EISTI
Ecole Internationale des Sciences du Traitement de l'Information

MACHINE LEARNING

Réseaux de neurones :

- Neurone formel et réseau de neurones
- Apprentissage des réseaux de neurones
- Comment palier au surajustement?

LE NEURONE FORMEL

Notons $x_1,\ldots, x_p$ les variables explicatives. Un neurone se définit en deux étapes :

Une combinaison linéaire des variables :

$$\Sigma = w_0 + \sum_{i=1}^{n} w_i x_i$$

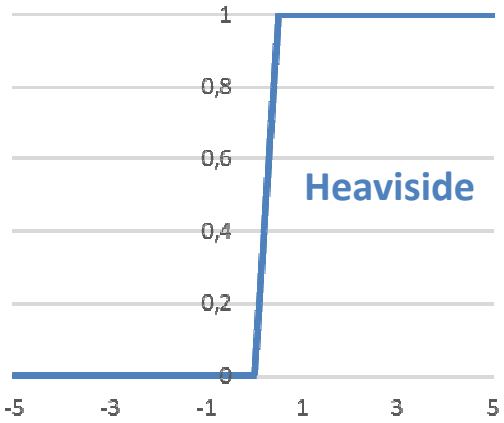
où $w_1,\ldots,w_p$ sont les poids. La constante w_0, appelé le biais, est facultative.

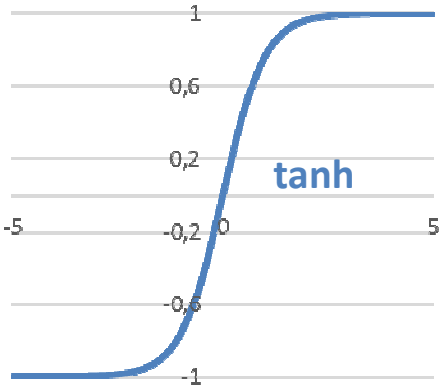
Une fonction d'activation ou de transfert

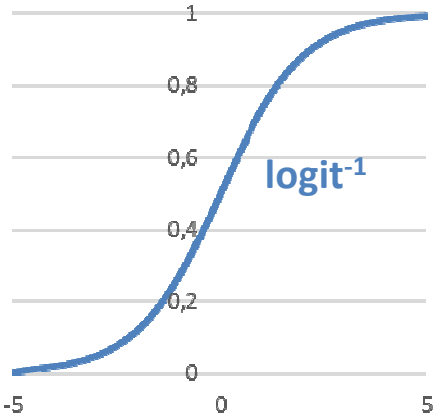
$$f(\Sigma) = f(\sum_{i=0}^{n} w_i x_i)$$

- Fonction de Heaviside : H(x)=1 si $x\geq 0$ et H(x)=0 si $x<0$ (perceptron)
- Fonction tangente hyperbolique : f(x)=tanh(x)$\in$ [-1;1]
- Fonction logistique inverse (ou sigmoide) : f(x)=1/(1+exp(-x))$\in$ [0;1]
- Fonction identité, radiale, ReLU,…

cf. annexe pour comportement des fonctions sigmoïdales





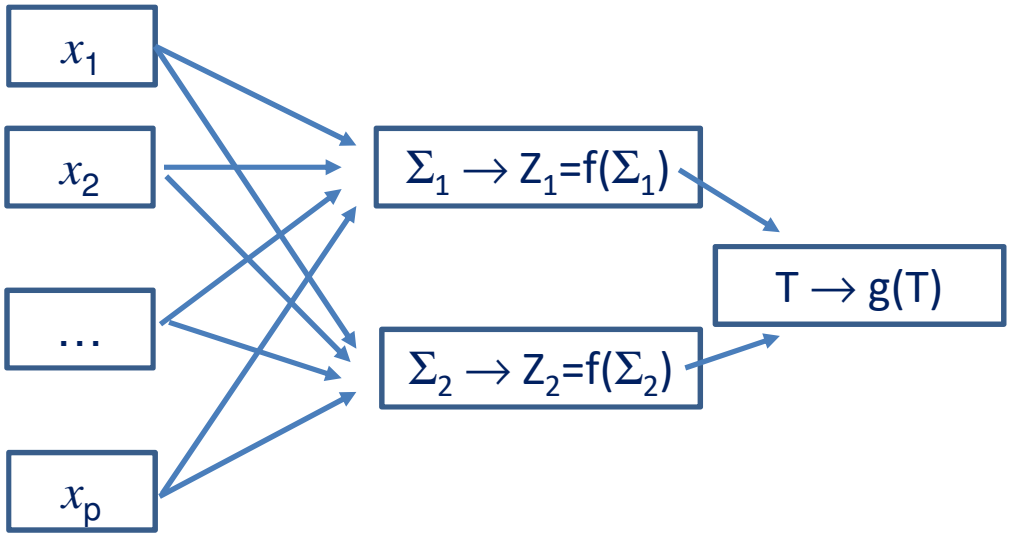


UN RÉSEAU DE NEURONES

Un réseau de neurones est une combinaison linéaire de N_c neurones formels à laquelle on applique une fonction d'activation,

$$g\left(\sum_{j=1}^{N_C} w'_j f(\Sigma_j)\right) = g\left(\underbrace{\sum_{j=1}^{N_C} w'_j \underbrace{f\left(\sum_{i=1}^{p} w_{ij} x_i\right)}_{=Z_j}}_{=T}\right)$$

Ce qui se représente sous la forme du graphe



On parle de perceptron multicouches.

Sur cet exemple, il s'agit ici de la forme la plus simple de perceptron avec une seule couche cachée.

On peut empiler les couches cachées.

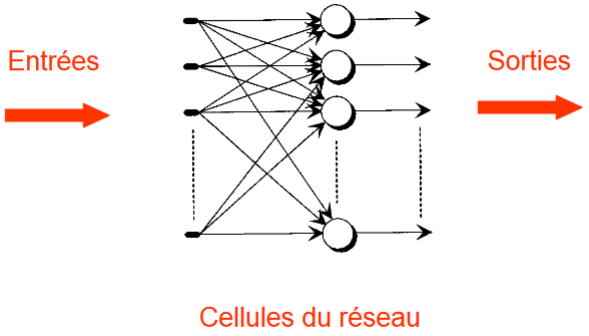
Dans le perceptron les neurones sont connectés avec les neurones de la couche d'avant ou après mais il n'y a pas de lien au-delà.

Il existe d'autres formes très performantes de réseaux $\Rightarrow$

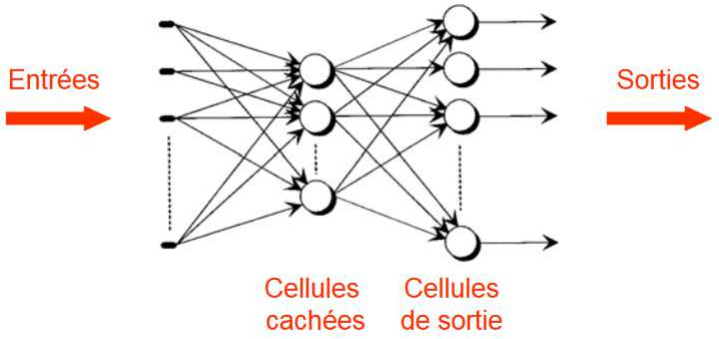
DIFFÉRENTES ARCHITECTURES DE RÉSEAUX

Cadre du cours

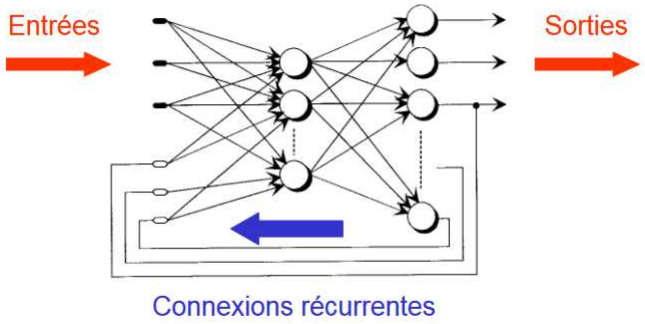
Mono-couche



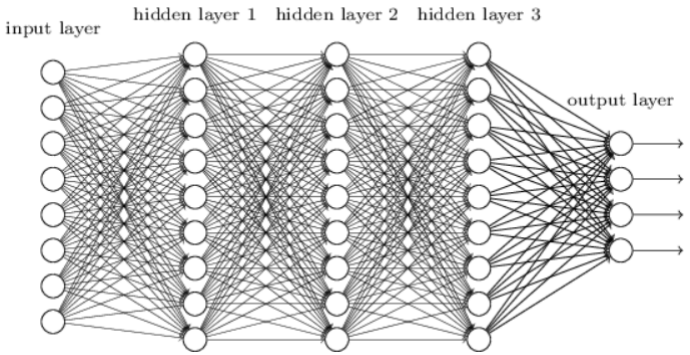
Multi-couche



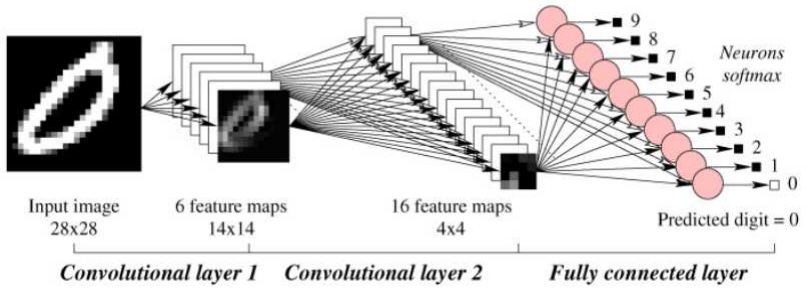
Récurrents



Deep learning



Réseau de convolution

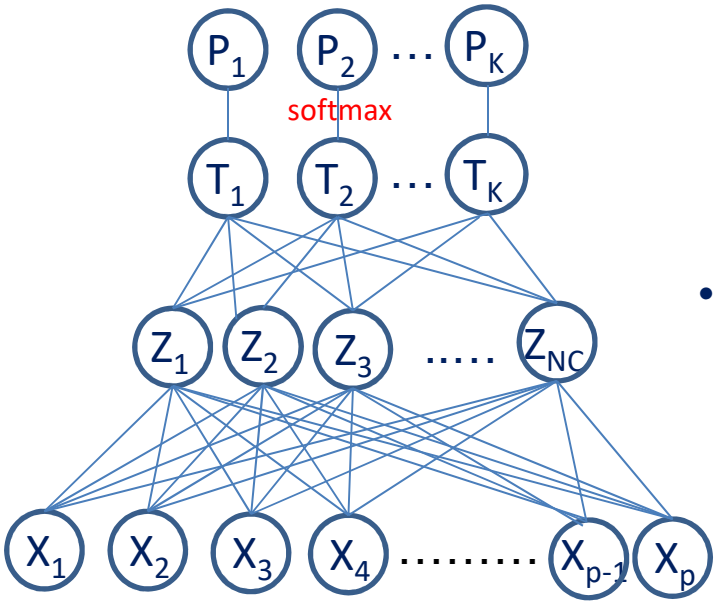


http://www.isir.upmc.fr/UserFiles/File/LPrevost/connex%201%20&%202.pdf
http://neuralnetworksanddeeplearning.com/chap5.html
http://www.sciencedirect.com/science/article/pii/S2212764X17300614

APPLICATION À LA CLASSIFICATION

Reprenons le cadre probabiliste introduit pour les méthodes bayésiennes. Notons $\mathbf{X}=(X_1,\ldots,X_p)$ le vecteur *aléatoire* constitué des variable *aléatoires* explicatives, Y la variable *aléatoire* cible et C_k, $k\in\{0,\ldots,K\}$ ses modalités. Pour un individu $\boldsymbol{x}$, l'objectif est de déterminer la classe C_k qui maximise la probabilité *a posteriori*

$$P\left(Y\in C_k \,|\, X(\boldsymbol{x})\right)=p_k(\boldsymbol{x})$$



- Dans le cas d'une classification binaire, la sortie du réseau de neurones représente la probabilité *a posteriori* de la classe positive, $P(Y=1\,|\,\boldsymbol{x})=p(\boldsymbol{x})$ (*). La fonction d'activation *g* est

$$g(T)=\frac{1}{1+e^{-T}}=p(\boldsymbol{x})$$

- Dans le cas d'une classification multi-classes (K>2), la couche de sortie comprend autant d'unités qu'il y a de modalités, chacune représentant la probabilité *a posteriori* (**). Afin que la somme des probabilités égale 1, la fonction d'activation *g* de la couche de sortie est la fonction softmax,

$$g(T_k)=\frac{e^{T_k}}{\sum_{i=1}^{K} e^{T_i}}=p_k(\boldsymbol{x})$$

(*) On note que l'objectif est le même que pour la régression logistique : modéliser $p(\boldsymbol{x})$.

(**) En fait, on crée une variable aléatoire Y_k qui vaut 1 si $Y\in C_k$ et 0 sinon et on modélise la probabilité a posteriori de la classe positive $P(Y_k=1\,|\,\boldsymbol{x})$

PROPRIÉTÉS DES RÉSEAUX DE NEURONES

Les réseaux de neurones sont bien connus pour leur capacité à modéliser des fonctions de formes quelconques grâce aux fonctions d'activation sigmoïdales mais aussi grâce aux deux propriétés suivantes.

- La propriété d'*approximation universelle* (*) garantit que toute fonction suffisamment régulière peut être approchée (au sens moindres carrés) par un réseau de neurones à **une** couche cachée (possédant un nombre fini de neurones ayant la **même** fonction d'activation), et ce pour une précision arbitraire et dans un domaine fini de l'espace de ses variables.

Cette propriété justifie l'utilisation de l'architecture précédente à une seule couche cachée

- Les réseaux de neurones présentent aussi l'avantage d'être *parcimonieux* (**). Pour une précision donnée, le nombre de paramètres du réseau à estimer est proportionnel au nombre de variables d'entrée. Donc, quand le nombre de variables est grand, il est plus avantageux d'utiliser un réseau de neurones qu'un modèle polynomial par exemple.

(*) démonstration 1989 Cybenko/Funahashi
(**) démonstration 1994 Hornik et al.

AJUSTEMENT DU RÉSEAUX DE NEURONES

LA FONCTION DE COÛT

Une fois la structure du réseau fixée, il faut déterminer les poids w tels que la sortie du réseau soit la plus « proche » possible des observations. En classification, il existe deux fonctions de coût. Notons $\boldsymbol{x}_i=(x_{1i},\ldots,x_{pi})$ un exemple de la base d'apprentissage de taille n.

Dans le cas binaire on note y_i la valeur de la variable cible et $p(\boldsymbol{x}_i,\mathbf{w})$ la sortie du réseau de neurones de poids $\mathbf{w}$.

Dans le cas multi-classes on note $y_{ik}=1$ si $y_i \in C_k$ et 0 sinon, et $p_k(\boldsymbol{x}_i,\mathbf{w})$ la sortie du réseau correspondant à la classe C_k.

L'erreur quadratique moyenne

$$E(\boldsymbol{w})=\frac{1}{2}\sum_{i=1}^{n}(y_i-p(\boldsymbol{x}_i,\boldsymbol{w}))^2$$

Cas binaire

$$E(\boldsymbol{w})=\frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{K}(y_{ik}-p_k(\boldsymbol{x}_i,\boldsymbol{w}))^2$$

Cas multi-classes

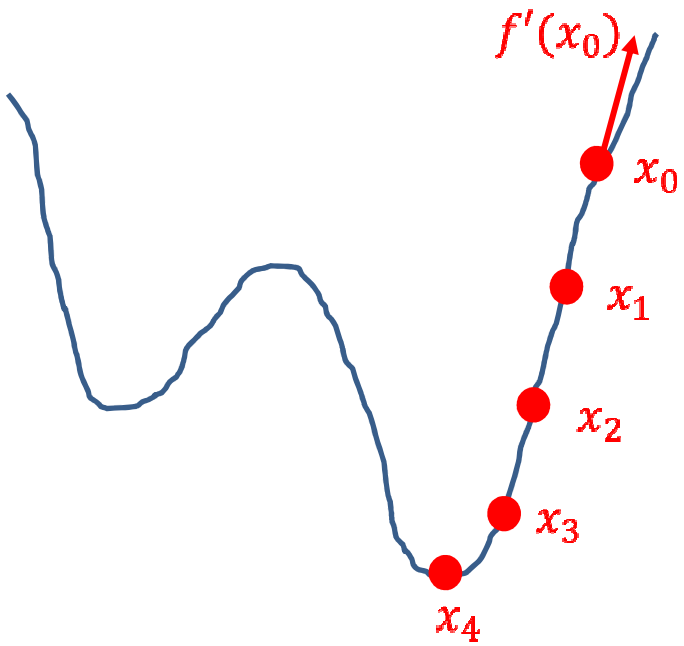
L'entropie croisée

$$E(\boldsymbol{w})=-\sum_{i=1}^{n}y_i\ln p(\boldsymbol{x}_i,\boldsymbol{w})+(1-y_i)\ln(1-p(\boldsymbol{x}_i,\boldsymbol{w}))$$

il s'agit du log de la vraisemblance comme pour la régression logistique

$$E(\boldsymbol{w})=\sum_{i=1}^{n}\sum_{k=1}^{K}y_{ik}\ln p_k(\boldsymbol{x}_i,\boldsymbol{w})$$

DESCENTE DU GRADIENT



Algorithme pour minimiser une fonction

- Choisir un point au hasard x_0
- Répéter jusqu'à convergence :
  - Calculer la pente (gradient) $f'(x_0)$
  - Avancer dans le sens opposé à la pente

$$x_1 = x_0 - \alpha * f'(x_0)$$

où α est le taux d'apprentissage.

La convergence de l'algorithme dépend :

- du taux d'apprentissage α
- de l'initialisation aléatoire x_0

https://www.charlesbordet.com/fr/gradient-descent/#

CONVERGENCE DE L'ALGORITHME

Le taux d'apprentissage α

- α trop petit : convergence lente
- α trop grand : les poids oscillent et ne se stabilisent pas
- Prendre α grand au début pour convergence rapide puis le faire diminuer progressivement

L'initialisation des poids

- Les poids de la couche cachée sont uniformément distribués autour de 0
- Les poids de la couche de sortie sont plus grand [-1,1]
- Penser à centrer et réduire les variables sinon cela n'a aucun sens
- Les poids doivent être différents d'un neurone à l'autre sinon ils feront tous la même chose

Problème des valeurs du gradient

Pour certaines fonction, le gradient risque de devenir

- très grand (lors d'une descente abrupte) donc engendrer une instabilité dans l'algorithme à cause d'un taux d'apprentissage trop élevé
- très petit (sur un plateau) donc ralentir (voire stopper) la convergence. Ce problème apparait souvent en *deep learning* et trouve une résolution avec la fonction d'activation ReLU,

$$f(x) = x^+ = \max(0, x)$$

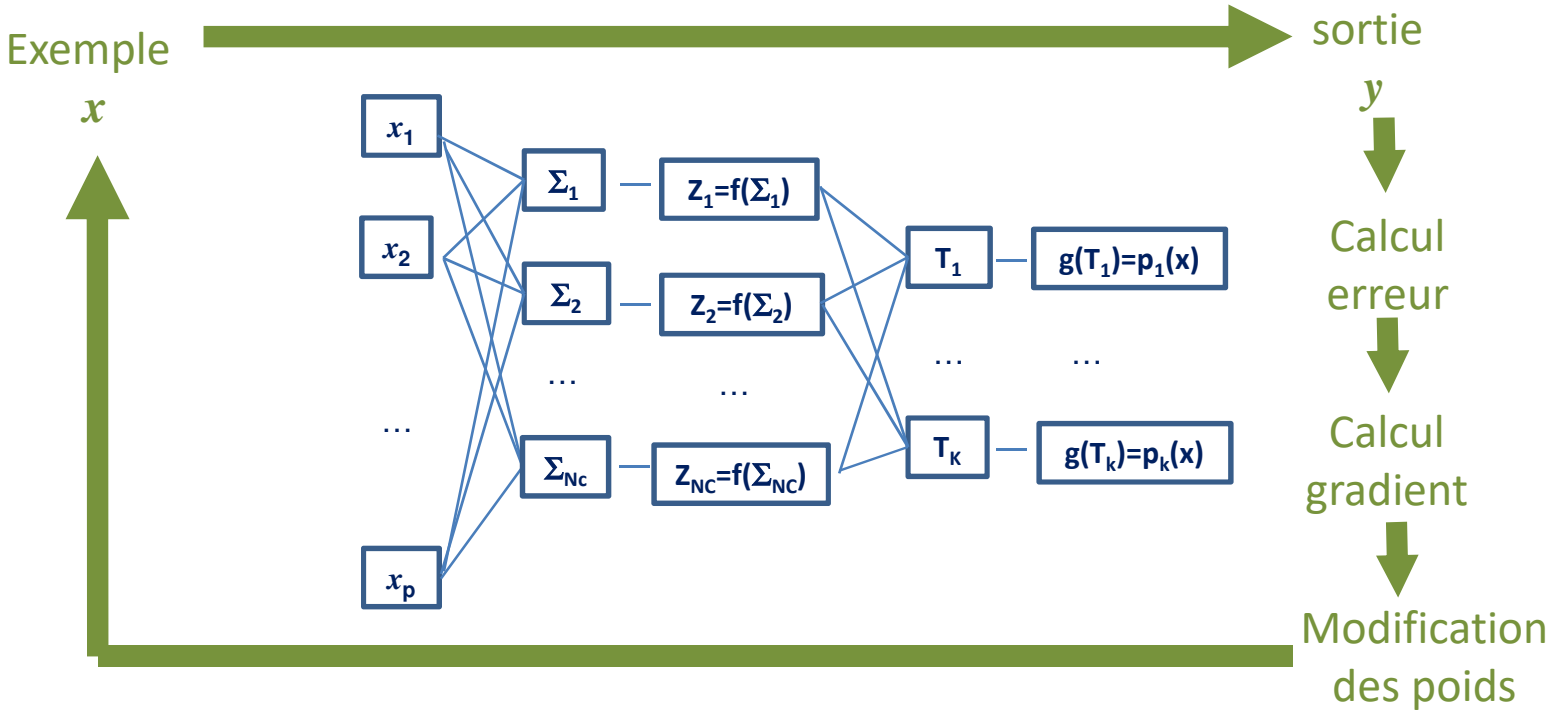
PRINCIPE DE LA RÉTRO-PROPAGATION

(*) se référer au cours d'optimisation pour les méthodes de gradient (BFGS,…)

Les poids sont ajustés par une méthode de descente du gradient (*)

$$w^{(q+1)} = w^{(q)} - \eta \nabla E(w^{(q)})$$

Toute la difficulté réside dans le calcul du gradient. La méthode de rétro-propagation permet un calcul efficace de $\nabla E(w)$.



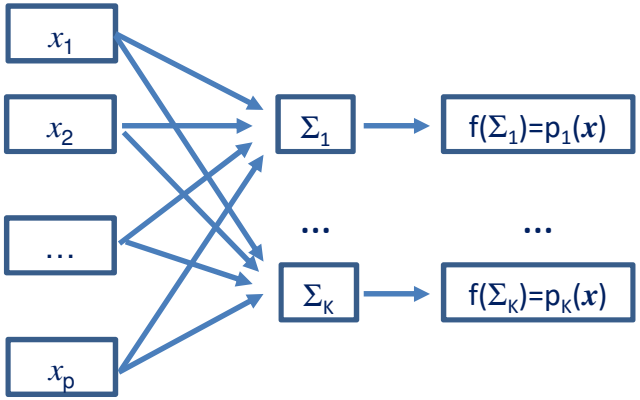
Tant que l'erreur>ε

- Propager un exemple dans le réseau
- Calculer l'erreur en sortie puis le gradient
- Modifier les poids dans la direction opposée au gradient

DESCENTE DU GRADIENT : CAS PERCEPTRON MONOCOUCHE

Afin de simplifier le propos, nous allons considérer le perceptron monocouche avec un neurone par classe k=1,…,K

$$p_k(\boldsymbol{x}) = f(\sum_{j=1}^{p} w_{jk} x_j)$$



L'algorithme de Widrow-Hoff est une variante de la descente du gradient. Les poids du réseaux ne sont pas ajustés une fois que l'erreur est calculée sur tous les exemples de la base d'apprentissage mais **après <u>chaque</u> exemple**.

Voici l'algorithme pour un seul neurone. Il faut répéter sur chacun des neurones, k=1,…,K.

```
Initialiser les poids w_ik, i=1,…,p
Randomiser l'ordre des exemples
Pour tout exemple (x, y)
   Calculer la sortie du réseau p_k(x)
   Pour i=1,…,p
      Δw_ik = α[y − p_k(x)] x_i (*)
      w_ik ← w_ik + Δw_ik
   Fin pour i
Fin pour tout exemple
```

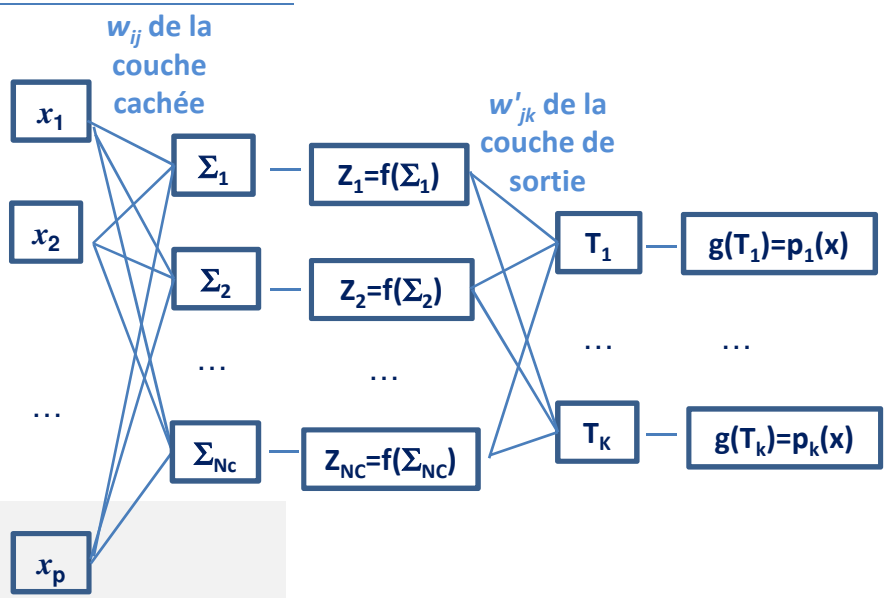
α = taux d'apprentissage

(*) démonstration en annexe

DESCENTE DU GRADIENT : CAS PERCEPTRON MULTICOUCHE

Dans le cas multi-couches, il y a deux types de poids à ajuster :

- ceux de la couche cachée, w_{ij}, de l'entrée x_i vers le neurones Z_j
- ceux de la couche de sortie, w'_{jk}, du neurone j vers la sortie k




```
Initialiser les poids
Répéter jusqu'à critère de fin
    Randomiser l'ordre des exemples
    Pour tout exemple (x, y)
        Calculer les sorties du réseau p_k(x)
        Calculer les delta :
            • Pour chaque sortie k
                Calculer δ_k
                Δw'_jk ← −αδ_k z_j
            • Pour chaque neurone caché j
                Calculer δ_j
                Δw_ij ← −αδ_j x_i
        Ajuster les poids :
            w'_jk ← w'_jk + Δw_jk
            w_ij ← w_ij + Δw_ij
```

Le calcul du delta n'est pas le même suivant la couche.

Son expression sera démontrée en cours

SUR-APPRENTISSAGE

La méthode de rétro-propagation conduit souvent à un sur-apprentissage.

Afin de limiter ce risque, on introduit une pénalité dans la fonction de coût à optimiser.

C'est la méthode de régularisation du ***weigth decay***. L'objectif est de limiter la valeur absolue des poids en utilisant la pénalisation

$$\Omega = \frac{1}{2}\sum_i w_i^2$$

La fonction de coût devient alors

$$E'(w) = E(w) + \tau\,\Omega$$

Le paramètre τ détermine l'importance relative des deux termes. Si τ est trop grand les poids tendent rapidement vers 0 et le modèle ne tient plus compte des données. Si τ est trop petit, le terme de régularisation est négligeable et le réseau de neurones peut être sur-ajusté.

Remarque : cette méthode est appelée *ridge regression* dans le cas de modèles linéaires

PARAMÈTRES *VS* HYPERPARAMÈTRES

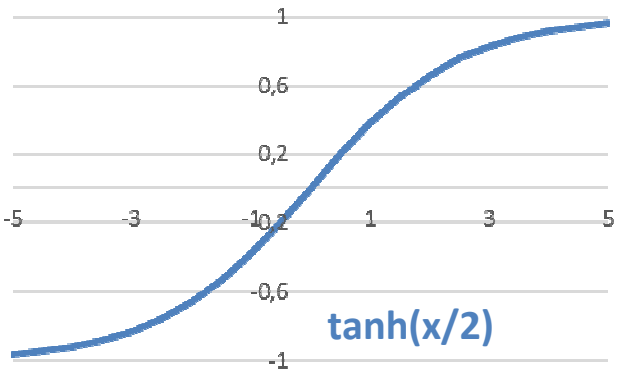
Comme nous l'avons vu, un réseau de neurones dépend de paramètres, les poids, qui sont ajustés par descente du gradient pour minimiser une erreur par rapport à une base d'apprentissage.

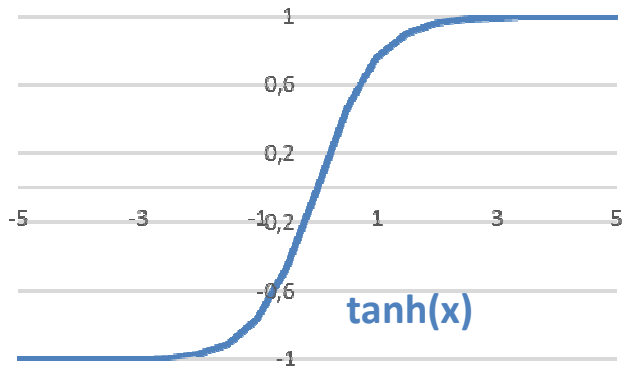
Mais un réseau de neurones dépend aussi d'**hyperparamètres**, tels que le nombre de couches cachées, le nombre de neurones dans une couche, le taux d'apprentissage.

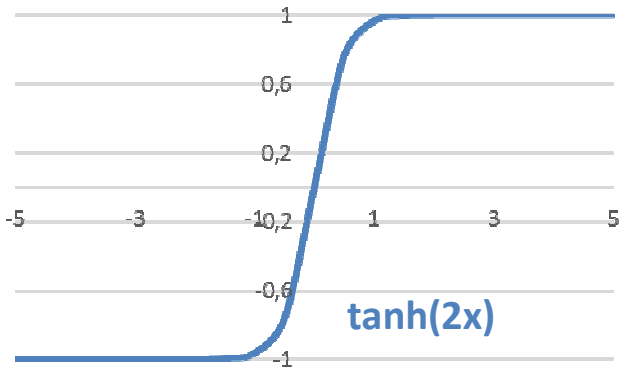
Pour déterminer ces hyperparamètres, on utilise une base de validation. On teste plusieurs valeurs de ces hyperparamètres et on choisit celles qui minimise l'erreur de validation.

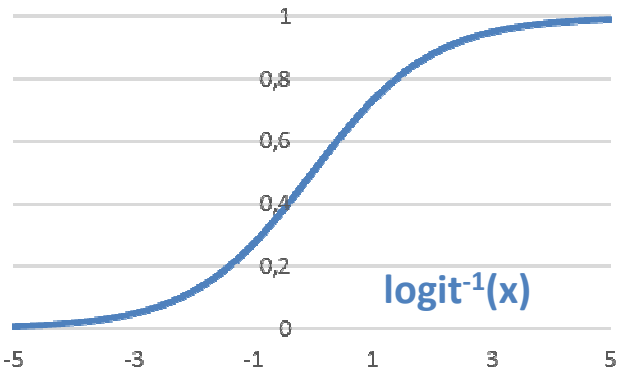
Il est conseillé de faire appel à un plan d'expériences (grille ou hypercube latin par exemple) pour organiser les valeurs à tester.

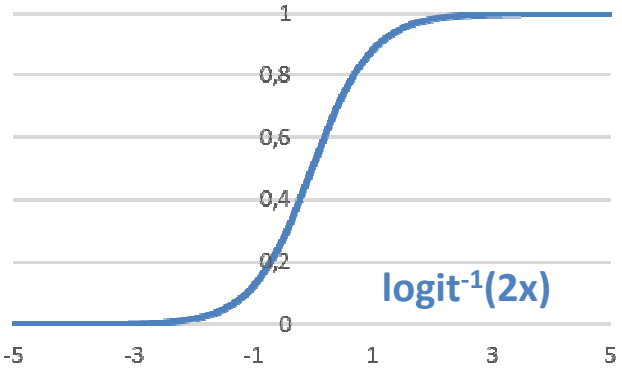
ANNEXE

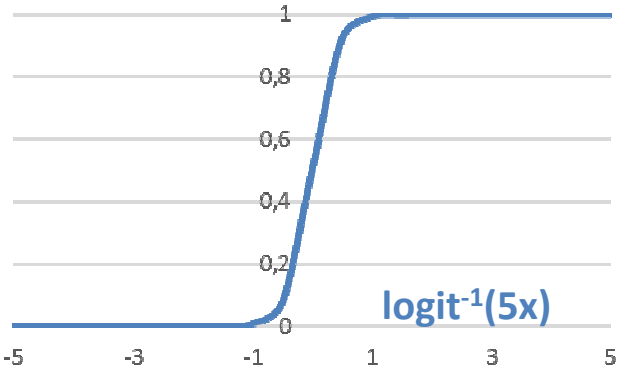

1
0,6
0,2
-0,2
-0,6
-1
-5
-3
-1
1
3
5
tanh(x/2)


1
0,6
0,2
-0,2
-0,6
-1
-5
-3
-1
1
3
5
tanh(x)


1
0,6
0,2
-0,2
-0,6
-1
-5
-3
-1
1
3
5
tanh(2x)


1
0,8
0,6
0,4
0,2
0
-5
-3
-1
1
3
5
logit-1(x)


1
0,8
0,6
0,4
0,2
0
-5
-3
-1
1
3
5
logit-1(2x)


1
0,8
0,6
0,4
0,2
0
-5
-3
-1
1
3
5
logit-1(5x)

ANNEXE

Démonstration de « la règle du delta » pour perceptron multicouches (1/2)

Pour un exemple ($\boldsymbol{x}$,y). Notons y_k=1 si y∈ C_k et 0 sinon

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{K}(y_k - p_k(\boldsymbol{x}))^2 = \frac{1}{2}\sum_{k=1}^{K}(y_k - g(T_k))^2 = \sum_{k=1}^{K}\frac{1}{2}E_k(\boldsymbol{w})$$

où $T_k = \sum_{j=1}^{N_C} w'_{jk} Z_j$ avec $Z_j = f(\Sigma_j)$ et $\Sigma_j = \sum_{i=1}^{p} w_{ij} x_i$.

Poids de la 2ème couche

Poids de la 1ère couche

Dérivées partielles par rapport aux poids de la première couche

$$\frac{\partial E(\boldsymbol{w})}{\partial w'_{jk}} = \frac{\partial E(\boldsymbol{w})}{\partial T_k} \times \frac{\partial T_k}{\partial w'_{jk}}$$

- $\frac{\partial E(\boldsymbol{w})}{\partial T_k} = \frac{1}{2}\frac{\partial}{\partial T_k}E_k(\boldsymbol{w}) = \frac{1}{2}\frac{\partial}{\partial T_k}(y_k - g(T_k))^2 = \frac{1}{2}\times 2(y_k - g(T_k))\frac{\partial}{\partial T_k}(y_k - g(T_k)) = -(y_k - g(T_k))g'(T_k)$
- $\frac{\partial T_k}{\partial w'_{jk}} = \frac{\partial}{\partial w'_{jk}}\sum_{j=1}^{N_C} w'_{jk} Z_j = Z_j$

Erreur commise

$$\Rightarrow \frac{\partial E(\boldsymbol{w})}{\partial w'_{jk}} = \delta_k Z_j \quad \text{où} \quad \delta_k = -(y_k - g(T_k))g'(T_k)$$

Remarque : Si g(x) est la fonction logistique, g'(x)=g(x)(1-g(x))

ANNEXE

Démonstration de « la règle du delta » pour perceptron multicouches (2/2)

Pour un exemple ($\boldsymbol{x}$,y). Notons y_k=1 si y$\in C_k$ et 0 sinon

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{K}(y_k - p_k(\boldsymbol{x}))^2 = \frac{1}{2}\sum_{k=1}^{K}(y_k - g(T_k))^2 = \sum_{k=1}^{K}\frac{1}{2}E_k(\boldsymbol{w})$$

où $T_k = \sum_{j=1}^{N_C} w'_{jk} Z_j$ avec $Z_j = f(\Sigma_j)$ et $\Sigma_j = \sum_{i=1}^{p} w_{ij} x_i$.

Poids de la 2ème couche

Poids de la 1ère couche

Dérivées partielles par rapport aux poids de la première couche

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{ij}} = \sum_{k=1}^{K}\frac{1}{2}\frac{\partial E_k(\boldsymbol{w})}{\partial w_{ij}} \quad \text{où} \quad \frac{\partial E_k(\boldsymbol{w})}{\partial w_{ij}} = \frac{\partial E_k(\boldsymbol{w})}{\partial T_k} \times \frac{\partial T_k}{\partial Z_j} \times \frac{\partial Z_j}{\partial \Sigma_j} \times \frac{\partial \Sigma_j}{\partial w_{ij}}$$

- $\frac{\partial E_k(\boldsymbol{w})}{\partial T_k} = \frac{\partial}{\partial T_k}(y_k - g(T_k))^2 = -2g'(T_k)(y_k - g(T_k)) = 2\delta_k$
- $\frac{\partial T_k}{\partial Z_j} = \frac{\partial}{\partial Z_j}\sum_{j=1}^{N_C} w'_{jk} Z_j = w'_{jk}$
- $\frac{\partial \Sigma_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\sum_{i=1}^{p} w_{ij} x_i = x_i$
- $\frac{\partial Z_j}{\partial \Sigma_j} = \frac{\partial}{\partial \Sigma_j} f(\Sigma_j) = f'(\Sigma_j)$

$$\Rightarrow \quad \frac{\partial E(\boldsymbol{w})}{\partial w_{ij}} = \delta_j x_i \quad \text{où} \quad \delta_j = \left(\sum_{k=1}^{K}\delta_k w'_{jk}\right) f'(\Sigma_j)$$

Remarque : Si f(x)=tanh(x) alors f'(x)=1-f(x)^2