (7.542559E-1 1.917711E-1) cible  (1 0)
entrée (1 0 1 0) sortie (7.570931E-1 1.821486E-1) cible  (1 0)
entrée (0 1 1 1) sortie  (7.182224E-1 9.895389E-1) cible (1 1)
entrée (1 1 0 1) sortie (9.467315E-1 7.844200E-1) cible (1 1)
= (( (1  1.758570E0 -8.508574E-1)
      (1  1.146305E0  1.303350E0)
      (0  4.706065E-2  1.357111E0)
      (1  4.043341E-2  1.401578E0))
    (( 1.799281E0)  ( 1.056758E0)))
```

A présent, on essaie le réseau obtenu sur d'autres exemples, une "base de généralisation" :
Comme on le voit, les résultats sont tout à fait médiocres, puisque l'apprentissage a été bref.

```
(sortie  (prg '( 1 0 0 0) R)) = ( 7.362969E-1 -5.989558E-1)
(sortie  (prg '( 0 1 0 0) R)) = ( 6.731747E-1  6.975862E-1)
(sortie  (prg '( 0 1 0 1) R)) = ( 6.946766E-1  9.411789E-1)
(sortie  (prg '( 1 1 0 0) R)) = ( 9.423766E-1  1.694165E-1)
(sortie  (prg '( 1 1 1 0) R)) = ( 9.474131E-1  7.805542E-1)
(sortie  (prg '( 1 1 1 1) R)) = ( 9.513969E-1  9.589334E-1)
```

14-12° Revoir la fonction "app" pour un apprentissage par lots : tous les exemples sont proposés, l'erreur moyenne sur ces exemples n'étant rétropropagée qu'après. Cette stratégie permet d'éviter une trop grande focalisation du réseau sur l'apprentissage d'un exemple, qu'il pourrait oublier durant l'apprentissage de l'exemple suivant.

14-13° Expérimenter sur des exemples de reconnaissance tel que celui donné par les dessins ci contre :



00 01 10 11

on voudrait qu'en traçant sur une grille carrée des dessins formés par une seule ligne, on ait en sortie deux neurones dont le premier réponde 1 si la courbe est fermée et 0 sinon, le second donnant 1 s'il y a un point double et 0 sinon.

On commence par modifier la fonction f de façon à ramener R à l'intervalle [0, 1]

```
(de f(x) (/ 1 (+ 1 (exp (- x)))))
(de dd (s) (* s (- 1 s)))      ; dérivée de f par rapport à s = f(x)
```

Puis on soigne un peu plus la sortie, en ne retenant que des millièmes :

```
(de mil (x) (truncate (* 1000 x)))
(de mlis (L) (mapcar 'mil (cdr L)))          ; afin d'avoir une vue des poids à trois décimales seulement
(de vue (r) (if (null r) () (print (mapcar 'mlis (car r))) (vue (cdr r))))
```

En tenant compte des observations empiriques des spécialistes, on revoie "init" pour avoir des poids initiaux aléatoirement distribués de façon uniforme autour de 0,5 avec une marge de 0,3

```
(de init1 (n) (if (eq n 0) nil (cons (div (random 20 80) 100) (init1 (- n 1)))))
        ; produit une liste de n réels aléatoires entre 0,2 et 0,8
(de init2 (n1 n2) (if (eq n1 0) nil (cons (cons 0 (init1 n2)) (init2 (- n1 1) n2)))))
        ; produit une couche de n1 neurones ayant chacun n2 poids
(de init3 (n1 n2 RC) (cons (init2 n1 n2) RC))
        ; renvoie le réseau à l'envers où on a rajouté une couche, RC étant le réseau déjà construit
        ; Init4 et init ne sont pas changées.
```

On utilise une couche cachée ayant davantage de cellules que la couche d'entrée.
Enfin, considérons que si une entrée est assez grande, (supérieure à quelques unités) la sortie sera proche de 1, et donc sa dérivée proche de 0, ce qui donnera des poids presque stationnaires. On a donc intérêt à choisir un coefficient μ beaucoup plus grand. Cependant un meilleur apprentissage sera acquis avec μ voisin de 0,5 qui le fait plus lent.

```
(setq LX  '((0 1 1 1 0 1 1 1 1)  (0 1 0 1 0 1 0 1 0)  (0 1 0 0 1 1 0 0 0)  (1 1 0 0 1 0 0 1 0)  (1 0 1 0 1 0 1 0 1))
      LY  '((1 1) (1 0) (0 1) (0 1) (0 0)))      R (init '(9 12 2))) ; encadrement d'une case blanche et angle droit
```

14-14° **Test de parité :** 8 entrées à 0 ou à 1, 1 sortie donnant 1 si et seulement si il y a un nombre impair de 1, μ = 0,5, et une seule couche cachée. prendre un réseau (6 6 1) et stopper dès que la reconnaissance pour pair donne une sortie > 0.75 et < 0.25 pour impair.

14-15° **Reconnaissance d'une séquence à l'aide d'un réseau (5 8 8 2)**
Les exemples sont (0 2 4 6 8), (2 4 6 8 0), (6 8 0 4 2) $\rightarrow$ (0.6 0.8) et (6 4 0 2 8), (6 2 4 0 8), (2 8 6 4 0), (4 2 8 0 6) $\rightarrow$ (0 0)

14-16° Test de XOR avec un réseau (2 5 1). On veut réaliser la fonction f(0, 1) = f(1, 0) = 1 et f(0, 0) = f(1, 1) = 0

Une expérimentation qui semble périodique :
(set 'r (init'(2 5 1)))
(set 'r (app 0.5 0.2 R LX LY)) ; pour n = 350 en fait, on obtient le réseau :
r = (((7.310586E-1 7.052695E-1 8.474059E-1 1.134796E0 9.369898E-1 6.095495E-1 1.349365E-3 5.850312E-1 9.474262E-1 1.139973E0 1.015554E0) (7.310586E-1 6.257773E-1 7.140535E-1 5.590437E-1 8.839923E-1 1.083451E0 1.382182E0 7.162994E-1 4.672311E-1 4.940568E-1 7.893940E-1))
((7.257317E-1 5.277596E-2) (7.579583E-1 2.489196E-1) (7.752672E-1 -4.597723E-1) (7.910465E-1 -3.507776E-1) (7.751603E-1 -3.533371E-2) (7.333024E-1 1.136565E0) (7.213864E-1 1.425045E-1) (7.377288E-1 -1.849592E-1) (7.675576E-1 -4.357184E-1) (7.891023E-1 -4.103289E-1))
((4.273237E-1)))

14-17° **Reconnaissance de majuscules sans boucles,** on code 1 une barre verticale; 0,8 une barre ascendante; 0,5 une horizontale et 0,2 une barre decendante. Par exemple on donne les leçons :

Lettre	Codage entrée (2 ou 3 ou 4)			Sortie attendue
L	1	0,5		0,1
T	0,5	1	0,5	0,15
H	1	0,5	1	0,2
F	1	0,5	0,5	0,25
V	0,2	0,8		0,6
X	0,2	0,8	0,2	0,65
A	0,8	0,5	0,2	0,7

Avec un réseau (3 7 1) que donnera un T mal fichu codé 0,6 1, un E (1 0,6 0,5 0,4), un H (0,5 1 0,5 1) etc ?

14-18° **Le Diabolo** (Le jeu du diabolo fut inventé juste avant la guerre de 14-18) [Davalo Naim 85] Un réseau (5 2 5) doit faire l'identité sur les exemples 10000, 01000, 00001, 00100, 00010 au seuil de 0.05.

14-19° **Réseau de Kohonen.** Le but est un apprentissage non supervisé où les entrées possibles sont classées. Il n'y a que deux couches, l'entrée $(x_1, \ldots, x_p)$ comportant autant de neurones que la dimension n des exemples, et la sortie comportant p neurones. On définit le voisinage V_j de la sortie j, par exemple si p = 16, chaque neurone peut avoir son voisinage décrit par ses 4 voisins, puis 3, puis 2. Chaque poids w_{ij} est modifié par la règle suivante :
A chaque sortie j (avec $1 \le j \le p$), on calcule la distance $\sum_{1 \le i \le n} (x_i - w_{ij})^2$ et si j_0 est la sortie pour laquelle cette distance est minimale, on met les poids à jour pour tous les i, mais seulement pour les j dans le voisinage de j_0 par la règle $w_{ij} \leftarrow w_{ij} + \eta(x_i - w_{ij})$

Puis on fait décroître la taille du voisinage, et enfin η qui mesure le taux d'apprentissage, doit décroître aussi de 0,8 initialement avec un décrement de 0.005. La convergence doit se faire en une dizaine de cycles vers un réseau reconnaissant p classes.

14-20° **Réseau "Radial Basis Function Network",** de [Poggio 90] et [Platt 91]. Ce sont des réseaux à trois couches dont la première possède n neurones correspondant à la dimension des vecteurs d'entrées.
La couche cachée possède un nombre de neurones qui peut être n ou un peu moins, (ce nombre peut même être variable si on crée un neurone supplémentaire chaque fois que l'erreur dépasse un certain seuil) mais sa particularité est que les fonctions de transfert qui leur sont associées sont des fonctions gaussiennes avec un centre et un écart-type propre à chacun de ces neurones et qui sont modifiés en fonction des performances. Cette moyenne et cette variance constituent deux poids arrivant à chaque neurone, et qui vont être modifiés à chaque rétropropagation. Le but de l'apprentissage est que chaque neurone de la couche cachée devienne un spécialiste reconnaissant un exemple, il le laissera passer et non les autres. Programmer un tel réseau et l'essayer sur la reconnaissance des couples de {0, 1}.

14-21° Réseau de Hopfield [Hopfield 82], [Kurbel 94]. Soit un ensemble de n "neurones" d'état $x_i = \pm 1$, tous connectés entre eux par des "poids" $w_{i,j} = w_{j,i}$ avec $w_{i,i} = 0$. A chaque instant, l'entrée du neurone xi, est la somme pondérée des états de tous les autres neurones qui lui sont connectés, c'est à dire $\sum_{1\le j\le n} w_{i,j}\, x_j$.
La règle d'évolution du réseau est donnée par la modification simultanée de tous les neurones suivant laquelle x_i est remplacé par sgn ($\sum_{1\le j\le n} w_{i,j}\, x_j$).
Hopfield démontre que l'énergie du réseau défini par la fonction $H = (-1/2)\sum_{1\le i,j\le n} x_i x_j$ est décroissante et que pour un état initial, le réseau converge vers un état "attracteur".
Soit maintenant un ensemble S formés par s vecteurs de $\{-1, 1\}^n$ appelés prototypes, le problème est de constituer un réseau susceptible de reconnaître des vecteurs voisins (bruités) de ces prototypes. On peut alors montrer que chacun de ces prototypes possède un "bassin attracteur" dans lequel des vecteurs bruités (correspondant à un état du réseau) vont évoluer avec une convergence plus ou moins rapide grâce à la règle ci-dessus vers un des exemples de S. Appelons S la matrice n*s formée par les vecteurs exemples, son élément générique $s_{i,j}$ désigne donc la i-ième composante de l'exemple j.
Souhaiter une convergence revient à obtenir un point fixe S c'est à dire WS = S. Si S est l'état initial, on définit $W = SS^t - sI_n$ et donc $w_{i,\,i} = 0$ et $w_{i,\,j}$ l'élément i, j de SS^t c'est à dire $\sum_{1\le k\le s} s_{ik}s_{jk}$ On remarque donc que pour un rang k entre 1 et s, $\sum_{1\le j\le n} w_{ij}s_{ik} = \sum_{1\le j\le n}(\sum_{1\le p\le s} s_{ip}s_{jp})s_{jk} = \sum_{1\le j\le n}\sum_{1\le p\le s} s_{ip}(s_{jp}s_{jk}) = \sum_{1\le p\le s} s_{ip} \sum_{1\le j\le n} (s_{jp}s_{jk})$
Dans le cas où les exemples sont deux à deux orthogonaux c'est à dire si $SS^t = I$, alors cette dernière expression est $s_{i,k}$ ce qui signifie que le réseau est stable (chaque état de neurone est égal à son entrée). Une autre façon d'initialiser les poids est de prendre $W = SS^+ + Z(I_n - SS^+)$ où Z est quelconque n*n et S^+ est la pseudo-inverse de S, car alors $WS = SS^+S + ZS - ZSS^+S = S$. Empiriquement de bons résultats sont obtenus pour la proportion $s \le 0.14\ n$, au delà, le réseau a tendance à oublier.
Définition de l'inverse généralisée de Moore-Penrose : si A est une matrice n*p, son inverse généralisée A^+ est une matrice p*n vérifiant $AA^+A = A$ et $A^+AA^+ = A^+$ ainsi que $(A^+A)^t = A^+A$ et $(AA^+)^t = AA^+$, et bien sûr $A^+ = A^{-1}$ si A inversible.(A^+ est non nécessairement unique). Si r est le rang de A, A est semblable à la matrice J qui est nulle sauf les r premiers termes 1 de la diagonale, soit A = PJQ avec P et Q inversibles, alors $Q^{-1}JP^{-1}$ remplit la première condition. Méthode itérative de calcul de la pseudo-inverse de Greville : Si A est une matrice-colonne, alors $A^+ = (A^t.A)^{-1}.A^t$ où A^t est la transposée de A.
Si maintenant A est une matrice rectangulaire de p lignes et n colonnes, n itérations vont donner A^+. On pose a_k la k-ième colonne de A et A_k la matrice formée par les k premières colonnes de A. Si $a_1 = 0$, on pose $A_1^+ = 0$ sinon A_1^+ calculée comme pseudo-inverse de la colonne. A chaque etape, $A_{(k-1)}^+$ étant connue, on pose $d_k = A_{(k-1)}^+ a_k$, $c_k = a_k - A_{(k-1)}d_k$, et enfin b_k = si $c_k = 0$ alors $(1 + d_k^t.d_k)^{-1}d_k^t.A_{(k-1)}^+$ sinon c_k^+. La matrice A_k^+ est alors formee par $A_{(k-1)}^+ - d_k.b_k$ a laquelle on rajoute en dessous la ligne b_k. A la fin $A^+ = A_n^+$

Exemple si $A = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & -1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$ $A_1^+ = \frac{1}{2}\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$ $d_2 = \frac{1}{2}$ $c_2 = \frac{1}{2}\begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}$ $b_2 = \frac{1}{3}\begin{bmatrix} -1 & 2 & 1 \end{bmatrix}$

$A_2^+ = \frac{1}{3}\begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \end{bmatrix}$ $d_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ $c_3 = 0$ $b_3 = \frac{1}{2}\begin{bmatrix} 1 & -1 & 0 \end{bmatrix}$ $A_3^+ = \frac{1}{3}\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix}$ $d_4 = \frac{1}{3}\begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$ $c_4 = 0$

$b_4 = \frac{1}{5}\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$ d'où l'inverse $A^+ = A_4^+ = \frac{1}{15}\begin{bmatrix} 3 & 0 & 3 \\ -1 & 5 & 4 \\ 4 & -5 & -1 \\ 3 & 0 & 3 \end{bmatrix}$

Le but de l'expérimentation est de reconnaître un certain nombre (une dizaine) de mots de 5 lettres, en entrant des versions déformées de ces mots. On fera pour cela un codage permettant de passer d'une chaîne de 5 caractères à un code binaire et l'inverse permettant d'obtenir une réponse claire.

14-22° Programmer les transitions génétiques nécessaires à un renouvellement d'une population en les produisant aléatoirement, pour des chromosomes de taille fixe.

Un chromosome est une liste de caractères notée C ici :

```
(de hasard (L) (nth (random 0 (length L)) L)) ; renvoie un élément tiré au hasard dans la liste L
; Copigauche est une famille d'opérations où un gène au hasard est recopié sur sa gauche.
(de copigauche (n) (let ((p (random 2 n))) ; se généralise en translation à droite plus loin
        (list 'lambda '(C) (list 'append (list 'firstn (- p 2) 'C)
                (list 'cons (list 'nth (1- p) 'C) (list 'nthcdr (1- p) 'C))))))
(de copidroite (n) (let ((p (random 1 (1- n))))
        (list 'lambda '(C) (list 'append (list 'firstn (1- p) 'C)
                (list 'mcons (list 'nth (1- p) 'C) (list 'nth (1- p) 'C) (list 'nthcdr (1+ p) 'C))))))

; Transuiv est une famille d'opérations où deux gènes consécutifs sont inversés
(de transuiv (n) (let ((p (random 1 (1- n))))
        (list 'lambda '(C) (list 'append (list 'firstn (1- p) 'C)
                (list 'mcons (list 'nth p 'C) (list 'nth (1- p) 'C) (list 'nthcdr (1+ p) 'C))))))
; Transpo est une famille d'opérations où 2 gènes au hasard sont inversés
(de transpo (n) (let (( p (random 1 (quo n 2))) (q (random (1+ (quo n 2)) n)))  ; p < q aléatoires
                (list 'lambda '(C) (list 'append (list 'firstn  (1- p) 'C)
                (list 'cons (list 'nth (1- q) 'C) (list 'firstn (1- (- q p)) (list 'nthcdr p 'C)))
                (list 'cons (list 'nth (1- p) 'C) (list 'nthcdr q 'C))))))
```

Transg est une famille d'opérations où une sous-suite (BED sur l'exemple) de longueur aléatoire est translatée sur sa gauche avec recopie du dernier (ici D) ex: ACBEDCBAA → ABEDDCBAA

```
(de transd  (n) (let (( p (random 1 (1- n))) (q 1)); p indice du premier, q longueur de la sous-suite
        (set 'q (rand 1 (- n p)))(print p '- q)(list 'lambda '(C) (list 'append (list 'firstn p 'C)
                (list 'firstn q (list 'nthcdr (1- p) 'C)) (list 'nthcdr (+ p q ) 'C)))))

(de transg (n) (let  ((p (random 2 n)) (q 1)); p indice du premier, q longueur de la sous-suite
        (set 'q (random 1 (- (1+ n) p)))(print p '- q)(list 'lambda '(C) (list 'append (list 'firstn (- p 2) 'C)
                (list 'firstn q (list 'nthcdr (1- p) 'C)) (list 'nthcdr (- (+ p q) 2) 'C)))))
; Crossover est une famille d'opérateurs où un échange aléatoire de deux sous-suites est effectué
(de crossover (n) ; crossover C1 C2 retounera une liste dont les car et cdr seront les deux fils
(let ((p (random 1 (1- n))) (q 0)); p indice du premier, q longueur de la sous-suite
        (set 'q (random 1 (1+ (- n p)))) (print p '- q) (list 'lambda '(C1 C2) (list 'cons
                (list 'append (list 'firstn (1- p) 'C1) (list 'firstn q (list 'nthcdr (1- p) 'C2))
                (list 'nthcdr (1- (+ p q) ) 'C1)) (list 'append (list 'firstn (1- p) 'C2)
                (list 'firstn q (list 'nthcdr (1- p) 'C1)) (list 'nthcdr (1- (+ p q) ) 'C2)) ))))
```

On peut construire de même l'inversion et la mutation et rien n'empêche de concevoir d'autres transitions. A chaque instant on a une population P de np chromosome, chaque chromosome est constitué d'un nombre ng de gènes précédés de la valeur v d'une fonction que l'on cherche à minimiser.

14-23° Programmer les transitions nécessaires à un renouvellement d'une population en les produisant aléatoirement. Ces transitions opérant sur des chromosomes de longueur variable, et choisissant à chaque fois leurs paramètres aléatoires (ce ne sont plus des fonctions mais des schémas de fonctions). On peut prendre par exemple comme chromosome, un ensemble de règles de n hypothèses et p conclusions.

On choisit de représenter les chromosomes sous la forme (a b (p c ((c1 c2 ... cp) h1 h2 ... hn) (p c ((c1 c2 ... cp) h1 h2 ... hn) ... (p c ((c1 c2 ... cp) h1 h2 ... hn))
a, b (il peut y en avoir davantage) sont des paramètres numériques et les règles sont des listes de predicats précédées d'une priorité p et d'un coefficient c, les prédicats sont dans la liste PRED.

Comme opérateurs, nous construisons la création aléatoire de règle, la suppression, le bruit apporté à un paramètre numérique et aussi des mutations telles que "prendre le prédicat suivant ou précédent dans la liste de ceux-ci.

```
(de hasard (L) (nth (random 0 (length L)) L)); renvoie un élément tiré au hasard dans la liste L
(de poisson (m) (poissonbis (exp (- m)) 0 (divide (random 0 100) 100)))
(de poissonbis (ex n x)
        (if (< x ex) n (poissonbis ex (1+ n) (* x (divide (random 0 100) 100)))))

(de nouvregle (n p) (let ((r nil) (c nil)) ; construction d'une nouvelle règle
        (repeat n (set 'r (cons (hasard PRED) r)))
        (repeat p (set 'c (cons (hasard PRED) c))) (regulbis (mcons (poisson 1) 0 c r))))

(de produc (k) (let ((pop nil)(c nil)) ; produit une population de k chromosomes aléatoires
        (repeat k (set 'c nil) (repeat (1+ (poisson 3)) (set 'c (cons (nouvregle NH NC) c)))
        (set 'pop (cons (print '" Nouvelle solution aléatoire "
        (mcons  (// (random 3 9) 10.) (// (random 10 30) 10.) (arrange (tri c)))) pop))) pop))
(de aplat (L) (cond ((null L) nil); pour aplatir une liste
        ((atom (car L)) (cons (car L) (aplat (cdr L))))
        (t (append (aplat (car L)) (aplat (cdr L))))))
(de recons (L n p) (ifn (null L) ; L sera une liste de règles de la forme (pr f (a b ... ) c d ... )
        (cons (mcons (car L) (cadr L) (firstn p (cddr L)) (firstn n (nthcdr (+ 2 p) L)))
                (recons (nthcdr (+ n p 2) L) n p))));
; Suivant et Precedent sont des transitions remplaçant un gène au hasard par son voisin

(de creation (C) (mcons (car C) (cadr C) (arrange (tri (cons (nouvregle nh nc) (cddr C))))))

(de suppression (C) (let ((p (random 2 (length C))))
        (if (null (cdddr C)) (creation C) ; on ne supprime pas une règle unique
        (mcons (car C) (cadr C) (arrange (append (firstn (- p 2) (cddr C)) (nthcdr (1+ p) C)))))))

(de mutation (C) (transfo 'mutt C)) (de suivant (C) (transfo 'suivv C)) (de precedent (C) (transfo 'precedd C))

(de transfo (f C) (mcons (car C) (cadr C) (recons (transfobis f (aplat (cddr C))) nh nc)))
(de transfobis (f L) (let ((p (random 0 (length L)))) (while (numberp (nth p L)) (set 'p (random 0 (length L))))
        (append (firstn p L) (cons (funcall f (nth p L) PRED) (nthcdr (1+ p) L)))))

(de mutt (X L) (let ((nouv (hasard L))) (if (eq nouv X) (mutt X L) nouv)))
(de suivv (X L) (cond ((null (cdr L)) 'ANY) ((eq (car L) X) (cadr L)) (t (suivv X (cdr L)))))
(de precedd (X L) (suivv X (reverse L)))

(de priorite (C) ; modifie de ±1 la priorité d'un règle
        (let ((p (random 2 (length C)))) (append (firstn p C) (cons
        (cons (+ (car (nth p C)) (hasard '(-1 1))) (cdr (nth p C))) (nthcdr (1+ p) C)))))

(de bruit (C) ; modifie l'un des deux premiers paramètres en ajoutant une v.a. gaussienne
        (let ((p (random 0 2)) (br (divide (- (random 1 10) 5) 20)))
        (if (eq p 0) (cons (+ (car C) br) (cdr C)) (mcons (car C) (+ (cadr C) br) (cddr C)))))
; Transpo : 2 gènes au hasard sont inversés
(de transpo (C) (mcons (car C) (cadr C) (recons (transpobis (aplat (cddr C))) nh nc)))

(de transpobis (L) (let ((p (random 0 (div (length L) 2))) (q (random 1 (length L))))
        (while (numberp (nth p L))  (set 'p (random 1 (div (length L) 2))))
        (while (or (numberp (nth q L)) (<= q p)) (set 'q (random p (length L))))
        (append (firstn p L) (cons (nth q L) (firstn (- q p 1) (nthcdr (1+ p) L)))
                (cons (nth p L) (nthcdr (1+ q) L))) ))

(de crossover (C1 C2) ; crossover C1 C2 retournera une liste dont les car et cdr seront les deux fils
        (let ((r (min (length C1) (length C2))) (p 0) (q 0)); p indice du premier et q longueur de la sous-suite
        (setq p (random 0 (1- r)) q (random 1 (- r p)))
        (cons (regul (append (firstn p C1) (firstn q (nthcdr p C2)) (nthcdr (+ p q) C1)))
                (regul (append (firstn p C2) (firstn q (nthcdr p C1)) (nthcdr (+ p q) C2))))))
```

```
(de supmin (C) (if  (< (length C) 4) (supmin (creation C))
        (mcons (car C) (cadr C) (retmin nil 100 (cddr C))))) ; supprime la règle la moins forte

(de retmin (L m LR) (cond ; on suppose la force d'une règle entre 0 et 100
        ((null LR) (retbis L m))
        ((< (cadar LR) m) (retmin (cons (car LR) L) (cadar LR) (cdr LR)))
        (t (retmin (cons (car LR) L) m (cdr LR)))))
(de retbis (L m) (cond
        ((null L) L)
        ((eq (cadar L) m) (cdr L))
        (t (cons (car L) (retbis (cdr L) m)))))
```

"Regul "est une fonction propre au problème considéré, elle doit vérifier que le chromosome est une solution valide, et le cas échéant, doit en modifier aléatoirement les paramètres afin qu'il le devienne.

14-24° Programmer la fonction "generation" qui, à une population P et pour des probabilités p_m et p_c fixées pour l'application d'une mutation ou du crossover, produit une nouvelle population de même taille. On appliquera en outre la "reproduction" où chaque individu se recopie avec une probabilité inversement proportionnelle à sa valeur. On cherche alors à minimiser une telle valeur, par exemple le nombre de zéros dans une chaîne de 0 et de 1.

14-25° Quelle stratégie de renouvellement pourrait-on imaginer pour les transitions elles-mêmes ?

On choisit la stratégie d'évolution suivante où à chaque génération on applique les premières transitions dans l'ordre sur les premiers chromosomes de P. (Il n'est plus question de probabilités). On renforce ou punit les opérateurs suivant :
Pour chaque transition t de poids p agissant sur C de valeur v (si à une place), on obtient le résultat (v' C')
Si v' < v alors p est décrémenté, et incrémenté si si v' > v, C' est alors rajouté à P
        (Au cas du crossover on tient compte de la comparaison de v'1+v'2 avec v1+v2)
Lorsque toutes les transitions ont opéré, P est trié suivant v croissant et tronqué aux np premiers éléments. De même que chaque chromosome sera précédé de sa valeur, chaque opérateur est un corps de fonction précédée par sa contre-performance p (que l'on cherche également à minimiser et par le nom de la famille. On a donc la forme de données :
        P = ((4 A B A) (5 D E V) (5 D A B) (6 A A V)....)
On a de même unes liste OP de nop (nous recommendons de prendre le double de np) de transitions dont on fait opérer les nt (une partie) sur les premiers chromosomes de P.
        OP = ((-2 crossover lambda (C1 C2) (...)) (-1 copig lambda (C) (...)) .....)
Afin de renouveller OP lui-même pour le faire évoluer vers les bonnes transitions tout comme P évolue vers de bonnes solutions, on produit aléatoirement une nouvelle transition du type de la meilleure (des premières). Si aucune amélioration n'a été constaté à la génération courante, alors on réinitialise la population des opérateurs en les brouillant aléatoirement, ceci pour éviter la convergence vers un opérateur unique qui ne servirait plus. "Valeur" est ici la seule fonction qui sera propre au problème considéré, P et OP sont des variables globales.

```
(de initrans (n) (print "'On réinitialise les opérateurs.") (set 'OP nil) ; donne n transitions
        (until (< n (length OP)) (set 'OP (append OP  (mapcar (lambda (x) (list 0 x))
                '(mutation crossover supmin suppression bruit suivant  ranspo precedent creation)))))
        (set 'OP (firstn n (mapcar 'cdr (tri (mapcar (lambda (x)(cons (random 0 99) x)) OP))))))
(de generation (P OP) ; renvoie la liste dont le car est le nouveau P et le cdr, le nouvel OP triés.
        (gen1 nil P nil OP 0))

(de gen1 (PV PR OPV OPR k) (cond ; parties vues et restantes de P et OP, k est un indice
        ((or (null OPR) (null PR) (eq k np))
                (cons (firstn np (elimf (tri (append PV PR)))) (genop (tri (append OPV OPR)))))
        ((member (cadar OPR) '(crossover crossover1 crossover2)) ; et autres opérateurs à 2 places
```

```
                (gen4 (car OPR) (car PR) (if (and PR (< k (div np 2))) (hasard PR) (hasard PV))
                        PV (cdr PR) OPV (cdr OPR) (1+ k)))
        (t (gen2 (car OPR) (regul  (funcall (cadar OPR) (cdar PR))) (cons (car PR) PV)
                (cdr PR) OPV (cdr OPR) (1+ k))) ))

(de gen2 (oper res PV PR OPV OPR k) ; effectue les évaluations sur les résultats "res" obtenus par "op"
        (if (or (null res) (equal (cdar PV) res)) (gen1 PV PR (cons oper OPV) OPR (1+ k))
                ; on continue si res = op(res)
        (gen3 oper (valeur res) PV PR OPV OPR k)))          ; pas le crossover

(de gen3 (oper res PV PR OPV OPR k) ; évalue l'opérateur "op" suivant les "valeurs" en tête de "res"
        (print oper '"  " (car PV)'" --> " res)             ; on garde le meilleur entre le père et le fils
        (if (< (car res) (caar PV)) (set 'drap nil))
        (gen1 (if (< (caar PV) (car res)) PV (cons res (cdr PV )))       ; puis renforcement -1, 0, 1 de l'opérateur
                PR  (cons (cons (+ (car oper) (- (car res) (caar PV))) (cdr oper)) OPV) OPR k))

(de gen4 (oper C1 C2 PV PR OPV OPR k)
        (gen5 oper (funcall (cadr oper) (cdr C1) (cdr C2)) C1 C2 (ret C2 PV) (ret C2 PR) OPV OPR k))

(de gen5 (oper res C1 C2 PV PR OPV OPR k) ; évalue les résultats du cross-over
        (if (or (null C2) (null (car res)) (null (cdr res))) (gen1 PV PR (cons oper OPV) OPR k)
                (gen6  oper (valeur (car res)) (valeur (cdr res)) C1 C2 PV PR OPV OPR k)))


(de gen6 (oper C1r C2r C1 C2 PV PR OPV OPR k) ; évalue "op" qui est un cross-over
        (print oper '"  " C1 C2 '"-->" C1r C2r)
        (if (< (+(car C1r) (car C2r)) (+ (car C1) (car C2))) (set 'drap nil))
        (gen1 (append (firstn 2 (tri (list C1 C2 C1r C2r))) PV) PR ; les 2 meilleurs chromosomes dans PV
                (cons (cons (- (+ (car oper) (car C1r) (car C2r)) (car C1) (car C2))  (cdr oper)) OPV) OPR k))

(de genop (L)  ; renvoie la liste d'opérateurs L précédée d'un nouvel opérateur du type du premier de L
        (if (< 0 (caar L)) L (print '" Nouvelle transition créée : " (cadar L))
                (firstn (* 2 NP) (cons (list 0 (cadar L)) L))))

(de sgn (x) (cond ((< 0 x) 1) ((> 0 x) -1) (t 0))) ; trois sortes de signes
(de ret (X L) (cond ((null L) L) ((equal X (car L)) (cdr L)) (t (cons (car L) (ret X (cdr L))))))
(de elim (L) (cond ; permet d'éliminer les répétitions de L c'est à dire de ne conserver que des individus distincts
        ((null L) L)   ; élimination faible
        ((numberp (car L)) (cons (car L) (elim (cdr L))))
        ((member (car L) (cdr L))  (elim (cdr L)))
        (t (cons (car L) (elim (cdr L)))) ))
(de enchaine (n) ; enchaîne n générations en modifiant les globaux NUM, P, OP et STAT, lit np, nt et nop
        (let ((etat nil) (drap t))(until (or (eq num n) (< (caar P) 40)); teste si on est est sûr du coût 0 optimal
        (set 'STAT (cons (list (caar P)(caar (last P))) STAT)) ; drap = "pas d'amélioration"
        (set 'P (append (firstn (- np nt) P) (valoriser        ; état est constitué par ((P) OP)
                (produc (+ nt (- np (length P))))))) ; permet de rajouter NT individus aléatoires
        (setq drap t etat (generation P OP) P (car etat) OP (if drap (initrans (* 2 NP)) (cdr etat)))
        (print '"Génération " (set 'num (1+ num)) '" est finie. Valeurs: " (mapcar 'car P) '" Transitions: " OP))))

(de valoriser (LI) (tri (mapcar 'valeur LI))) ; permet de calculer la valeur des individus d'une population

(de fusion (L1R L2R LF) (cond ; où chaque élément de LR est de la forme (clé ... code de l'état...). Tri par
fusion
        ((null L1R) (if (null L2R) (reverse LF) (fusion nil (cdr L2R) (cons (car L2R) LF))))
        ((null L2R) (fusion nil (cdr L1R) (cons (car L1R) LF)))
        ((< (caar L1R) (caar L2R)) (fusion (cdr L1R) L2R (cons (car L1R) LF)))
        (t (fusion L1R (cdr L2R) (cons (car L2R) LF))) ))
(de sep (pair LP LI LR) ;liste des éléments de rangs pairs, impairs et restant à séparer
        (if LR    (if pair (sep nil (cons (car LR) LP) LI (cdr LR)) (sep t LP (cons (car LR) LI) (cdr LR)))
                (cons LP LI)) ) ; renvoie ((a0 a2 a4 ...) a1 a3 a5 ....)
(de tri (L) (if (null (cdr L)) L (tribis (sep t nil nil L))))
(de tribis (LC) (fusion (tri (car LC)) (tri (cdr LC)) nil))
```

14-26° En prenant des lettres au hasard, chercher à minimiser la somme des codes ASCII

```
(setq ng 5 np 40 nop 20 nt  15)
(set 'P '  ((a z e r t) (o f u h i) (k o p i l) (o t a r i) (e v b o j) (r a y u i) ( q s d f g) (i g h j k)
           (u g h a k) (j u y b o) (h e z s x) (h j n o c) (h a q z e) (j o k u h) (t y r e d) (g h c v i)
           (d f g u p) (r a s e r) (g u b z a) (a r i t o) (g o z a i) (b e n z a) (j o u t e) (h a s i f)
           (r u t a g) (g o p e d) (k a l o m) (b i n o c) (o u s t e) (v l a n h) (e h t o l) (t r u c q)
           (r a g e n) (t r o k i) (i o u l a) (k r o b u) (j l u t a) (k a s d o) (r e f u p) (t a r u p)))

(set 'P (mapcar (lambda (x) (cons (valeur x) x)) P)) ; on initialise une population de 40 chromosomes

(de initrans (nop) (set 'OP nil)

(repeat (div nop 7) (set 'OP (append OP (mapcar (lambda (f) (mcons 0 f (funcall f ng)))
          '(copig copid transpo transuiv crossover transd transg))))))

(de enchaine (n) ; enchaîne n générations en modifiant les globaux P et OP
          (let ((g 0)) (repeat n (let ((etat (generation P OP)))  (set 'P (car etat)) (set 'OP (cdr etat)) (set 'g (1+ g))
                                    (print '"La génération " g '" est achevée."))))))
(de valeur (C) (apply '+ (mapcar 'cascii C)))
```

A la onzième génération le min (485 a a a a a) est trouvé, on observe que copie à gauche est le type de transition le plus efficace alors que naturellement transpo et crossover ne modifiantrien à "valeur" sont rejetés en queue de OP.

14-27° **Trouver les différents minimums d'une fonction** (une sinusoïde amortie) ou reprendre le polynôme de l'exercice sur le recuit simulé, en retardant l'utilisation du cross-over.

On choisit la fonction f (a, b, c, d) = 1 + cos (π + 4π (0,abcd)) où a, b, c, d sont les 4 chiffres décimaux d'un nombre compris entre 0 et 1. On connaît donc à l'avance les 3 minimums qui sont obtenus pour (0 0 0 0), (5 0 0 0) et (9 9 9 9).

```
(setq ng 10 np 50 nop 50 nt 50 pi 3.14159 G '(0 1 2 3 4 5 6 7 8 9))      ; les gènes sont les chiffres de 0 à 9
(de produc (np) (set 'p ())              ; produit des chromosomes (une liste de 50)
          (repeat np (let ((c ()))
                    (repeat ng (set 'c (cons (nth (random 0  (length g)) g) c)))
                    (set 'p (cons c p))))   p)
(de valeur (C) (1+ (cos (* pi (1+ (* 4 (valbis 0 (reverse C))))))))         ; c'est la sinusoïde amortie
(de valbis (r C) (if (null C) (* 0.1 r) (valbis (+ (car C) (* 0.1 r)) (cdr C))))
(set 'P (tri (mapcar (lambda (x) (cons (valeur x) x)) (produc np))))       ; est la population aléatoire initiale
(initrans) (enchaine 25)    ; pour enchaîner 25 générations
```

En 4 générations, ils sont tous obtenus, et après 25 générations :

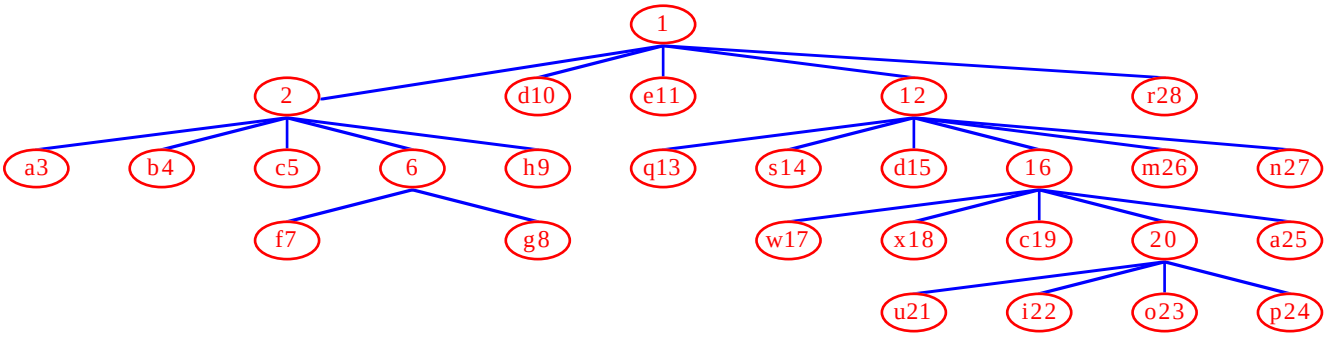
((0 0 0 0 0) (0 5 0 0 0) (0.0000007748 0 0 0 1) (0.0000007748 4 9 9 9) (0.0000008344 9 9 9 9) (0.000003159 4 9 9 8) (0.000003159 5 0 0 2) (0.000003159 0 0 0 2) (0.000003218 9 9 9 8) (0.000007152 9 9 9 7) (0.000012576 5 0 0 4) (0.00001972 0 0 0 5) (0.00001978 4 9 9 5) (0.00001978 9 9 9 5) (0.0000284 4 9 9 4) (0.0000285 9 9 9 4) (0.0000386 0 0 0 7) (0.0000387 4 9 9 3) (0.0000388 9 9 9 3) (0.00005048 5 0 0 8) (0.00005048 0 0 0 8) (0.00005060 4 9 9 2) (0.00005066 9 9 9 2))

14-28° **La voie royale,** il s'agit simplement de minimiser la fonction qui compte le nombre de zéros dans une chaîne de 32 bits.
Utiliser ce problème pour lequel la fonction d'évaluation est rapide à calculer et le minimum facile à atteindre pour faire des comparaisons entre les stratégies génétiques classique et celle du 25°. La mesure de la performance des différentes approches sera le nombre d'évaluations de la fonction pour parvenir à son minimum.

14-29° Programmation génétique d'une courbe séparatrice [Koza 92]. On souhaite séparer un ensemble de points-exemples en deux classes prédéfinies. Pour cela on constitue un ensemble de points du plan (au moins 100) chacun étant affecté à une classe A ou B. Un contrôle peut être effectué en prenant des points "de part et d'autre d'une fonction" comme y = 2x - 3, y = x^3 - 3x + 1, y = sin x (prévoir au moins 3 tests) ...
Le but étant de retrouver une telle fonction, ou d'en trouver une (relativement simple) si on ne donne que les points sans la séparatrice. On forme alors un ensemble de gènes à zéro place qui sont la variable x et les constantes (des nombres décimaux de dixièmes en dixièmes par exemple), des fonctions unaires (sin, exp ...) et des gènes à deux places (+, -, * ...), un chromosome étant un arbre bien formé (par exemple (+ (* x x) (* 3 x))). L'évaluation de ce chromosome pour un point (x, y) dira s'il se trouve au dessus ou en dessous de la fonction représentée par le chromosome.
L'algorithme génétique consiste alors à partir d'une population aléatoire de tels chromosomes, et d'opérer de générations en générations des mutations : un bruit sur une constante comme ajouter 0.1, une mutation de constante en x ou l'inverse, une mutation plus générale remplaçant un sous-arbre par une expression aléatoire, le cross-over échangeant des sous-expressions entre deux chromosomes ...
La performance d'un chromosome est mesurée par le nombre de points exemples bien placés dans leur classe, mais on peut faire intervenir également la complexité de l'expression représentée par le chromosome.
Extensions possibles, mettre en oeuvre l'algorithme pour des points de R^3 ou d'une dimension supérieure et simplification des expressions obtenues par diverses règles : développer et ranger les polynômes, effectuer les calculs numériques lorsqu'il y en a ...

Indication pour le crossover (la mutation est du même ordre mais plus facile) :

```
(defun noeud (E) (if ; donne le nombre total de noeuds de l'arbre C, donc 3 pour '(a b)
        (atom E) 1 ; les noeuds sont parcourus dans l'ordre racine gauche droite
        (+ (noeud (car E)) (noeud (cdr E)))))
(defun sousarbre (C q) (cond ; donne le sous-arbre débutant au noeud d'indice q, 1 pour la racine
        ((eq q 1) C) ; mais renvoie nil si q est > noeud de C, et erreur si q <= 0
        ((eq q 2) (car C))
        ((< (1+ (noeud (car C))) q) (sousarbre (cdr C) (- q (noeud (car C)))))
        (t (sousarbre (car C) (1- q)))))
(set 'A '((a b c (f g) h) d e (q s d (w x c (u i o p) a) m n) r)) ; ainsi c est en 5, (f g) en 6, w en 17, et r en 28
```




```
; exemple (sousarbre A 20 ))        ; donne (u i o p)
(defun substitue (C q R) (cond      ; renvoie la liste C où le q-ième sous-arbre est remplacé par R
        ((eq q 1) R)                ; exemple (substitue A 12 'x) renvoie ((a b c (f g) h) d e x r)
        ((eq q 2) (cons R (cdr C)))
        ((< (1+ (noeud (car C))) q) (cons (car C) (substitue (cdr C) (- q (noeud (car C))) R)))
        (t (cons (substitue (car C) (1- q) R) (cdr C)))))
(defun crossover (C1 C2)            ; (attention (random a) donne un entier entre 0 et a-1 en X-lisp
        (let ((p (1+ (random (noeud C1)))) (q (1+ (random (noeud C2)))))
        (let ((A1 (sousarbre C1 p)) (A2 (sousarbre C2 q))) (cons (substitue C1 p A2) (substitue C2 q A1)))))
(set 'B '(b (x y z) d (e f))) ; Exemple (crossover A B) peut donner par exemple (en majuscules les sous-arbres échangés) :
        ((a b c (f g) h) d e (q s d (w x c E a) m n) r)) et (b (x y z) d ((U I O P) f))
```

```
(set 'm 7)  ; profondeur maximale
(set 'op0  '(x 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0))  ; constantes
(set 'op1 '(opp sin cos exp))  ; opérateurs unaires
(set 'op2 '(+ * -))  ; opérateurs binaires
(defun hasard (L) (nth (random (length L)) L))
(defun mig (k) (cond ; création aléatoire d'un individu de profondeur k
        ((eq k 0) (hasard op0))
        ((eq k 1) (list (hasard op1) (mig 0)))
        ((eq 1  (random 2)) (list (hasard '(opp sin cos exp)) (mig (1- k))))
        (t (list (hasard op2) (mig (1- k)) (mig (1- k)))  )))
(defun mut (A) (let ((n (noeud A))) (let ((p (1+ (random n))))
        (substitue A p (mig (random (1+ (prof (sousarbre A p))))))  )))
(defun construc (n) (if (< 0 n) (cons (mig (random m)) (construc (1- n))) ))
```

14-30° Problème de la fourmi de Santa-Fe [Koza 92]. On demande la réalisation d'une application sur le thème suivant ou inspiré par celui-ci : dans une grille (par exemple 20*80), certaines cases F représentent de la nourriture, d'autres M des obstacles, et d'autres P des prédateurs. Avec un point de départ (x, y) et une direction initiale, (4 directions possibles) un programme écrit sous forme d'arborescence doit en s'éxecutant, modifier la position (x, y) d'une fourmi, ainsi que sa direction.
On prendra toujours le même départ dans la même grille.
Un individu est un programme exprimable par un arbre (car il y aura des alternatives telles que "si mur-devant alors" "si pred alors" ...) où on aura des "gènes" représentant des instructions élémentaires (gauche, droite, avancer ..) et, outre les alternatives, des séquences comme par exemple (seq avancer gauche). Un "chromosome" est donc un programme, c'est à dire un objet structuré tel que, par exemple :
(seq (seq (seq avancer gauche) (si-nourriture-devant droite (seq avancer gauche)))
(si-mur-devant droite avancer)).
Un individu pourrait être évalué de la manière suivante, on répète (par exemple 40 fois) le programme, tant que la fourmi se trouve dans la grille, sa valeur obtenue est alors :
(cardinal de l'arbre représentant le programme)
+ (nombre de cases M rencontrées, la fourmi se trouvant dessus)
+ 2(nb de cases P rencontrées, la fourmi étant sur la même case)
- 5(nb de cases F rencontrées, la nourriture est alors retirée)
On cherche à minimiser cette fonction d'évaluation grâce à un algorithme génétique simple :
1 Une population aléatoire (de mettons 20 individus) est formée.
2 A chaque génération vont s'appliquer de manière aléatoire des opérateurs génétiques tels que la mutation (remplacement d'une instruction par une autre) et le croisement (deux individus échangent un sous-arbre). On peut imaginer d'autres opérateurs.
3 Aussitôt une évaluation des "enfants" est faite de façon à ne conserver que le meilleur entre un parent et un enfant (ou bien les deux meilleurs des 4 si croisement). Cette évaluation peut être visualisée à l'écran par le parcours de l'individu.
4 La population est triée suivant cette évaluation, les répétitions sont éliminées, des individus aléatoires sont éventuellement crées pour maintenir toujours le même nombre d'individus.
5 Retour en 2 ou arrêt.
Pour l'expérimentation, on réfléchira en outre à plusieurs modifications : représenter la fourmi par ses coordonnées uniquement, les "gènes" de déplacements étant les huit directions E, NE, N, NO, O, SO, S, SE; autres heuristiques définissant la valeur de la fourmi ...