Apprentissage par rétropropagation du gradient de réseaux neuromimétiques multi-couches

Bernard ESPINASSE

2008

1 - formalisation mathématique

2 - implémentation dans BP de PDP

3 - un exemple complet: le XOR

Type de réseau neuromimétique

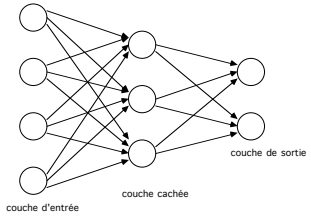
- Réseau d'automates structurés en couches :

- premiere couche (indice 0) reçoit des entrées,
- la dernière (indice n), couche de décision, produit les sorties.
- couches intermédiaires: couches cachées

- Connexions :

- seules connections possibles: relient une couche à une couche d'indice supérieur
- pas de connexions intra-couches

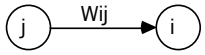
En se limitant à des couches à 1 dimension, exemple d'un tel réseau à 3 couches:



(il peut y avoir plusieurs couches cachées)

Modèle de cellule

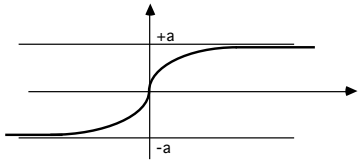
Chaque cellule **i** du réseau calcule sa sortie (output) **oi** comme une fonction différentielle **f (fonction d'activation)** de la somme pondérée **Ai** de ses entrées oj en provenance des cellules **j** de la couche précédente:



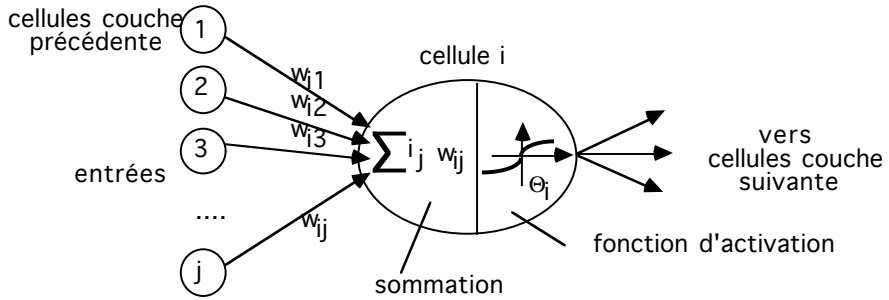
$$o_i = f(A_i) \text{ avec } A_i = \sum_j w_{ij}\ o_j$$

où **wij** est le poids de la connexion de la cellule j vers la cellule i.

- fonctions d'activation choisies sont en général des fonctions de **type sigmoïde** :



Le modèle de d'automate retenu est le suivant:

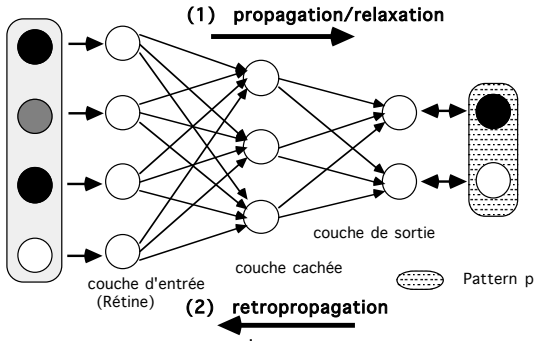


avec **Θ_i = seuil** de la fonction d'activation du neurone **i** encore appelé **biais$_i$** .

Principe général de la rétropropagation

- même principe que pour le **perceptron**

- se fait à partir d'un ensemble d'exemples constitués par des **couples (entrées, sorties désirées)**.



Principe général de la rétro-propagation

Phase de propagation avant (relaxation du réseau) :

- un exemple (pattern) p est présenté en entrée du réseau, une sortie réelle est calculée de proche en proche de la couche d'entrée vers la couche de sortie:

Phase de rétro-propagation :

- **on calcule l'erreur** comme étant la somme quadratique des erreurs sur chaque cellule de la couche de sortie

- l'erreur est ensuite **retro-propagée** dans le réseau, de la sortie vers l'entrée, en **effectuant une modification de chaque poids** des connexions du réseau; c'est la

- **processus répété en présentant successivement chaque exemple**
- **si pour tous les exemples présentés, l'erreur< seuil choisi, alors le réseau a convergé.**

Principe général de la rétro-propagation

- On cherche à **minimiser une fonction d'erreur E=∑** erreurs quadratiques entre les sorties désirées et les sorties calculées par le réseau et ce sur chaque neurone de sortie et pour chaque pattern.

- L'idée de base est de pouvoir analyser **l'influence d'un poids d'une cellule cachée sur la sortie**.

- Cette influence est évaluée en calculant le **gradient** (dérivée partielle) de la fonction erreur E par rapport à un poids donné **wij**

- et en **propageant cette valeur dans le sens inverse** soit de la couche de sortie vers la couche d'entrée (rétro-propagation de l'erreur)

Formalisation mathématique de la rétro-propagation

- On calcule **la sortie globale Op du réseau pour ce pattern donné p**.
- Minimiser l'erreur quadratique entre Tp (sortie désirée) et Op revient à minimiser une fonction de l'ensemble des poids de connexions

$$E(W) = 1/m \sum_p E_p \quad \text{avec} \quad E_p = \sum_n (o_{np} - t_{np})^2$$

(n indice uniquement les cellules de la couche de sortie et p le pattern présenté)

- Au lieu de minimiser directement E, **on cherchera à minimiser successivement l'erreur sur chaque pattern c'est à dire les Ep**

(*pour simplifier la présentation omettons l'indice p, identifiant le pattern, notons E pour Ep)*

Minimiser E (en fait Ep) nécessite de connaitre les gradients de E par rapport à W, soit $\frac{dE}{dw_{ij}}$.

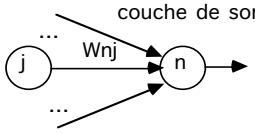
- Plutôt que de calculer ces gradients, on préfère calculer les gradients de E par rapport aux entrées totales Ai, c'est à dire les $\frac{dE}{dA_i}$, avec $A_i = \sum_j w_{ij} o_j$.

on a: $\frac{dE}{dw_{ij}} = \frac{dE}{dA_i} \cdot \frac{dA_i}{dw_{ij}}$ (règle de dérivation des fct. composées)

comme $A_i = \sum_j w_{ij} o_j$ on a: $\frac{dA_i}{dw_{ij}} = o_j$ d'où $\frac{dE}{dw_{ij}} = \frac{dE}{dA_i} \cdot o_j$.

Pour les cellules de sorties

Considérons les cellules de la couche de sortie que nous indicerons par l'indice n :



$\frac{dE}{dA_n}$ associés aux cellules n sont facilement calculables:

$$E = \sum_n (o_n - t_n)^2$$

(pour un pattern p donné, t_n étant la sortie désirée sur la cellule n, et o_n la sortie calculé pour la cellule n)

or $\frac{dE}{dA_n} = \frac{dE}{do_n} \cdot \frac{do_n}{dA_n}$ d'òu $\frac{dE}{dA_n} = 2(o_n - t_n) \cdot \frac{do_n}{dA_n}$

mais $o_n = f(A_n)$ d'où: $\frac{dE}{dA_n} = 2(o_n - t_n) \cdot \frac{df(A_n)}{dA_n}$

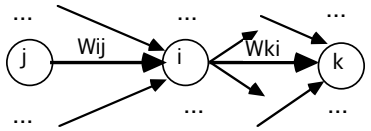
$\frac{df(A_n)}{dA_n}$ est la dérivé de f = f'.

on posera $\delta s = \frac{dE}{dA_n}$ d'où pour les cellules d'indice n de la couche de sortie:

$$\boxed{\delta s = 2(o_n - t_n).f'(A_n)}$$

Pour les cellules cachées (1)

Le gradient attaché à une cellule considérée i peut être calculé à partir des gradients attachés à toutes les cellules k auquelles elle envoie sa sortie o_i:



En effet (règle de dérivation sur les fonctions composées): $\frac{dE}{dA_i} = \frac{dE}{dA_k} \cdot \frac{dA_k}{dA_i}$

de même: $\frac{dA_k}{dA_i} = \frac{dA_k}{do_i} \cdot \frac{do_i}{dA_i}$; or $o_i = f(A_i)$ d'où $\frac{do_i}{dA_i} = \frac{df}{dA_i} . = f'(A_i)$

et $A_k = \sum_k w_{ki}\, o_i$ d'où $\frac{dA_k}{do_i} = w_{ki}$; d'òu $\frac{dA_k}{dA_i} = w_{ki}.f'(A_i)$

soit: $\frac{dE}{dA_i} = f'(A_i) \sum_k w_{ki} \frac{dE}{dA_k}$

Pour les cellules cachées (2)

soit encore en posant : $\delta i = \frac{dE}{dA_i}$

$$\boxed{\delta i = f'(A_i) \sum_k w_{ki}\, \delta k \quad \text{avec } \delta k = \frac{dE}{dA_k}}$$

- δpi = terme "delta" représente l'effet d'un changement de poids d'un neurone j sur la sortie du neurone i et ce pour le pattern p.
- Le gradient $\frac{dE}{dw_{ki}}$ se calcule à partir des δi évalués de proche en proche à partir de la couche de sortie n vers la couche d'entrée 0, en utilisant le poids "à l'envers" (rétro-propagation)

Algorithme de rétro-propagation (1)

1- relaxation du réseau

présentation d'un exemple **p** et calcul de l'état du réseau par propagation directe: chaque neurone calculant son état **opi** en suivant le sens entrée-sortie, soit :

- si i est une cellule d'entrée $o_i = x_{pi}$
- sinon $o_i = f(A_i)$ avec $A_i = \sum_j w_{ji}\, o_j$

2- présentation sortie désirée Tp et calcul des gradients

a- gradient δ_S sur la couche de sortie:

$$\boxed{\delta s = 2(o_n - t_n).f'(A_n)}$$

b- gradient d_i sur les couches cachées par rétro-propagation:

$$\boxed{\delta i = f'(A_i) \sum_k w_{ki}\, \delta_k \quad \text{avec } \delta_k = \frac{dE}{dA_k}}$$

Algorithme de rétro-propagation (2)

3- Phase de réajustement des poids: Apprentissage du réseau

Consiste en la modification des poids en utilisant ces gradients de la façon suivante:

$$\Delta p w_{ij} = -\varepsilon . \frac{dE}{dw_{ij}} .$$

$$\Delta w_{ij} = \varepsilon . \delta pi . opj$$

$$\text{soit} \quad w^{p}_{ji} = w^{p-1}_{ji} - \varepsilon . \delta_{\mathbf{i}} . o_{pj}$$

avec ε paramètre appelé **pas du gradient ou learning rate**.

4- On continue en 1 tant que E pas < à un seuil fixé à priori et ce en parcourant plusieurs fois la base d'exemples.

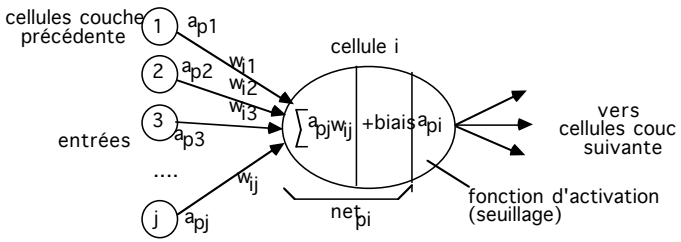
BP de PDP

Implémentation de la rétropropagation

- logiciel **PDP** (Parallel Distributed Processing),
- mis au point par **J.L.McClelland, D.E.Rumelhart et le PDP Group du MIT (Massachussetts Institute of Technology**)

BP : Modèle de la cellule

Modèle de la cellule



avec

biaisi = terme jouant un rôle équivalent à un seuil pour la fonction d'activation du neurone i

api = activationi = fonction d'activation (ou de seuillage) = outputi pour le pattern p= **opi**

netpi = somme nette des entrées du neurone i (avec biais) pour pattern p

$$\mathbf{net_{pi} = \sum_{j} w_{ij} . a_{pj} + biais_i}$$

BP : Modèle de la cellule

Fonction d'activation et expression de api

La fonction d'activation choisie est la fonction sigmoïde:

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

dont la dérivée s'exprime facilement

sig'(x) = sig(x).(1 - sig(x)).

d'où dans notre modèle de cellule:

$$\mathbf{a_{pi} = sig(net_{pi}) = \frac{1}{1 + e^{-net_{pi}}}}$$

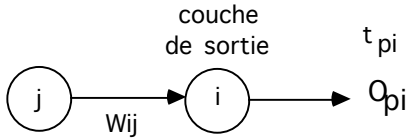
et donc

$$\mathbf{a'_{pi} = a_{pi} (1 - a_{pi})}$$

BP : Règle d'apprentissage (1)

Règle d'apprentissage :

fonction erreur E est la somme des erreurs quadratiques entre les sorties désirées t_{pi} et les sorties calculées par le réseau o_{pi} et ce sur chaque neurone i de sortie et pour chaque pattern p:



$$E = \sum_{pi} (t_{pi} - o_{pi})^2$$

La règle d'apprentissage est:

$$\Delta_p w_{ij} = -\varepsilon \frac{dEp}{dw_{ij}}$$

BP : Règle d'apprentissage (2)

Calcul de $\Delta_p w_{ij}$:

$$\Delta_p w_{ij} = -\varepsilon\, \delta_{pi} .o_{pj} = -\varepsilon\, \delta_{pi} .a_{pj}$$

Explicitons tout d'abord les valeurs des δ_{pi} .

$$\frac{dEp}{dw_{ij}} = \frac{dEp}{dnet_{pi}} \cdot \frac{dnet_{pi}}{dw_{ij}}$$

Or $net_{pi} = \sum_j w_{ij} .a_{pj} + biais_i$, on en déduit:

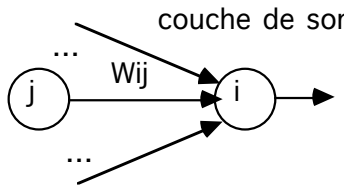
$$\frac{dnet_{pi}}{dw_{ij}} = a_{pj}$$

Posons $\delta_{pi} = \frac{dEp}{dnet_{pi}}$, on a:

$$\delta_{pi} = \frac{dEp}{da_{pi}} \cdot \frac{da_{pi}}{dnet_{pi}} \quad \text{avec} \quad \frac{da_{pi}}{dnet_{pi}} = a'_{pi}$$

$$\textbf{soit} \quad \delta_{pi} = \frac{dEp}{da_{pi}} . a'_{pi}$$

Pour les neurones i de la couche de sortie



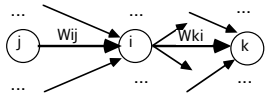
$$\frac{dEp}{da_{pi}} = (t_{pi} - o_{pi}) \quad \text{d'où} \quad \delta_{pi} = (t_{pi} - o_{pi}). a'_{pi}$$

$$\text{et} \quad \frac{dEp}{dw_{ij}} . = (t_{pi} - o_{pi}).a'_{pi}\, a_{pi}$$

Du fait de l'expression de a_{pi} on a $a_{pi}' = a_{pi} (1 - a_{pi})$ et $\Delta w_{ij} = -\varepsilon . \delta_{pi} .o_{pj}$ d'où:

$$\Delta_p w_{ij} = -\varepsilon\, (t_{pi} - o_{pi}) . a_{pi}\, ^2(1 - a_{pi})$$

Pour les neurones des couches cachées



$$\delta_{pi} = \frac{dEp}{da_{pk}} = \sum_k \frac{dEp}{dnet_{pk}} \cdot \frac{dnet_{pk}}{da_{pi}}$$

avec:

$$\frac{dEp}{dnet_{pk}} = \delta_{pk} ; \qquad \frac{dnet_{pk}}{da_{pi}} = w_{ki}$$

où l'indice k porte sur les neurones sur lesquels i envoie des connexions

donc $\delta_{pi} = . \sum_k \delta_{pk} .w_{ki} .a'_{pi}.$

Du fait de l'expression de a_{pi} on a pour les neurones des couches cachées:

$\Delta w_{ij} = -\varepsilon . \delta_{pi} . o_{pj}$ d'où

$$\Delta_p w_{ij} = -\varepsilon \sum_k \delta_{pk} .w_{kj} . a_{pi}\, ^2(1 - a_{pi})$$

Pas du gradient ou learning rate: e

- constante que l'on essaie de choisir la plus grande possible
- plus le pas est petit, plus le temps d'apprentissage est long
- Mais si on augmente trop la valeur du pas, alors on risque d'osciller autour d'un minimum local, ou de diverger.

Le momentum: α

- Introduisant dans le calcul de la variation de poids à l'instant t+1, la variation de poids qui vient juste d'être calculé à l'instant t:

$$\Delta wij\ (t+1) = \varepsilon . \delta pi . apj + \alpha . \Delta wij\ (t)$$

L'introduction du momentum permet de diminuer le temps d'apprentissage, tout en limitant les risques d'oscillation.

Conduite de l'apprentissage avec BP (1)

Connexions

- sans rebouclage sortie / entrée
- sans connexion entre neurones de la même couche
- possible de simuler de telles connexions en imposant des contraintes sur les poids (réseaux recurrents,séquentiels et en cascade) [RUMELHART 88].

Initialisation des poids

- plupart des problèmes requièrent des valeurs différentes pour les poids.
- Si ceux-ci sont égaux, alors comme l'erreur "delta" propagée est proportionelle à la valeur des poids.
- Ceux-ci vont donc changer pendant l'apprentissage, mais seront toujours égaux.
- Donc quelque soit la valeur initiale des poids, le réseau va converger vers la même valeur.

au lancement BP lit fichier **'xxx.str'** (startup) dans lequel on indiquera fichier **'.net'**. Ce fichier **'xxx.net'** (architecture réseau)

Conduite de l'apprentissage avec BP (2)

initialisation des poids peut être faite :

- imposer valeurs de départ: commande ***get weights <nom>.wts*** dans fichier **.str**,
- de façon aléatoire, grâce à la commande **%** lue dans le fichier **.net**,
- Dans le cas où les poids sont aléatoirement initialisés, on peut imposer des valeurs aléatoires, positives (positive random weight), négatives (negative random weight), ou quelconques (random weight).
- La variable ***wrange*** spécifie l'amplitude du nombre aléatoire générant les poids initiaux .
- Si la contrainte du poids est d'être positif, alors l'amplitude varie entre 0 et +wrange; dans le cas négatif, entre -wrange et 0; et sinon entre -wrange/2 et +wrange/2.
- De même on peut imposer aux poids des contraintes sur leurs évolution futur lors de la simulation. Les poids peuvent être liés (linked weigths), modifiables ou non.
- Notons qu'une contrainte spécifiée sur les poids liés est appliquée tout le long du déroulement du programme, alors que la contrainte sur la valeur initiale des poids n'est appliquée qu'en début de programme.

Remarques: biais traités dans bp comme des liens dont les poids sont égaux à la valeur du biais, et dont les entrées sont toujours à 1.

Modes d'apprentissage: par pattern et par epoch (1)

La back-propagation rétro-propage l'erreur "delta" dans le réseau. deux options possible dans BP:

- **Apprentissage par pattern:** On rétro-propage l'erreur "delta" après chaque présentation de pattern (**set lgrain pattern) ***
- **Apprentissage par epochs:** On accumule les erreurs "delta" de chaque pattern et on rétro-propage la somme obtenue lorsque l'on a parcouru toute la base des patterns (**set lgrain epoch)**.

Il semblerait que lorsque les poids sont changés après chaque pattern, on obtienne de meilleurs résultats surtout si le pas du gradient est petit et si la base des patterns est grande.

Déroulement de l'apprentissage

- commande **strain** (sequential training): parcours de la base d'exemples, soit de façon séquentielle (strain)
- commande **ptrain** (permuted training): parcours de la base d'exemples, soit de façon aléatoire mais néanmoins en ne présentant qu'une seule fois les exemples (ptrain).

Modes d'apprentissage: par pattern et par epoch (1)

Critères de performances: tss, pss et gcor.

- **ecrit** = error criterion: spécifiée par l'utilisateur. Par défaut ecrit = 0.0
- **tss** = total sum of squares: somme des erreurs quadratiques des sorties de tous les patternsde la base d'exemples.
- **pss** = pattern sum of squares: somme des erreurs quadratiques des sorties d'un pattern.

L'apprentissage se déroule tant que **ecrit > tss**, ou que **nepochs ≥ valeur initiale**
avec nepochs = number of epochs,
Nombre de passage de la base d'exemples.

gcor = « gradient correlation", vecteur de corrélation entre wed(t) et wed(t-1) permet de savoir si la valeur du gradient reste relativement stable d'une epoch à l'autre.

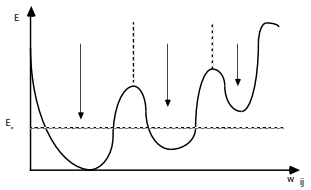
En effet le vecteur wed peut être considéré comme un vecteur pointant la direction la plus pentue lorsque l'on se trouve dans l'espace des poids, par exemple si gcor change de signe, alors ceci indique que le gradient change de direction (le minimum local ou global a peut être été dépassé).

Difficultés d'apprentissage (1)

Minimum local et minimum global

L'algorithme de la back-propagation effectue une **descente de gradient sur E** (E = somme des erreurs quadratiques entre les sorties désirées et les sorties calculées par le réseau) .

La fonction E peut aussi s'interpréter comme une fonction énergie, et il est possible de définir un "**paysage d'énergie**". On a, par exemple, en fixant tous les poids et en faisant varier un poids quelconque wij une variation de E de la forme [WEISBUCH 89]:



Suivant la configuration initiale, indiquée par les flèches, la procédure de descente de gradient va converger dans l'un quelconque des minima.

Difficultés d'apprentissage (2)

- Le problème posé est de trouver la configuration conduisant au minimum global. Un moyen d'y parvenir est de faire plusieurs essais à partir de configurations initiales choisies arbitrairement, et après avoir fait converger le réseau on ne garde que la configuration ayant abouti au minimum global ou au meilleur des minima locaux.
- On peut remarquer qu'avec cette méthode rien n'assure d'aboutir au minimum global: mais en fait dans le cas des réseaux neuronaux la procédure est légèrement différente par rapport aux calculs de recherche opérationnelle, en ce sens qu'on ne cherche pas obligatoirement le minimum global mais plutôt à atteindre une valeur de E, E0 fixée à priori.
- Ainsi, dans la plupart des cas le réseau ne trouve pas la meilleure solution, mais une solution qui convient: cette faculté de travailler avec du "flou" est une des particularité des réseaux neuronaux, ils ne trouvent pas des solutions exactes mais des solutions approchées.
- Pratiquement, on s'est rendu compte que les réseaux comportant peu de neurones cachés, avaient beaucoup plus de minima locaux, et inversement. D. Rumelhart et J. Mc Clelland proposent même dans [RUMELHART 88] que "les espaces ayant de grandes dimensions (c'est à dire avec beaucoup de poids) ont relativement peu de minima locaux".

Problème de la valeur du pas de gradient

- La procédure d'apprentissage modifie les poids des connexions d'une valeur proportionelle à la valeur delta. Cette proportionnalité est fonction de la valeur ***e*** encore appelée "***learning rate***" ou pas du gradient.
- La vraie procédure de descente de gradient requiert un pas infiniment petit, ce qui augmente de façon notable le temps de calcul. Mais une plus grande valeur risque de faire osciller le réseau dans le cas de paysage très pentu.
- En pratique, on essaye de choisir des valeurs pour le pas, les plus grandes possible, sans que cela conduise le réseau à osciller: Pratiquement dans nos essais on prend des valeurs relativement petites au départ (entre 0.01 et 0.25) pour ensuite les augmenter en se basant sur l'observation des variables tss et gcor.
- Si ***tss*** varie peu, alors on augmente le pas, et si ***gcor*** n'est pas stabilisé à une valeur, alors on diminue le pas (tss = total sum of square = E et gcor = gradient corrélation).

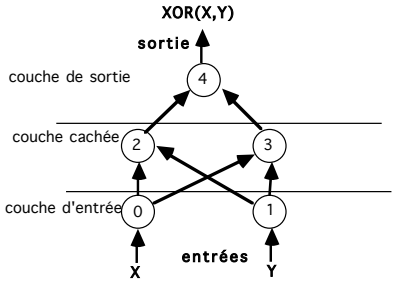
Exemple du XOR : architecture de réseau

L'exemple traite de la fonction booléenne du OU EXCLUSIF (XOR). Rappelons la table de vérité de la fonction XOR:

X	Y	XOR(X,Y)
1	1	0
1	0	1
0	1	1
0	0	0

Architecture du réseau

Nous allons construire un réseau qui aura en entrée X et Y et en sortie désirée XOR(X,Y).



Exemple du XOR : les fichiers (1)

xor.net

Ce fichier indique au programme l'architecture du réseau, et précise les contraintes d'initialisation et d'évolution des poids:

```
definitions:
nunits 5
ninputs 2
noutputs 1
end
network:
%r 2 2 0 2
%r 4 1 2 2
end
biases:
%r 2 3
end
```

Ce qui s'interprète ainsi:

- Le réseau est composé de 5 neurones. (nunits 5)
- Il y a 2 neurones d'entrée. (ninputs 2)
- Il y a 1 neurone de sortie. (noutputs 1)
- La première couche commence au neurone n° 0 et contient 2 neurones et est complètement connectée à la couche suivante qui commence au neurone n° 2 et qui contient 2 neurones. (%r 2 2 0 2)
- L'initialisation des poids est faite de façon aléatoire (%r).
- La couche commençant au neurone n°2 et qui contient 2 neurones est complètement connectée à la couche suivante qui commence au neurone n° 4 et qui contient 1 neurone. (%r 4 1 2 2).
- Les 3 neurones qui suivent le neurone n°2 ont des biais initialisés de façon aléatoire. (%r 2 3).

Exemple du XOR : les fichiers (2)

xor.str

Ce fichier contient une série de commandes que l'on envoie à l'initialisation du programme: acquisition de valeurs et configuration de variables

```
get network xor.net
get patterns xor.pat
set nepochs 30
set ecrit .04
set dlevel 3
set slevel 1
set lflag 1
set mode lgrain epoch
set mode follow 1
set param lrate .5
get weights xor.wts
tall
```

Ce qui s'interprète ainsi:

- L'architecture du réseau se trouve dans le fichier xor.net
- les exemples dans le fichier xor.pat
- Les paramètres fixés à l'initialisation sont:
- nepochs = 30: époque = 30
- ecrit = 0.4: critère d'erreur
- dlevel = 3: niveau d'affichage global
- slevel = 1: niveau de sauvegarde dans le fichier log
- lflag = 1: modification possible des connexions
- mode lgrain epoch: mode d'apprentissage par époque
- mode follow 1: calcul de la variable gcoor après chaque changement de poids
- param lrate .5: pas du gradient = 0.5
- tall: évaluation de tous les patterns de la base d'exemples un par un.

Exemple du XOR : les fichiers (3)

xor.tem

Ce fichier (template) permet de configurer l'affichage à l'écran. Cet affichage est exactement la présentation spécifiée.

```
define: layout
                              pname ipatterns   tpatterns
epoch  $      tss    $          $     $          $
           gcor   $
cpname $      pss    $

sender acts:     $                bia net act tar del

weights:         $                $  $  $     $

                                    $
end
epochno variable    1     $      n      epochno      5      1.0
tss     floatvar    1     $      n      tss          7      1.0
gcor    floatvar    2     $      5      gcor         7      1.0
cpname variable     2     $      n      cpname       -5     1.0
pss floatvar        2     $      n      pss          7      1.0
env.pname vector    0     $      2      pname        v 4    0 0    4
env.ipat matrix     0     $      n      ipattern     h 2    1.0    0  4   0  2
env.tpat matrix     0     $      n      tpattern     h 2    1.0    0  4   0  1
input vector        3     $      8      activation   h 4    100.0  0  4
weight matrix             3     $      n       weight h 4    100.0  2  3   0  4
bias vector         3     $      n      bias         v 3    100.0  2 15
netinput vector     3     $      n      netinput     v 3    100.0  2 15
outputvector        3     $      n      activation   v 3    100.0  2  3
deltas vector       3     $      n      delta        v 3 1000.0    2  3
targets vector      3     $      n      target       v 3    100.0  0  1
```

Exemple du XOR : les fichiers (4)

xor.pat

Ce fichier contient la base d'exemples (les patterns).

```
p00 0 0 0
p01 0 1 1
p10 1 0 1
p11 1 1 0
```


Exemple du XOR : Poids en début d'apprentissage (1)

Poids en début d'apprentissage

On peut à tout moment de l'apprentissage stocker dans un fichier les valeurs des poids des connexions du réseaux par la commande save. Voici le fichier des poids après initialisation aléatoire, poids de départ avant tout apprentissage.

```
0.432171
0.448781
-0.038413
0.036489
0.272080
0.081714

0.000000
0.000000
-0.276589
-0.402498
0.279299
```


Exemple du XOR : Poids en début d'apprentissage (2)

Interprétation d'un fichier de poids:

- Les 6 premiers chiffres sont les poids des six liens.
- Les 5 suivants sont les valeurs des biais (les deux biais d'entrée valant 0.00).

poids:

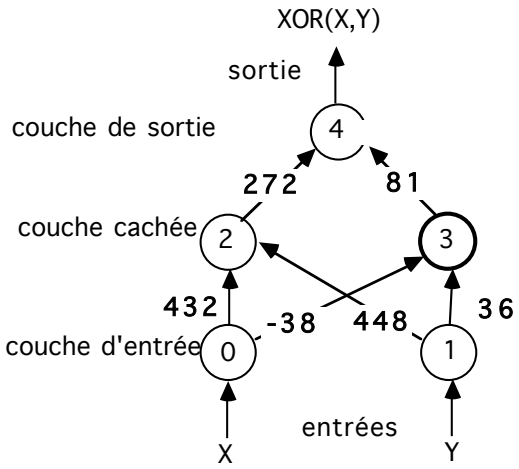
- ligne 1: poids du 1°neurone caché soit le neurone 2 provenant du neurone 0, c'est à dire w_{02},
- ligne 2: w_{02},
- ligne 3: w_{12},
- ligne 4: w_{03},
- ligne 5: w_{13},
- ligne 6: w_{34}.

biais:

- ligne 7: biais du neurone d'entrée 0,
- ligne 8: biais du neurone d'entrée 1,
- ligne 9: biais du neurone caché 2,
- ligne 10: biais du neurone caché 3,
- ligne 11: biais du neurone caché 4.

Exemple du XOR : Poids en début d'apprentissage (3)

Soit avant apprentissage:



Exemple du XOR : Poids en début d'apprentissage (3)

Ecran en apprentissage :

```
                                        pname   ipatterns   tpatterns
epoch    290     tss       0.0378          p00     0 0         0
                 gcor      0.9999          p01     0 1         1
cpname   p11     pss       0.0109          p10     1 0         1
                                           p11     1 1         0



sender acts:     100     100     99      82     bia    net    act  tar  del

weights:         572     573                    927    99     0
                 320     320                    152    82     9
                 643     -701                    10     0     -9
```

Remarque: Sur certaines versions de BP (sur SUN notamment), les affichages en noir indiquent des valeurs négatives.

Exemple du XOR : Poids en début d'apprentissage (4)

Ecran en fin d'apprentissage :

```
                                        pname   ipatterns   tpatterns
epoch    290     tss       0.0378          p00     0 0         0
                 gcor      0.9999          p01     0 1         1
cpname   p11     pss       0.0109          p10     1 0         1
                                           p11     1 1         0



sender acts:     100     100     99      82     bia    net    act  tar  del

weights:         572     573                    927    99     0
                 320     320                    152    82     9
                 643     -701                    10     0     -9
```

Remarque:

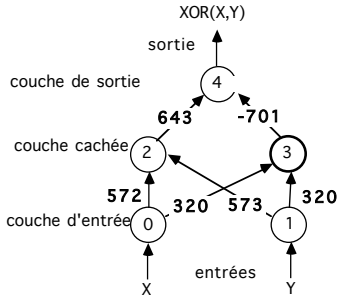
Sur certaines versions de BP (sur SUN notamment), les affichages en noir indiquent des valeurs négatives.

Exemple du XOR : Poids en début d'apprentissage (5)

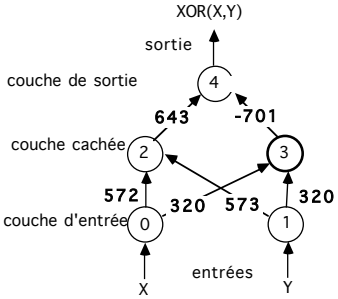
Poids de fin d'apprentissage :

```
5.728088
5.733082
3.203507
3.206242
6.436387
-7.014155
0.000000
0.000000
-2.187303
-4.873405
-2.843181
```

Soit:



Exemple du XOR : Poids en début d'apprentissage (6)



Interprétation :

- le neurone 2 fait le OU INCLUSIF:
  1 0 -> 1
  0 1 -> 1
  1 1 -> 1
  0 0 -> 0
- le neurone 3 (neurone inhibiteur), sépare le OU EXCLUSIF du OU INCLUSIF:
  1 1 -> 0

Exemple d'application: le 3 dans 8

Il s'agit de reconnaitre dans 8 bit une séquence de 3 bit a 1, par exemple:

1 1 1 0 0 0 0 0 donne 1
0 1 1 0 0 0 0 0 donne 0
1 0 1 1 1 0 0 0 donne 1
1 0 1 0 1 0 0 1 donne 0

Architecture du réseau: