Statistiques descriptives Analyse de données Algorithmes

(4) MODÈLES NON LINEAIRES

Jérôme Lacaille
Expert Émérite Safran

ARBRES DE DÉCISION

1. **RÉGRESSION**
2. **CLASSIFICATION ET RÉGRESSION**
3. **ALGORITHME**
4. **GÉNÉRALISATION ET ROBUSTESSE**
5. **RANDOM FORESTS**

→ **RÉSEAUX DE NEURONES**

SAFRAN
Snecma

ARBRE DE DÉCISION

Les méthodes de classifications basées sur des arbres de décision définissent des règles logiques en sélectionnant des variables et en découpant de manière dyadique et progressive l'espace de recherche.

- Un arbre de décision identifie des zones correspondant à des classes, et chaque zone peut se définir comme une conjonction d'inégalités, un hyperplan de l'espace des observations.

L'avantage d'une telle méthode est que ces règles de décisions sont très facilement interprétables par des techniciens opérant sur des machines.

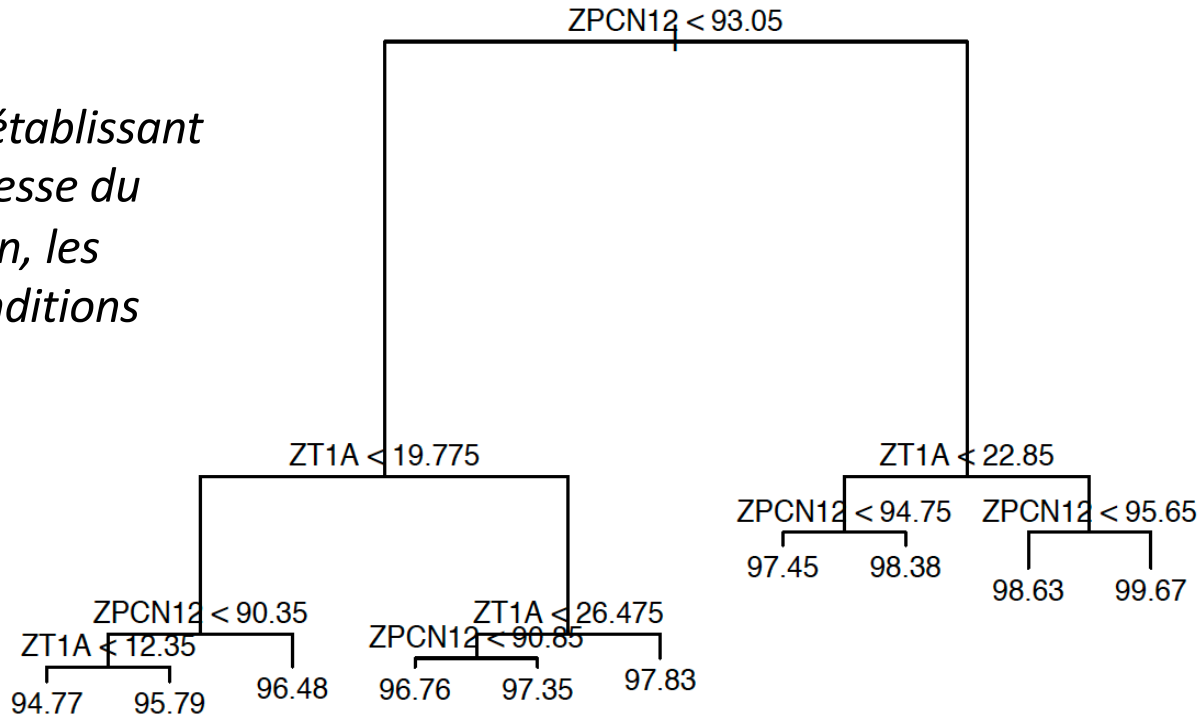
- On s'aperçoit cependant qu'un arbre de décision n'est pas nécessairement unique, que d'autres règles sont possibles, chacune ayant leurs avantages et leurs inconvénients. Par ailleurs, chaque règle indépendamment des autres n'est pas très robuste. Elle analyse bien les observations d'un échantillon donné, mais n'est pas nécessairement exploitable sur d'autres échantillons. C'est pourquoi on construit des solutions à base de mélanges aléatoires d'arbres de décision, les *random-forests* qui ont de très bonnes propriétés de généralisation.

L'arbre de décision est formé d'une succession de règles binaires séparant une partie de la population des observations en deux sous-populations.

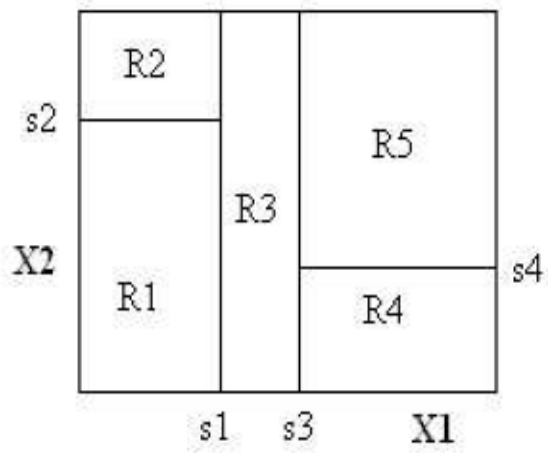
ARBRES DE DÉCISION
VOCABULAIRE

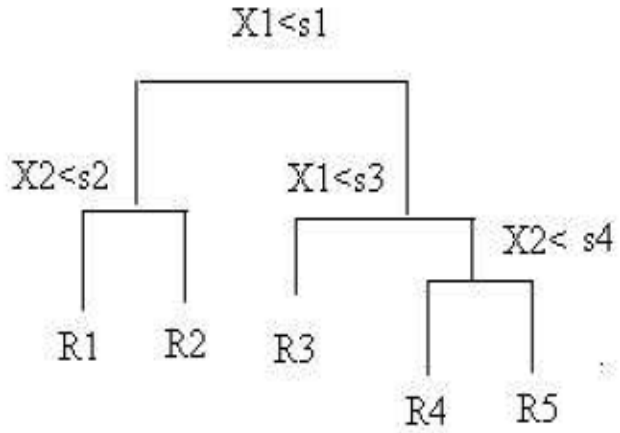
- Racine : le premier nœud de l'arbre.
- Feuille : un nœud terminal.
- Règle : l'expression entre deux nœuds.
- Régions : les ensembles définis par les feuilles formant la partition de l'espace des observations.

Un arbre de décision établissant la relation entre la vitesse du corps HP d'un turbofan, les commandes et les conditions extérieures.



ARBRES DE DÉCISION
PARTITION DE L'ESPACE DES ENTRÉES





Dans l'exemple ci-dessus un domaine de $\mathbb{R}^2$ est décomposé en régions par un arbre de décision.

CLASSIFICATION

On suppose que l'on dispose d'un échantillon

$$S = \{(x_i, y_i), x_i \in \mathbb{R}^p, y_i \in \{1 \dots K\}\}.$$

En général une région R_m ne contient pas que des informations relatives à la même classe on note

$$\hat{p}_m(k) = \frac{1}{N_m} \sum_{x_i \in R_m} \boldsymbol{\delta}_{y_i = k}$$

où $N_m = card(R_m)$.

Une observation $x_i \in R_m$ est classée en fonction de la représentation majoritaire dans la région R_m :

$$\hat{y}_i = \hat{k}_m(x) = \arg\max_{\mathrm{k}} p_m(k) \text{ si } x_i \in \mathrm{R}_m.$$

La fonction de classification peut s'écrire

$$\hat{y} = \sum_{m=1}^{M} \hat{k}_m(x)\, \boldsymbol{\delta}_{x \in R_m}$$

Les coefficients $\hat{k}_m$ s'obtiennent par le calcul de la population majoritaire de chaque région. Il faut donc optimiser le choix des régions R_m.

CRITÈRES QUALITATIFS

La qualité d'une région peut s'estimer facilement en étudiant sa pureté. On utilise généralement l'un des trois indices suivants :

- L'erreur de classification

$$I_E(m) = \frac{1}{N_m} \sum_{x_i \in R_m} \boldsymbol{\delta}_{y_i \neq \hat{k}_m} = 1 - \hat{p}_m(\hat{k}_m)$$

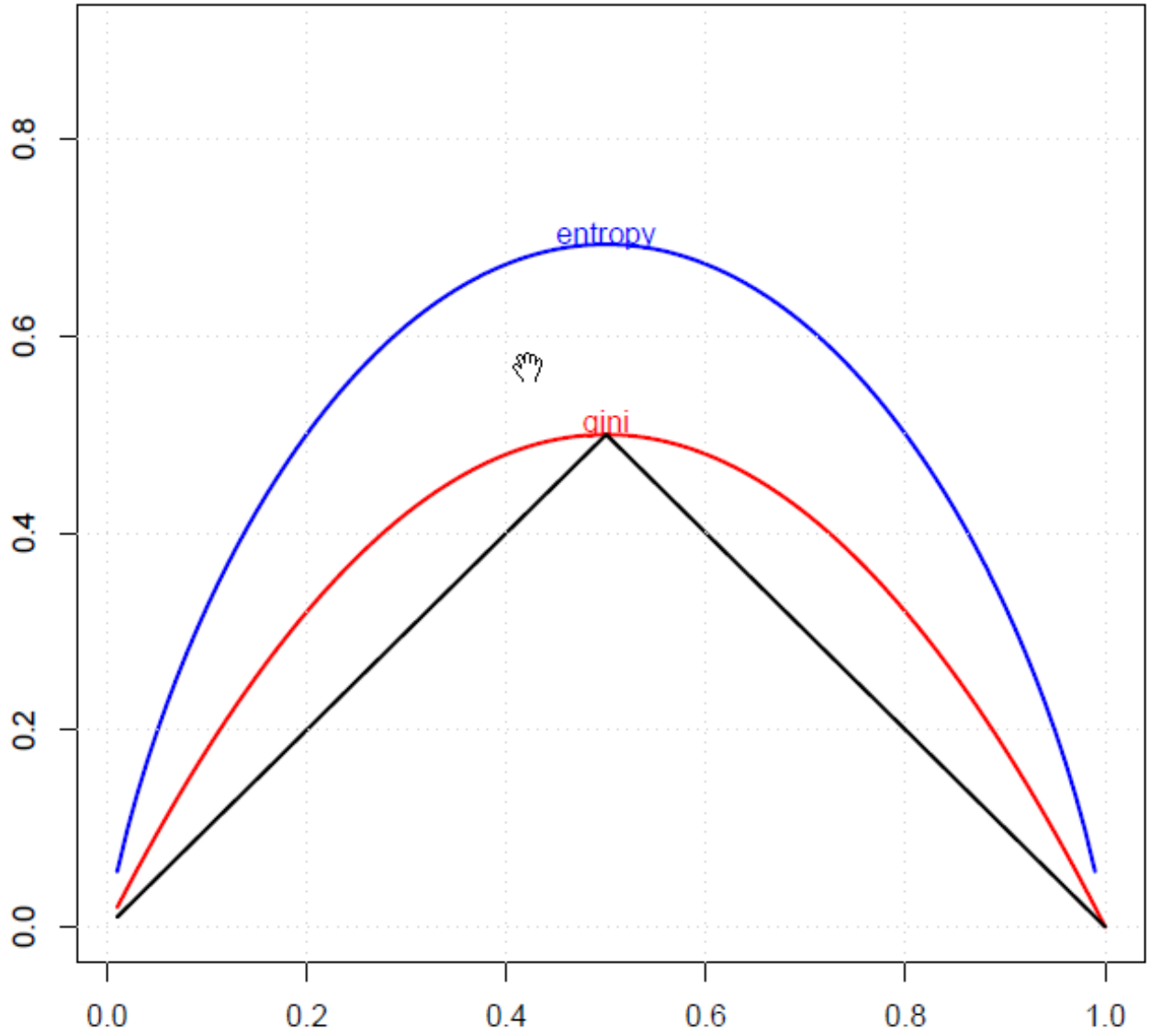
- L'indice de Gini

$$I_G(m) = \sum_{k \neq k'} \hat{p}_m(k)\hat{p}_m(k') = \sum_{k=1}^{K} \hat{p}_m(k)\big(1 - \hat{p}_m(k)\big)$$

- L'entropie

$$I_H(m) = -\sum_{k=1}^{K} \hat{p}_m(k) \log \hat{p}_m(k)$$

CRITÈRES QUALITATIFS
RAPPORTS ENTRE CRITÈRES



Ce graphe donne la valeur des trois indices précédents pour p variant de 0 à 1.

ARBRES DE RÉGRESSION

Dans le cas des régressions, la variable Y est quantitative. On prend donc pour chaque région R_m la moyenne comme estimation

$$\hat{y}_m(x) = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

Puis comme indice de qualité, l'écart quadratique

$$I_2(m) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{y}_m)^2$$

ALGORITHME D'APPRENTISSAGE

L'algorithme le plus fréquemment employé pour construire un arbre de décision opère de manière itérative.

- On suppose que l'on dispose d'une région R_m que l'on souhaite décomposer en deux parties $R_m = R_{m_1} \cup R_{m_2}$.
- Pour cela on doit choisir une variable X_l et un seuil s tels que

$$R_{m_1} = R_m \cap (X_l < s) \text{ et } R_{m_2} = R_m \cap (X_l \geq s)$$

  - Si la variable X_l est qualitative, au lieu de choisir un seuil on sélectionne une partition binaire (deux parties) de l'ensemble des classes de X_l.
    - Comme on dispose d'un échantillon fini d'observations et d'un nombre fini de variables, il n'y a qu'un nombre fini de cas à tester.
- On calcule ensuite le gain (ou *déviance*) : diminution du taux d'impureté.

$$G_I\ (m,l,s) = I(m) - \frac{N_1}{N_0} I(m_1) - \frac{N_2}{N_0} I(m_2)$$

- L'algorithme consiste à chaque étape à choisir la variable et le seuil maximisant ce gain.

$$(\hat{m}, \hat{l}, \hat{s}) = \arg\max_{m,l,s} G_I(m,l,s)$$

ROBUSTESSE

Pour améliorer la robustesse de l'arbre de décision, on doit limiter le nombre de feuilles sinon il ne généralise plus. Tel que l'algorithme est défini, il va construire un arbre ou chaque feuille finit par ne contenir presque plus qu'une seule observation. On doit donc élaguer l'arbre (ou l'arrêter avant la fin).

La première idée est de fixer un seuil s_{min} sur le gain en dessous duquel on décide que le gain n'est pas suffisant pour continuer.

- On arrête dès que

$$G_I\ (\hat{m}, \hat{l}, \hat{s}) < s_{min}$$

Une autre méthode consiste à rajouter un compensateur qui dépend du nombre de feuilles à l'indicateur global.

- Si T est l'arbre formé de M régions, on calcule

$$I(T) = \sum_{m=1}^{M} N_m\, I(m) + \lambda M$$

où $\lambda \in \mathbb{R}^+$.

- On élague les branches de l'arbre pour minimiser $I(T)$.

RÉSEAUX DE NEURONES

1. **RÉSEAUX**
2. **MÉMOIRE ASSOCIATIVE**
3. **EQUATIONS DE WIENER-HOPF**
4. **PERCEPTRONS MULTICOUCHES**
5. **RÉSEAUX RÉCURRENTS**

→ **MACHINES DE BOLTZMANN**

RÉSEAUX DE NEURONES

La notion de réseaux de neurones vient des tentatives d'approches du modèle neuronal biologique.

Le neurone de McCulloch et Pitts est un simple potentiel linéaire que l'on sature par une fonction d'activation non-linéaire, en général bornée.

$$v_i = \sum_{j=1}^{p} w_{i,j}\, x_j - \theta_i$$

avec $w_{i,j} \in \mathbb{R}$ et $\theta_i \in \mathbb{R}$.

- Pour simplifier les calculs on peut aussi considérer que l'unité $x_1 = 1$ fait partie des entrées et se dispenser de traiter le cas particulier du seuil θ_i.

$$y_i = \psi_T(v_i)$$

- En général on prend la fonction sigmoïde comme activation. *On verra plus loin que ce choix n'est pas anodin.*

$$\psi_T(v) = \frac{1}{1+e^{-\frac{v}{T}}} \qquad \text{avec} \qquad T \in \mathbb{R}^+$$

RÉSEAUX DE NEURONES STOCHASTIQUES

L'activation peut aussi être choisie stochastique, c'est le cas des modèles d'Ising ou des machines de Boltzmann binaires par exemple.

$$y_i \in \{0,1\} \quad \text{et} \quad P(Y_i = 1) = \psi_T(v_i)$$

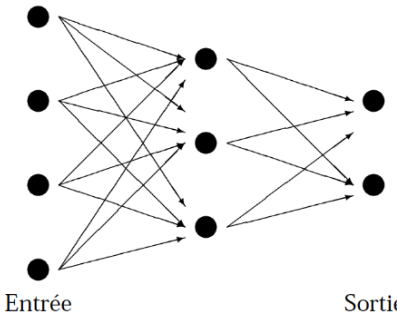
La mise à jour se fait alors par tirage aléatoire.

Le réel positif T est appelé température en thermodynamique (pour retrouver les $^\circ K$ de Boltzmann, il faudra diviser par la constante de Boltzmann k).

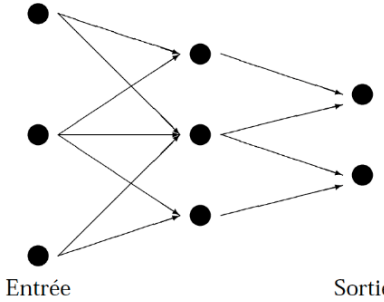
RÉSEAUX DE NEURONES
ARCHITECTURE

Les réseaux de neurones peuvent être globalement connectés, c'est le cas des réservoirs et de certaines machines de Boltzmann, où chaque cellule peut être connectée à toutes les autres.

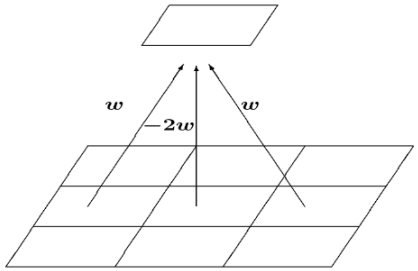
- Mais en général, l'architecture du réseau est construite pour être adaptée au problème.



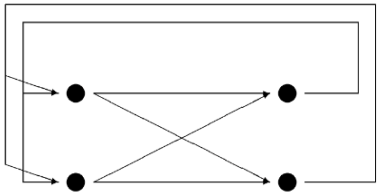
Réseau globalement connecté par couches
(extreme learning machine p. ex.)



Réseau localement connecté par couches
(deep network p. ex.)



Des invariances spatiales
(CNN – convolutive neural network)

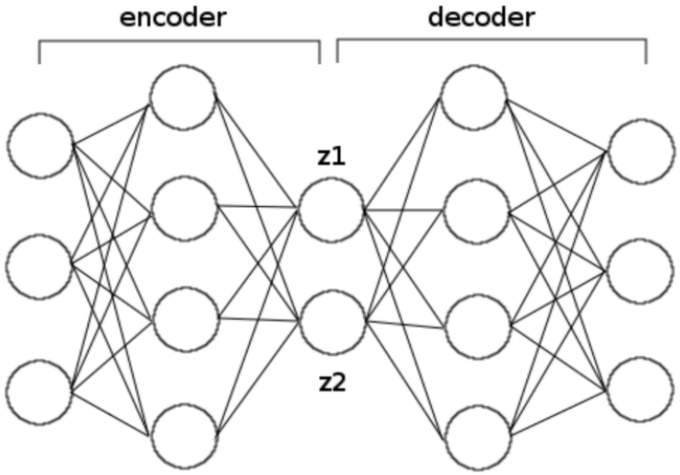


Réseau récurrent
(ESN – Echo state network)

RÉSEAUX DE NEURONES AUTO-ENCODEURS

Certains réseaux de neurones réalisent des tâches d'estimation ou de classification, ils sont donc contrôlés par une caractéristique de sortie. Mais il est aussi possible de concevoir un mécanisme de réduction de dimension auto-adaptatif à l'aide d'un réseau de neurone. C'est ce que font les auto-encodeurs.

- Un auto-encodeur est un réseau multicouches dont la sortie est égale à l'entrée mais pour lesquels nous nous intéressons à une couche intermédiaire.



Réseau auto-encodeur : la sortie va être apprise pour ressembler à l'entrée, donc si l'information se déplace de gauche à droite (perceptron) la couche centrale donne une projection non linéaire des entrées en dimension réduite.

MÉMOIRE ASSOCIATIVE

Il s'agit d'une simple matrice de corrélation, mais elle réalise bien une fonction d'apprentissage.

- Soit un échantillon

$$S = \{(x(n), y(n)), n = 1 \dots N, x(n) \in \mathbb{R}^p, y(n) \in \mathbb{R}^q\}$$

- On cherche à associer chaque sortie $y(n)$ à son entrée respective $x(n)$ en utilisant le modèle linéaire :

$$y = Wx + \epsilon \qquad W = (w_{i,j}), i = 1 \dots q, j = 1 \dots p$$

Il s'agit bien d'une régression linéaire qu'on aimerait rendre auto-adaptative.

On pose

$$W = Y'X = \sum_{n=1}^{N} y(n)x'(n) \quad \text{(matrice } q \times p\text{)}.$$

- On rappelle que les matrices d'échantillons placent les observations en lignes et les caractéristiques (variables) en colonnes :

$$X = (x(1) \dots x(n))' \text{ et } Y = (y(1) \dots y(n))'$$

MÉMOIRE ASSOCIATIVE (2)

Alors

$$Wx(n) = y(n)(x'(n)x(n)) + \sum_{m \neq n} y(m)\big(x'(m)x(n)\big)$$

- Si les vecteurs d'entrée sont normés $x'(n)x(n) = 1$ (mettons qu'on ne s'intéresse qu'aux directions des vecteurs), alors

$$Wx(n) = y(n) + \epsilon(n)$$

avec

$$\epsilon(n) = \sum_{m \neq n} y(m) \cos\big(x(m), x(n)\big)$$

MÉMOIRE ASSOCIATIVE APPRENTISSAGE

Si les vecteurs d'entrée étaient orthonormés, on obtiendrait une correspondance parfaite, et la mise à jour de la matrice W peut se faire simplement par incréments successifs :

$$W(n+1) = W(n) + y(n)x'(n)$$

Comme les vecteurs de l'échantillon ne sont pas orthonormés dans le cas général on utilise une transformation dans une base orthonormée par le procédé de Gramm-Schmidt.

$$z(n) = x(n) - \sum_{m=1}^{n-1} \left(\frac{z'(m)x(n)}{z'(m)z(m)} \right) z(m)$$

MÉMOIRE ASSOCIATIVE CORRECTION D'ERREUR

Reprenons le neurone de McCulloch et Pitts (on suppose l'activation linéaire pour simplifier).

$$v_i(n) = \sum_{j=1}^{p} w_{i,j}\, x_j(n) \quad \text{avec} \quad \hat{y}_i(n) = v_i(n)$$

La réponse à chaque excitation $x(n)$ étant donnée par $y(n)$, il est possible de calculer l'erreur faite par le neurone :

$$\lambda(n) = \|y(n) - \hat{y}(n)\|^2 = \sum_{i=1}^{q} (y_i(n) - \hat{y}_i(n))^2$$

Si après avoir présenté l'observation n on décide de mettre à jour les poids du réseau alors on suit une descente de gradient

$$\frac{1}{2}\frac{\partial \lambda(n)}{\partial w_{i,j}} = -\big(y_i(n) - \hat{y}_i(n)\big)x_j(n)$$

$$\text{car} \quad \frac{\partial \hat{y}_i(n)}{\partial w_{i,j}} = x_j(n) \quad \text{et} \quad \frac{\partial \hat{y}_{i'}(n)}{\partial w_{i,j}} = 0 \quad \text{si} \quad i' \neq i.$$

MÉMOIRE ASSOCIATIVE
DESCENTE DE GRADIENT

On choisit donc un pas d'apprentissage η et on pose

$$w_{i,j}(n+1) = w_{i,j}(n) + \eta\big(y_i(n) - \hat{y}_i(n)\big)x_j(n)$$

Si l'échantillon provient d'un flux aléatoire infini, on met à jours les poids après chaque observation n. Sinon, on peut choisir de parcourir séquentiellement l'échantillon en boucle ou bien tirer des observations aléatoirement dans S.

Remarque :

$$\text{Si } \; 0 < \eta < \frac{2}{\mu_1(H)}$$

où μ_1 est la plus grande valeur propre du Hessien

$$H = \left(\frac{\partial^2 \lambda}{\partial w_{i,j} \partial w_{i,j}}\right)$$

alors l'apprentissage converge.

MÉMOIRE ASSOCIATIVE
RENFORCEMENT ET OUBLI

On aurait pu aussi traiter les données par batch et ne faire de mise à jour des poids qu'à la fin de chaque batch. Les calculs sont similaires.

Notons que l'expression précédente peut aussi s'écrire

$$w_{i,j}(n+1) = w_{i,j}(n) + \underbrace{\eta y_i(n) x_j(n)}_{\text{On retrouve la corrélation de la matrice associative.}} - \underbrace{\eta \sum_{j'=1}^{p} w_{i,j'}(n) x_{j'}(n) x_j(n)}_{\text{Le dernier terme est souvent appelé terme d'oubli.}}$$

EQUATIONS DE WIENER-HOPF

On retrouve dans des termes d'apprentissage et de réseaux de neurones l'expression de la régression linéaire.

- On se place dans le cadre d'une régression comme plus haut, mais on n'observe qu'une sortie.

$$\hat{y} = \sum_{j=1}^{p} w_j \, x_j$$

- On suppose que le neurone est soumis à une excitation aléatoire suivant la variable d'entrée X correspondant à une sortie souhaitée Y, l'erreur moyenne peut donc s'écrire

$$\begin{aligned}
\bar{\lambda} &= E\left(Y - \hat{Y}\right)^2 \\
&= E(Y^2) - 2E\left[\sum_{i=1}^{p} w_i X_i Y\right] + E\left[\sum_{i=1}^{p}\sum_{j=1}^{p} w_i w_j X_i X_j\right] \\
&= E(Y^2) - 2\sum_{i=1}^{p} w_i r_i^{xy} + \sum_{i=1}^{p}\sum_{j=1}^{p} w_i w_j r_{i,j}^{x}
\end{aligned}$$

EQUATIONS DE WIENER-HOPF
CALCUL DU GRADIENT

- Avec $R^x = (r_{i,j}^x)$ la matrice de covariance de X et $r^{xy} = \left(r_i^{xy}\right)$ le vecteur de corrélation entre l'entrée et la sortie.

$$\frac{1}{2}\frac{\partial \bar{\lambda}}{\partial w_j} = -r_j^{xy} + \sum_{i=1}^{p} w_i r_{i,j}^x \quad \text{car} \quad r_{i,j}^x = r_{j,i}^x$$

- Les équations de Wiener-Hopf expriment les conditions d'un minimum d'erreur obtenu pour $\frac{\partial \bar{\lambda}}{\partial w} = 0$:

$$(WH) \qquad \sum_{i=1}^{p} w_i r_{i,j}^x = r_j^{xy} \qquad \forall j \in \{1 \dots p\}$$

La solution est bien unique si la matrice de corrélation est inversible.

Pour converger vers ce minimum, il est possible d'utiliser un algorithme de descente de gradient,

$$w_j(n+1) = w_j(n) + \eta\left(r_j^{xy} - \sum_{i=1}^{p} w_i(n) r_{i,j}^x\right)$$

mais il faut tout de même estimer la matrice de corrélations R^x et le vecteur R^{xy}.

EQUATIONS DE WIENER-HOFF
MOINDRES CARRÉS ITÉRATIFS

L'idée est de remplacer les corrélations précédentes par leurs estimations instantanées :

$$\begin{cases} \hat{r}_{i,j}^{x}(n) = x_i(n)x_j(n) \\ \hat{r}_{j}^{xy}(n) = x_j(n)y(n) \end{cases}$$

- Ce qui permet de retrouver la règle précédente

$$w_j(n+1) = w_j(n) + \eta\left(x_j(n)y(n) - \sum_{i=1}^{p} w_i(n)x_i(n)x_j(n)\right)$$
$$= w_j(n) + \eta\big(y(n) - \hat{y}(n)\big)\, x_j(n)$$

Vectoriellement on peut écrire

$$w(n+1) = (I - \eta x(n)x'(n))w(n) + \eta x(n)y(n)$$

- On recherche une condition de convergence en moyenne de $w(n)$:

$$E[w(n)] \underset{n\to\infty}{\longrightarrow} w$$

EQUATIONS DE WIENER-HOPF DIAGONALISATION

On suppose que les observations arrivent suffisamment aléatoirement pour que $w(n)$ et $x(n)$ soient indépendants.

A priori, cette hypothèse est vraie si les observations sont générées aléatoirement par l'environnement, mais elle est fausse dès que l'on itère sur un échantillon de données fini.

Dans ce cas

$$E[x(n)x'(n)w(n)\,] = R^x\,E[w(n)]$$

- En reportant ce résultat dans l'équation précédente on a

$$E[w(n+1)] = (I - \eta R^x)E[w(n)] + \eta r^{xy}$$

- Comme R^x est symétrique positive, on la diagonalise

$$P'R^xP = \Lambda = \mathrm{diag}(\lambda_1 \ldots \lambda_p) \quad \text{avec} \quad PP' = I$$
$$\lambda_i > \lambda_{i+1} \geq 0$$

- Si une limite w existe, alors elle vérifie les équations de Wiener-Hopf donc

$$R^x w = r^{xy}$$

EQUATIONS DE WIENER-HOPF
CONVERGENCE DE L'ALGORITHME

- En reportant dans l'équation

$$E[w(n+1)] = (PP' - \eta P\Lambda P')\, E[w(n)] + \eta P\Lambda P' w$$
$$= P\{(I - \eta\Lambda)P' E[w(n)] + \eta\Lambda P' w\}$$

Soit

$$P'(E[w(n+1)] - w) = (I - \eta\Lambda)P'(E[w(n)] - w)$$

- On a donc la suite $u(n) = P'(E[w(n)] - w)$ qui est géométrique de raison $(I - \eta\Lambda)$.

$$u(n) = (I - \eta\Lambda)^n u(0)$$

C'est une suite vectorielle dont chaque composante est géométrique (la matrice est diagonale) et telle que

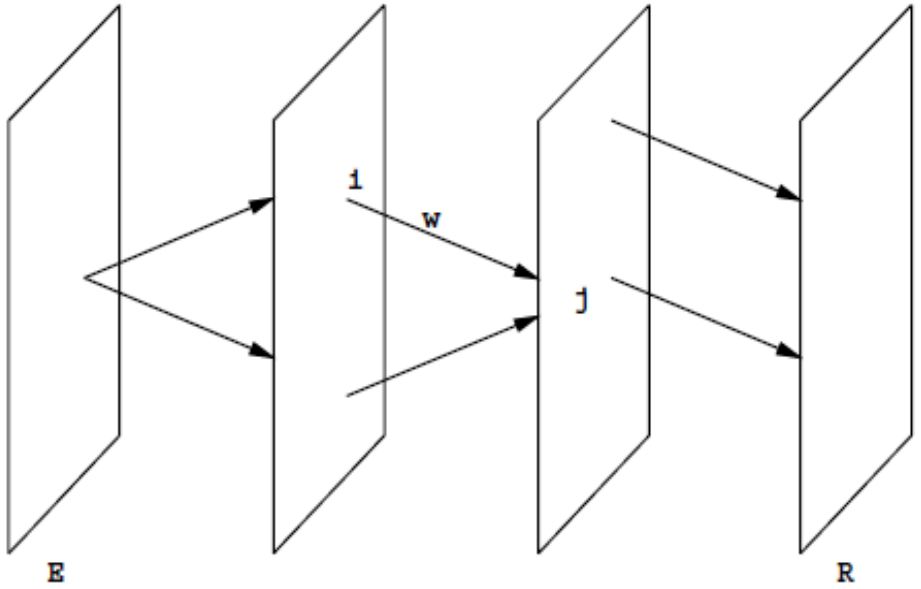
$$u_i(n) = (1 - \eta\lambda_i)^k\, u_i(0)$$

- Pour que la suite converge il faut donc que

$$-1 < 1 - \eta\lambda_i < 1 \;\; \forall i, \qquad \text{soit} \quad 0 < \eta < 2/\lambda_1$$

PERCEPTRON MULTICOUCHES

L'étape suivante naturelle est d'enchaîner les neurones les uns aux autres. Une architecture standard est le réseau multicouches (perceptron).



Un perceptron multicouches. L'information transite d'une couche à l'autre par propagation.

PERCEPTRON MULTICOUCHES
EQUATION DE PROPAGATION

$$x_j = \psi_T\left(v_j\right) \quad \text{avec} \quad v_j = \sum_{i \in Couche\ précédente} w_{j,i} x_i$$

et

$$\psi'_T(v) = \frac{1}{T}\psi_T(v)\left(1 - \psi_T(v)\right) = \frac{1}{T}x(1-x)$$

- Notations :

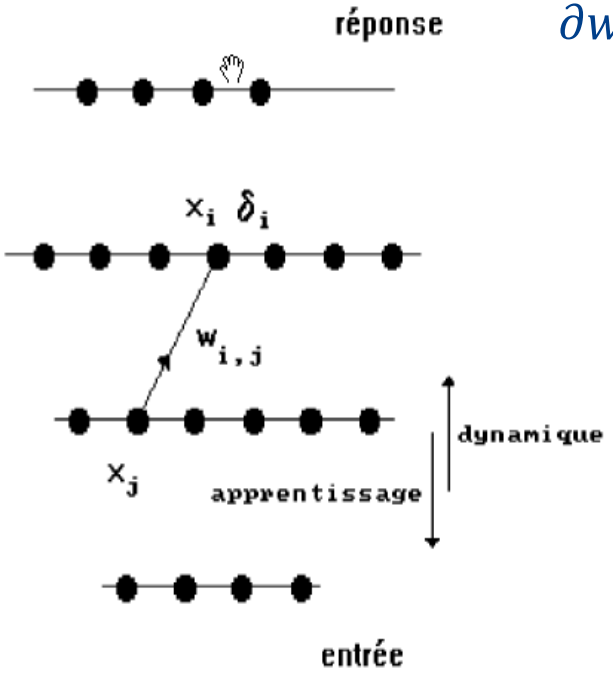
  Dans cette partie on note x les variables estimées par le réseau et toujours y la réponse attendue qui n'a besoin d'être connue que sur $\mathbb{R}$.

On va supposer que le réseau est entièrement connecté par couche. On note R la couche de réponse et E la couche d'entrée. L'erreur n'est calculée que sur les cellules de réponse.

$$\lambda(n) = \sum_{k \in R}\left(y_k(n) - x_k(n)\right)^2$$

PERCEPTRON MULTICOUCHES
RÉTRO-PROPAGATION (1)

Si on choisit un poids $w_{i,j}$ entre deux couches choisies au hasard.



$$\frac{\partial \lambda(n)}{\partial w_{j,i}} = \underbrace{\frac{\partial \lambda(n)}{\partial x_j(n)} \frac{\partial x_j(n)}{\partial v_j(n)}}_{-\delta_j(n)} \underbrace{\frac{\partial v_j(n)}{\partial w_{j,i}}}_{x_i(n)}$$

$$\frac{\partial \lambda(n)}{\partial w_{j,i}} = -\delta_j(n) x_i(n)$$

$$\delta_j(n) = -\frac{\partial \lambda(n)}{\partial x_j(n)} \frac{\partial x_j(n)}{\partial v_j(n)} = -\frac{\partial \lambda(n)}{\partial x_j(n)} \psi_T'\left(v_j(n)\right)$$

$$= -\frac{\partial \lambda(n)}{\partial x_j(n)} \frac{1}{T} x_j(n)\left(1 - x_j(n)\right)$$

Si $j = k \in R$, le calcul est assez simple :

$$\delta_k(n) = 2\left(y_k(n) - x_k(n)\right) \frac{1}{T} x_k(n)\left(1 - x_k(n)\right)$$

PERCEPTRON MULTICOUCHES RÉTROPROPAGATION (2)

Si $j \notin R$, on va exprimer δ_j en fonction des couches suivantes :

$$\frac{\partial \lambda(n)}{\partial x_j(n)} = \sum_{k \in Couche\ suivante} \left(\frac{\partial \lambda(n)}{\partial x_k(n)} \frac{\partial x_k(n)}{\partial v_k(n)} \right) \frac{\partial v_k(n)}{\partial x_j(n)}$$

$$= - \sum_{k \in Couche\ suivante} \delta_k(n) w_{k,j}$$

- On peut donc exprimer les valeurs de $\delta_j(n)$ par rétro-propagation de droite vers la gauche (sortie vers entrée) :

$$\delta_j(n) = \left(\sum_{k \in Couche\ suivante} \delta_k(n) w_{k,j} \right) \frac{1}{T} x_j(n) \left(1 - x_j(n) \right)$$

- Finalement la mise à jour des poids est choisie comme

$$w_{j,i}(n+1) = w_{j,i}(n) + \eta \delta_j(n) x_i(n)$$

RÉSEAUX RÉCURRENTS
CLIQUES

On définit un ensemble de cliques $\mathcal{K} = \{K \subset \{1 \dots p\}\}$ comme un ensemble de parties des sites.

Notations :

- x_K le vecteur des observations sur la clique K
- $J_K(x_K) = J_K(x)$ une fonction de clique réelle.
- $w_{i,K}$ une matrice de poids reliant les cliques aux sites.
- $K^* = \{i \mid w_{i,K} \neq 0\}$ la clique duale de K correspondant aux sites connectés à K.

On définit le potentiel

$$v_i\big(x(n)\big) = \sum_{K \in \mathcal{K} \text{ et } i \in K^*} w_{i,K} J_K\big(x(n)\big)$$

Et l'activation (on la suppose identique pour toutes les cellules, mais elle pourrait bien être spécifique à chaque site).

$$x_i(n+1) = \psi_T(v_i\big(x(n)\big)$$

RÉSEAUX RÉCURRENTS
ERREUR ET GRADIENT

Si le processus converge, alors la limite x vérifie

$$\forall i, \qquad x_i = \psi_T\left(\sum_{K;\, i\in K^*} w_{i,K} J_K(x)\right)$$

On va minimiser l'erreur commise par le réseau sur un sous-ensemble réponse R des sites.

$$\lambda(x) = \sum_{k\in R} (y_k - x_k)^2$$

Alors

$$\frac{\partial \lambda(x)}{\partial w_{i,K}} = -2 \sum_{k\in R} (y_k - x_k) \frac{\partial x_k}{\partial w_{i,K}}$$

RÉSEAUX RÉCURRENTS APPRENTISSAGE (1)

- On différencie l'équation limite

$$\frac{\partial x_k}{\partial w_{i,K}} = \psi_T'\left(v_k(x)\right)\left\{J_K(x)\chi_{i=k} + \sum_{K';k\in K'^*} w_{k,K'}\frac{\partial J_{K'}(x)}{\partial w_{i,K}}\right\}$$

$$= \psi_T'\left(v_k(x)\right)\left\{J_K(x)\chi_{i=k} + \sum_{K';k\in K'^*} w_{k,K'}\sum_{j\in K'}\frac{\partial J_{K'}(x)}{\partial x_j}\frac{\partial x_j}{\partial w_{i,K}}\right\}$$

$$= \psi_T'\left(v_k(x)\right)\left\{J_K(x)\chi_{i=k} + \sum_{j}\sum_{\substack{K';\\ k\in K'^*,\ j\in K'}} w_{k,K'}\frac{\partial J_{K'}(x)}{\partial x_j}\frac{\partial x_j}{\partial w_{i,K}}\right\}$$

- En définissant la matrice A et le vecteur $b(i,K)$ dépendant de x par

$$\begin{cases} a_{k,j} = \psi_T'\left(v_k(x)\right) \sum_{K';i\in K'^*,j\in K'} w_{k,K'}\frac{\partial J_{K'}(x)}{\partial x_j} \\ b_k(i,K) = \psi_T'\left(v_k(x)\right) J_k(x)\ \chi_{i=k} \end{cases}$$

RÉSEAUX RÉCURRENTS APPRENTISSAGE (2)

- On peut écrire matriciellement

$$(I - A)\ \frac{\partial x}{\partial w_{i,K}} = b(i,K)$$

- Pour T suffisamment grand, comme $\left|\psi'_T\right| \leq \frac{1}{4T}$, on peut supposer que la norme de A est inférieure à 1. Donc

$$(I - A)^{-1} = I + A + A^2 + \cdots$$

et

$$\frac{\partial x}{\partial w_{i,K}} = \sum_{n=0}^{+\infty} A^n\ b(i,K)$$

RÉSEAUX RÉCURRENTS
RÉTROPROPAGATION

Finalement en posant $\begin{cases} e_k = -2(y_k - x_k) & si\ k \in R \\ e_j = 0 & sinon \end{cases}$

$$\frac{\partial \lambda(x)}{\partial w_{i,K}} = e' \frac{\partial x}{\partial w_{i,K}} = \left(\sum_{n=0}^{+\infty} e' A^n \right) b(i,K)$$

- On définit le processus $\delta(m)$ par

$$\delta(0) = 0, \qquad et \qquad \delta(m+1) = e + A'\delta(m)$$

$$\text{tel que} \quad \delta' = \lim_{m \to +\infty} \delta'(m) = \sum_{n=0}^{+\infty} e' A^n$$

Ainsi

$$\frac{\partial \lambda(x)}{\partial w_{i,K}} = \delta' b(i,K) = \delta_i \, \psi_T'\left(v_i(x)\right) J_K(x)$$

RÉSEAUX RÉCURRENTS
RÉTROPROPAGATION (2)

Si on détaille la dynamique de $\delta(m)$ on remarque qu'elle suit la structure de voisinage du réseau, mais en évoluant dans le sens inverse et avec une activation linéaire.

$$\delta_j(m+1) = e_j + \sum_{K;j\in K} u_{K,j}\, L_K\big(\delta_{K^*}(m)\big)$$

Avec

$$\begin{cases} u_{K,j} = \dfrac{\partial J_K(x)}{\partial x_j} \\ L_K(\delta) = \displaystyle\sum_{i\in K^*} w_{i,K}\, \psi_T'\big(v_i(x)\big)\delta_i \end{cases}$$

A SUIVRE

APPRENTISSAGE