Réseaux de neurones

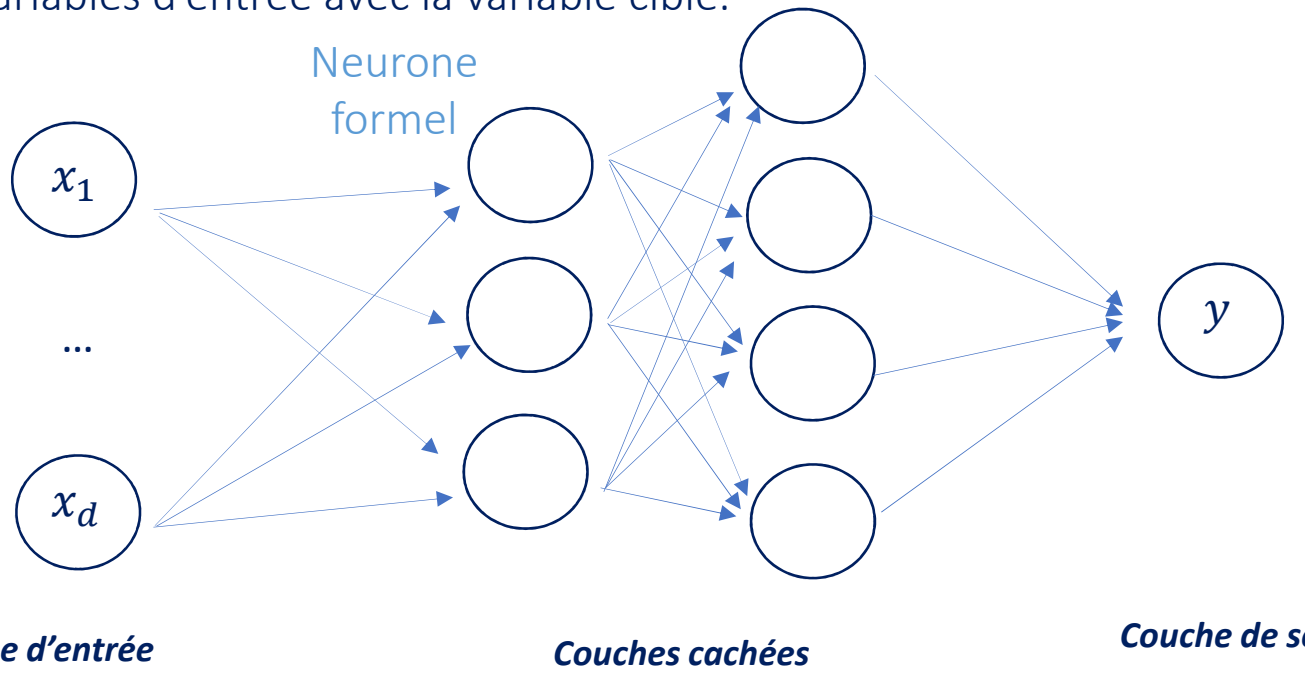
Apprentissage supervisé de classification

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

Réseau de neurones

Notons $\mathbf{x} = (x_1, \dots, x_d)$ les variables explicatives et y la variable cible.

Un réseau de neurones est constitué de couches successives de neurones formels de façon à connecter les variables d'entrée avec la variable cible.



- Si la variable cible est continue alors la sortie du réseau de neurone est unique et correspond à la valeur prédite $\hat{y}$ (hors programme).
- Si la variable cible est binaire alors la sortir du réseau de neurones est unique et correspond à la probabilité de la classe positive, $P(y = 1|x)$.
- Si la variable cible admet plus de K>2 classes alors le réseau de neurones a autant de sorties de de classes, chaque sortie est la probabilité d'une classe, $P(y \in C_k|x)$.

Neurone formel

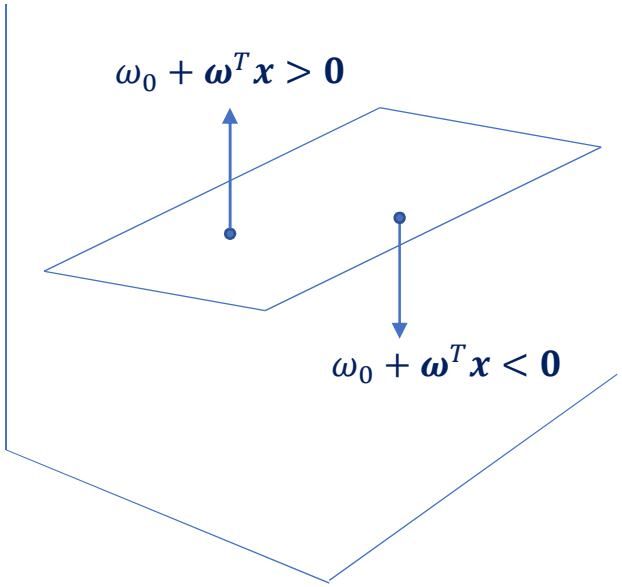
Combinaison linéaire des variables

Un neurone se définit en deux étapes :

1) Une combinaison linéaire des variables :

$$\Sigma = \omega_0 + \sum_{i=1}^{d} \omega_i x_i = \omega_0 + \boldsymbol{\omega}^T \boldsymbol{x}$$

où $\boldsymbol{\omega}^T = (\omega_1, \dots, \omega_d)$ sont les poids. La constante ω_0, appelé le biais, est facultative.



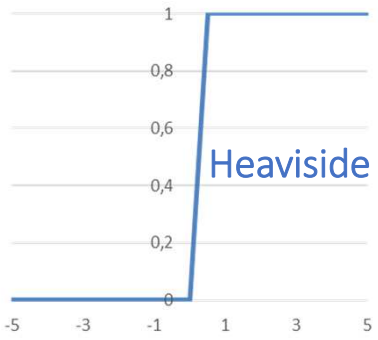
- $\boldsymbol{\omega}$ définit une frontière linéaire (inclinaison de l'hyperplan)
- ω_0 permet de positionner l'hyperplan

Neurone formel

Fonction d'activation

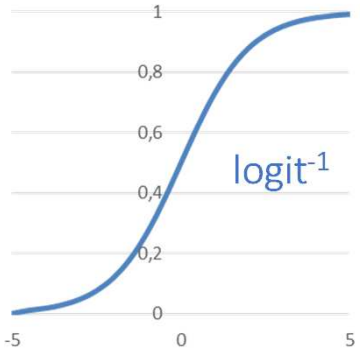
2) Une fonction d'activation ou de transfert

$$f(\Sigma) = \quad f(\omega_0 + \sum_{i=1}^{d} \omega_i x_i)$$



Fonction de Heaviside
(historique)

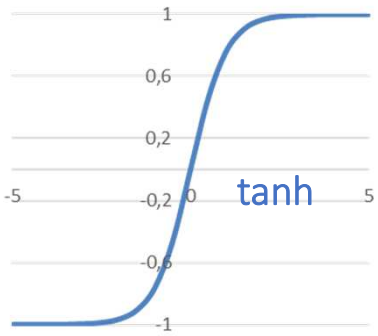
$$H(x) = \begin{cases} 0 \; si \; x < 0 \\ 1 \; si \; x \geq 0 \end{cases}$$



Fonction
logistique inverse ou sigmoïde

$$sigm(x) = \frac{1}{1 + e^{-x}} \in [0,1]$$

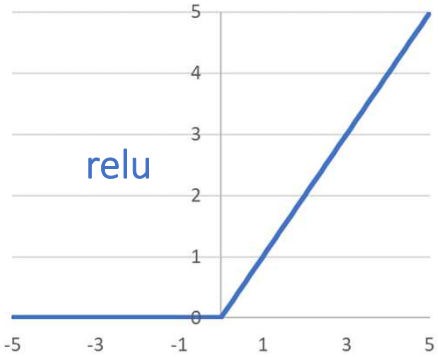
$$sigm'(x) = sigm(x)(1 - sigm(x))$$



Fonction
tangente hyperbolique

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \in [-1,1]$$

$$tanh'(x) = 1 - tanh\,(x)^2$$



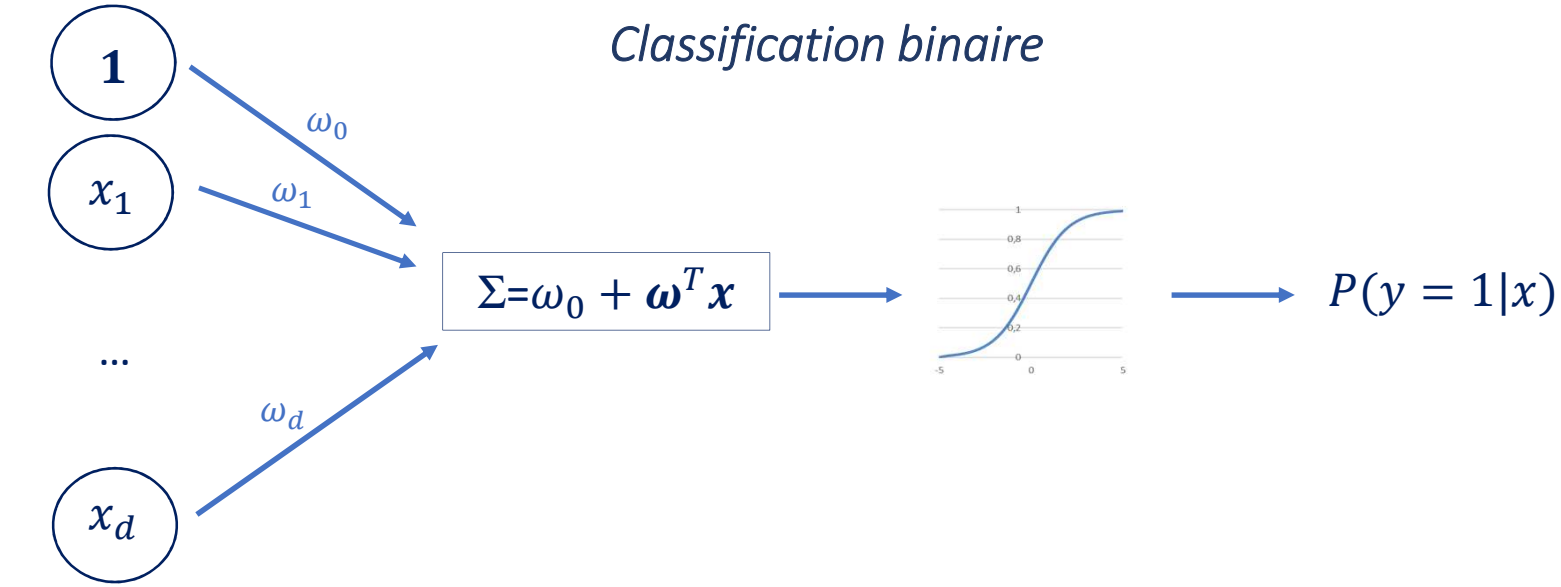
Fonction
relu

$$relu(x) = \max(0, x) \in \mathrm{IR}^+$$

$$\mathrm{relu}'(x) = \mathbf{1}_{x>0}$$

Neurone formel

Classification binaire



Supposons le cas particulier de la classification binaire, c.-à-d. quand la variable cible y prend deux modalités codées par 0 et 1. Le neurone retourne alors la probabilité conditionnelle,

$$f(\Sigma) = P(y = 1|\boldsymbol{x}).$$

Deux fonctions d'activation possibles pour obtenir un résultat entre 0 et 1 : Heaviside et sigmoïde.

$$P(y = 1|\boldsymbol{x}) = sigm(\omega_0 + \boldsymbol{\omega}^T \boldsymbol{x}) = \frac{1}{1 + e^{-(\omega_0 + \boldsymbol{\omega}^T \boldsymbol{x})}}$$

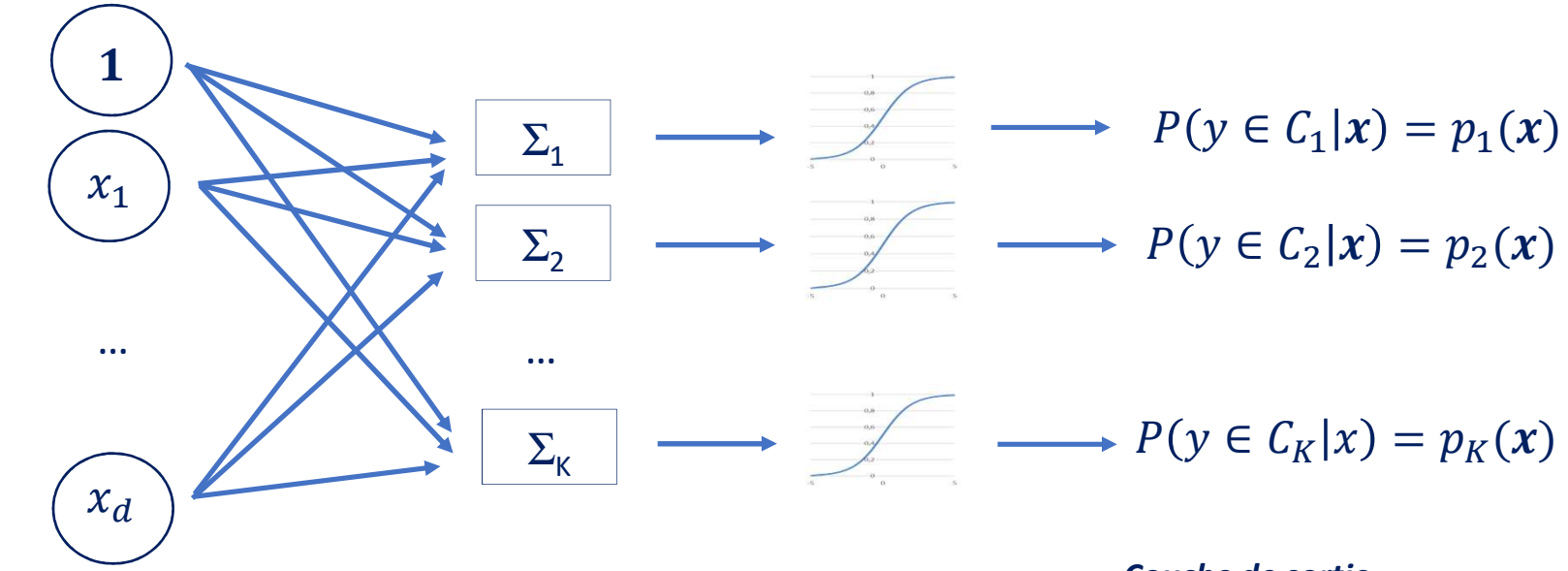
L'entrée $\boldsymbol{x}$ est classée 1 si $P(y = 1|\boldsymbol{x}) > 0{,}5$ c.-à-d. si $\omega_0 + \boldsymbol{\omega}^T \boldsymbol{x} > 0$

L'entrée $\boldsymbol{x}$ est classée 0 si $P(y = 1|\boldsymbol{x}) < 0{,}5$ c.-à-d. si $\omega_0 + \boldsymbol{\omega}^T \boldsymbol{x} < 0$

Perceptron multivarié

Définition et fonction softmax

Supposons le cas général de la classification binaire et notons $C_1, \dots, C_K$ les classes de la variable cible y. Le *perceptron multivarié* va construire autant de neurones formels qu'il y a de classes. Chaque neurone donne la probabilité d'un classe sachant $\boldsymbol{x}$.



Afin de s'assurer que la somme des probabilités égale 1, la seule fonction d'activation possible est la fonction *softmax*. Cette fonction prend en entrée un vecteur $z = (\Sigma_1, \dots, \Sigma_K)$ et retourne le vecteur

$$\left(\frac{e^{\Sigma_1}}{\sum_{i=1}^{K} e^{\Sigma_i}}, \dots, \frac{e^{\Sigma_K}}{\sum_{i=1}^{K} e^{\Sigma_i}} \right)$$

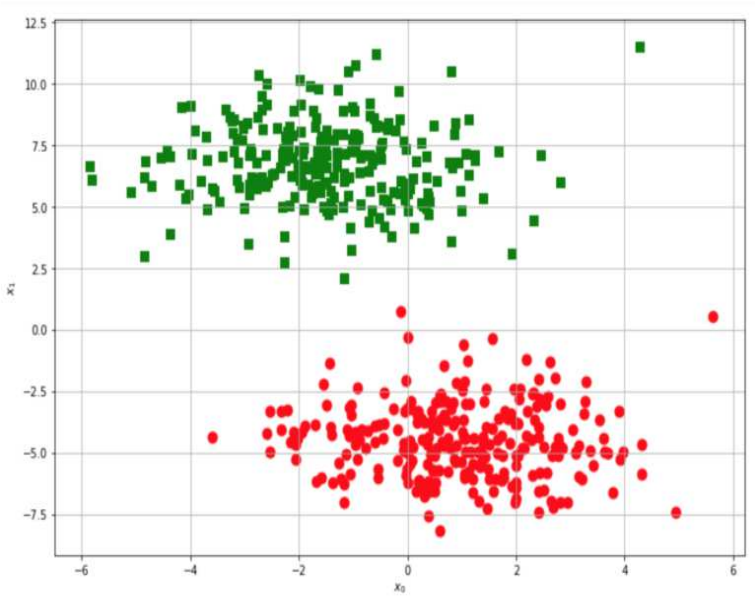
Somme des coordonnées = 1

La classe prédite est celle ayant la plus grande probabilité.

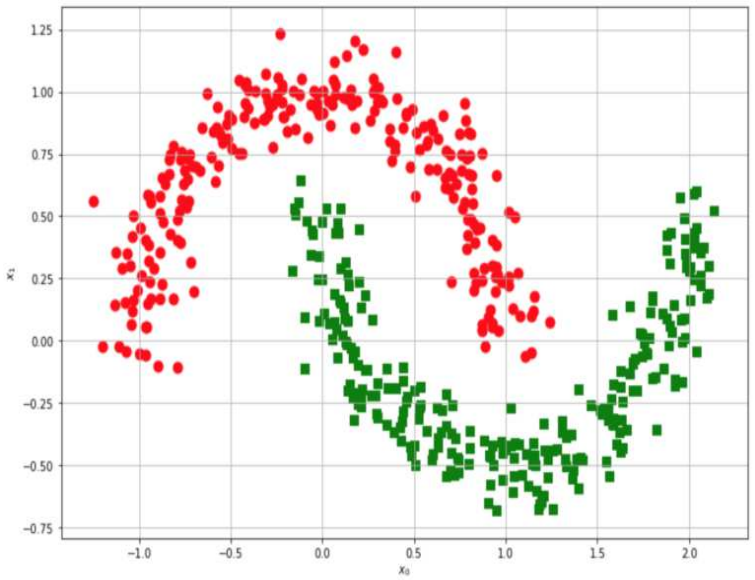
Perceptron multivarié

Séparateur linéaire

De par sa construction (fonction d'activation de fonctions affines), le perceptron multivarié ne peut traiter que des frontières séparatrices linéaires.



Frontière linéaire



Frontière non linéaire

Pour déterminer des frontières complexes, il est nécessaire d'ajouter des couches de neurones. Cette nouvelle architecture s'appelle le *perceptron multi-couches*.

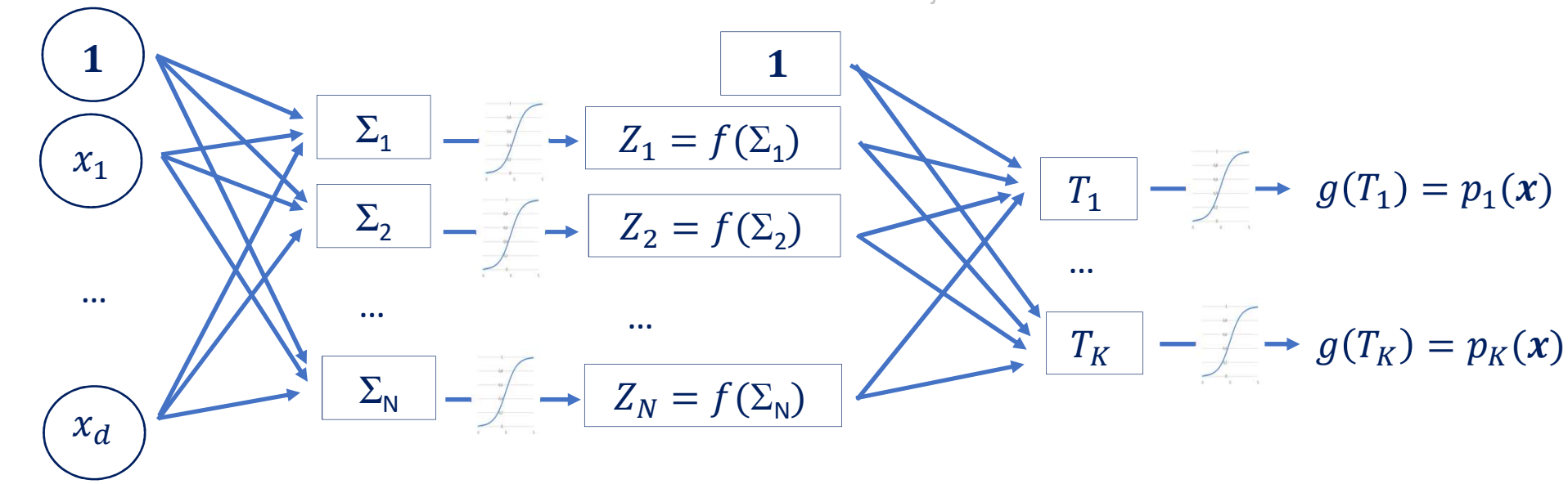
Le perceptron multicouches

Définition

Un perceptron multicouches est une combinaison linéaire de N neurones formels à laquelle on applique une (nouvelle) fonction d'activation,

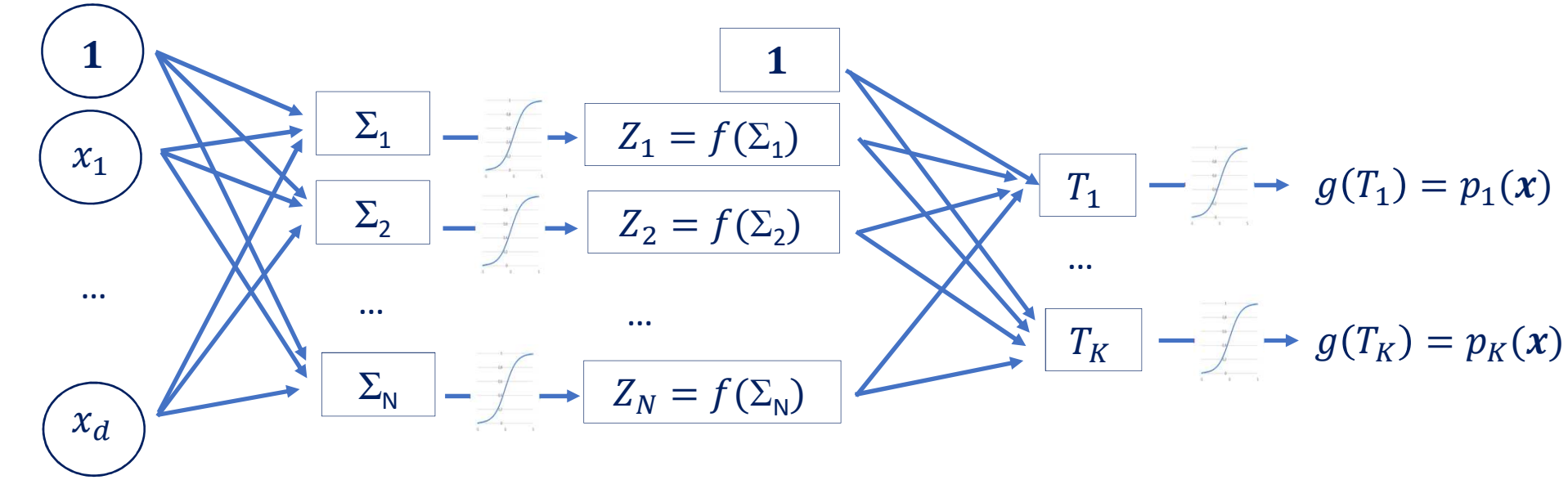
$$g\left(\underbrace{\omega'_0 + \sum_{j=1}^{N} \omega'_j \underbrace{f(\Sigma_j)}_{=Z_j}}_{=T}\right)$$

Formule pour une sortie (classe) du réseau de neurones



Le perceptron multicouches

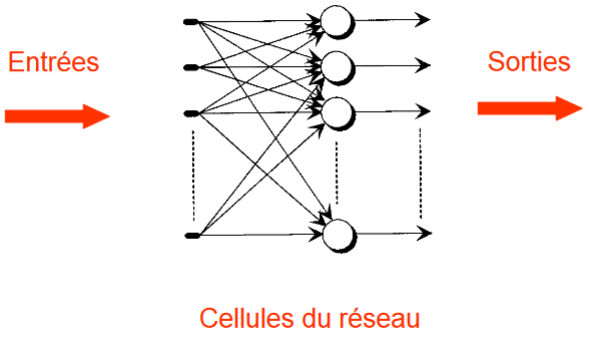
Définition



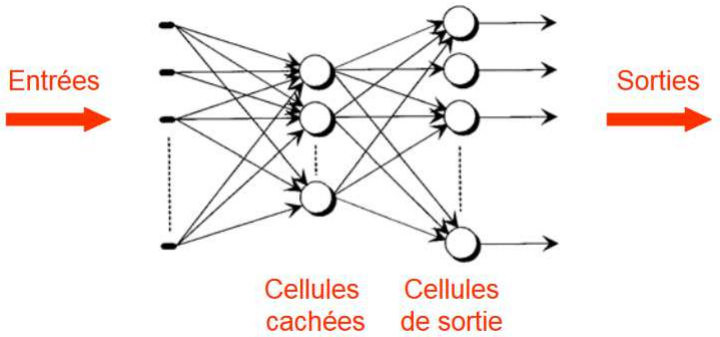
- La fonction d'activation f entre la couche d'entrée et la couche cachée est quelconque.
- La fonction d'activation g entre la couche cachée et la couche de sortie est la fonction softmax s'il y a K>2 classes et la fonction sigmoïde s'il y a 2 classes.
- Le nombre de neurones N de la couche cachée est indépendant du nombre de classes K en sortie
- 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 de neurones.

Différentes architectures de réseaux

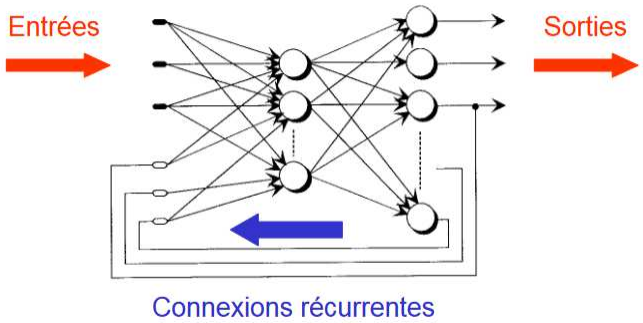
Mono-couche



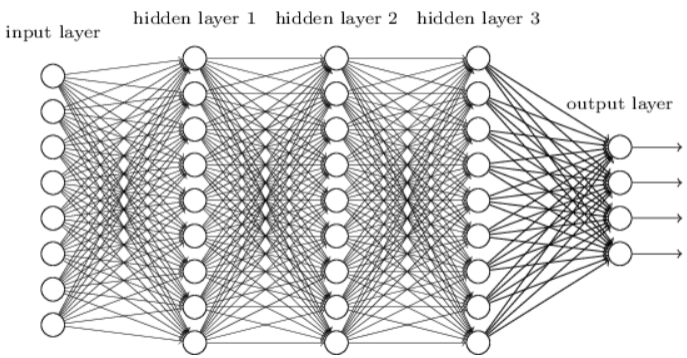
Multi-couche



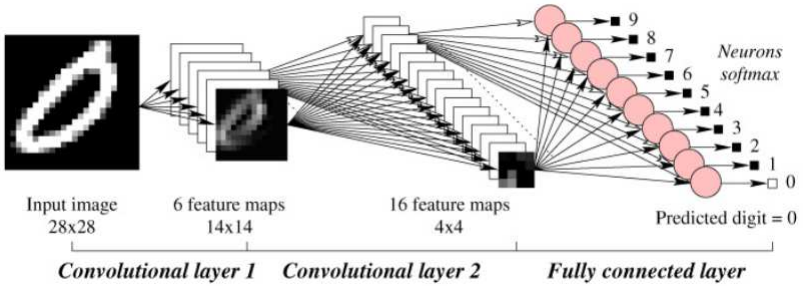
Récurrents



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

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 : cas binaire

Une fois la structure du réseau fixée, il faut déterminer les poids $\boldsymbol{\omega}$ tels que la sortie du réseau soit la plus « proche » possible des observations. En classification, il existe deux *fonctions de coût*. Notons $\boldsymbol{x_i} = (x_{i1}, \dots, x_{id})$ 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 $p(\boldsymbol{x_i}, \boldsymbol{\omega})$ la sortie du réseau de neurones de poids $\boldsymbol{\omega}$.

L'erreur quadratique moyenne

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

L'entropie croisée

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

$$E(\boldsymbol{\omega}) = -\sum_{i=1}^{n} y_i \ln\big(p(\boldsymbol{x_i}, \boldsymbol{\omega})\big) + (1 - y_i)\ln\big(1 - p(\boldsymbol{x_i}, \boldsymbol{\omega})\big)$$

y_i	$p(\boldsymbol{x_i}, \boldsymbol{\omega})$	$\hat{y}_i$
0	0,49	0
1	0,90	1
1	0,51	1
0	0,55	1

Pourquoi $p(\boldsymbol{x_i}, \boldsymbol{\omega})$ et non $\hat{y}_i$ dans le calcul de la fonction de coût?

Ajustement du réseaux de neurones

La fonction de coût : cas binaire

Une fois la structure du réseau fixée, il faut déterminer les poids $\boldsymbol{\omega}$ tels que la sortie du réseau soit la plus « proche » possible des observations. En classification, il existe deux *fonctions de coût*. Notons $\boldsymbol{x_i} = (x_{i1}, \dots, x_{ip})$ 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 $p(\boldsymbol{x_i}, \boldsymbol{\omega})$ la sortie du réseau de neurones de poids $\boldsymbol{\omega}$.

L'erreur quadratique moyenne

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

L'entropie croisée

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

$$E(\boldsymbol{\omega}) = -\sum_{i=1}^{n} y_i \ln\big(p(\boldsymbol{x_i}, \boldsymbol{\omega})\big) + (1 - y_i)\ln\big(1 - p(\boldsymbol{x_i}, \boldsymbol{\omega})\big)$$

y_i	$p(\boldsymbol{x_i}, \boldsymbol{\omega})$	$\hat{y}_i$	
0	0,49	0	$\Rightarrow (y_i - \hat{y}_i)^2 = (0-0)^2 = 0$
1	0,90	1	
1	0,51	1	
0	0,55	1	$\Rightarrow (y_i - \hat{y}_i)^2 = (0-1)^2 = 1$

Avec $\hat{y}_i$ un point mal classé a une contribution de 1 et un point bien classé ne contribue pas même s'il est proche de la frontière.

Ajustement du réseaux de neurones

La fonction de coût : cas binaire

Une fois la structure du réseau fixée, il faut déterminer les poids $\boldsymbol{\omega}$ tels que la sortie du réseau soit la plus « proche » possible des observations. En classification, il existe deux *fonctions de coût*. Notons $\boldsymbol{x_i} = (x_{i1}, \dots, x_{ip})$ 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 $p(\boldsymbol{x_i}, \boldsymbol{\omega})$ la sortie du réseau de neurones de poids $\boldsymbol{\omega}$.

L'erreur quadratique moyenne

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

L'entropie croisée

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

$$E(\boldsymbol{\omega}) = -\sum_{i=1}^{n} y_i \ln\big(p(\boldsymbol{x_i}, \boldsymbol{\omega})\big) + (1 - y_i)\ln\big(1 - p(\boldsymbol{x_i}, \boldsymbol{\omega})\big)$$

y_i	$p(\boldsymbol{x_i}, \boldsymbol{\omega})$	$\hat{y}_i$	écarts quadratiques
0	0,49	0	(0-0,49)²=0,24
1	0,90	1	(1-0,90)²=0,01
1	0,51	1	(1-0,51)²=0,24
0	0,55	1	(0-0,55)²=0,30
			$E(\boldsymbol{\omega})$=0,40

Calculer la fonction de coût avec $p(\boldsymbol{x_i}, \boldsymbol{\omega})$ et non $\hat{y}_i$ permet de donner plus de poids aux points qui sont près de la frontière et moins à ceux qui sont loin.

Ajustement du réseaux de neurones

La fonction de coût : cas multi-classes (K>2)

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

L'erreur quadratique moyenne

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

L'entropie croisée

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

Exemple de 3 classes

y_i	$p_1(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_2(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_3(\boldsymbol{x}_i, \boldsymbol{\omega})$	$\hat{y}_i$
1	0,6	0,3	0,1	1
1	0,4	0,5	0,1	2
2	0,2	0,45	0,35	2
3	0,55	0,1	0,45	1
3	0	0,1	0,9	3

Ajustement du réseaux de neurones

La fonction de coût : cas multi-classes (K>2)

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

L'erreur quadratique moyenne

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

L'entropie croisée

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

Exemple de 3 classes

y_i	$p_1(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_2(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_3(\boldsymbol{x}_i, \boldsymbol{\omega})$	$\hat{y}_i$
1	0,6	0,3	0,1	1
1	0,4	0,5	0,1	2
2	0,2	0,45	0,35	2
3	0,55	0,1	0,45	1
3	0	0,1	0,9	3

$$\Rightarrow (y_i - \hat{y}_i)^2 = (1-2)^2 = 1$$

Ajustement du réseaux de neurones

La fonction de coût : cas multi-classes (K>2)

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

L'erreur quadratique moyenne

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

L'entropie croisée

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

Exemple de 3 classes

y_i	$p_1(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_2(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_3(\boldsymbol{x}_i, \boldsymbol{\omega})$	$\hat{y}_i$
1	0,6	0,3	0,1	1
1	0,4	0,5	0,1	2
2	0,2	0,45	0,35	2
3	0,55	0,1	0,45	1
3	0	0,1	0,9	3

$\Rightarrow (y_i - \hat{y}_i)^2 = (1-2)^2 = 1$

$\Rightarrow (y_i - \hat{y}_i)^2 = (3-1)^2 = 4$

Calculer l'erreur avec $\boldsymbol{p}(\boldsymbol{x}_i, \boldsymbol{\omega})$ et non $\hat{y}_i$ permet de donner le même poids à toutes les classes mal prédites.

Ajustement du réseaux de neurones

La fonction de coût : cas multi-classes (K>2)

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

L'erreur quadratique moyenne

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

L'entropie croisée

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

Exemple de 3 classes

y_i	$p_1(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_2(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_3(\boldsymbol{x}_i, \boldsymbol{\omega})$	$\hat{y}_i$	y_{i1}	y_{i2}	y_{i3}
1	0,6	0,3	0,1	1	1	0	0
1	0,4	0,5	0,1	2	1	0	0
2	0,2	0,45	0,35	2	0	1	0
3	0,55	0,1	0,45	1	0	0	1
3	0	0,1	0,9	3	0	0	1

Ajustement du réseaux de neurones

La fonction de coût : cas multi-classes (K>2)

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

L'erreur quadratique moyenne

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

L'entropie croisée

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

Exemple de 3 classes

y_i	$p_1(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_2(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_3(\boldsymbol{x}_i, \boldsymbol{\omega})$	$\hat{y}_i$	y_{i1}	y_{i2}	y_{i3}	écarts quadratiques
1	0,6	0,3	0,1	1	1	0	0	$(0,6-1)^2+(0,3-0)^2+(0,1-0)^2=0,26$
1	0,4	0,5	0,1	2	1	0	0	$(0,4-1)^2+(0,5-0)^2+(0,1-0)^2=0,62$
2	0,2	0,45	0,35	2	0	1	0	$(0,2-0)^2+(0,45-1)^2+(1,0-1)^2=0,465$
3	0,55	0,1	0,45	1	0	0	1	$(0,1-0)^2+(0,55-2)^2+(2,0-2)^2=0,615$
3	0	0,1	0,9	3	0	0	1	$(0-0)^2+(0,1-2)^2+(2,0-2)^2=0,02$
								$E(\boldsymbol{\omega})=0,99$

Ajustement du réseaux de neurones

La fonction de coût : cas multi-classes (K>2)

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

L'erreur quadratique moyenne

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

L'entropie croisée

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

Exemple de 3 classes

y_i	$p_1(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_2(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_3(\boldsymbol{x}_i, \boldsymbol{\omega})$	$\hat{y}_i$	y_{i1}	y_{i2}	y_{i3}	écarts quadratiques
1	0,6	0,3	0,1	1	1	0	0	$(0,6-1)^2+(0,3-0)^2+(0,1-0)^2=0,26$
1	0,4	0,5	0,1	2	1	0	0	$(0,4-1)^2+(0,5-0)^2+(0,1-0)^2=0,62$
2	0,2	0,45	0,35	2	0	1	0	$(0,2-0)^2+(0,45-1)^2+(1,0-1)^2=0,465$
3	0,1	0,55	0,45	2	0	0	1	$(0,1-0)^2+(0,55-2)^2+(2,0-2)^2=0,615$
3	0	0,1	0,9	3	0	0	1	$(0-0)^2+(0,1-2)^2+(2,0-2)^2=0,02$
								$E(\boldsymbol{\omega})=0,99$

Ajustement du réseaux de neurones

La fonction de coût : cas multi-classes (K>2)

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

L'erreur quadratique moyenne

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

L'entropie croisée

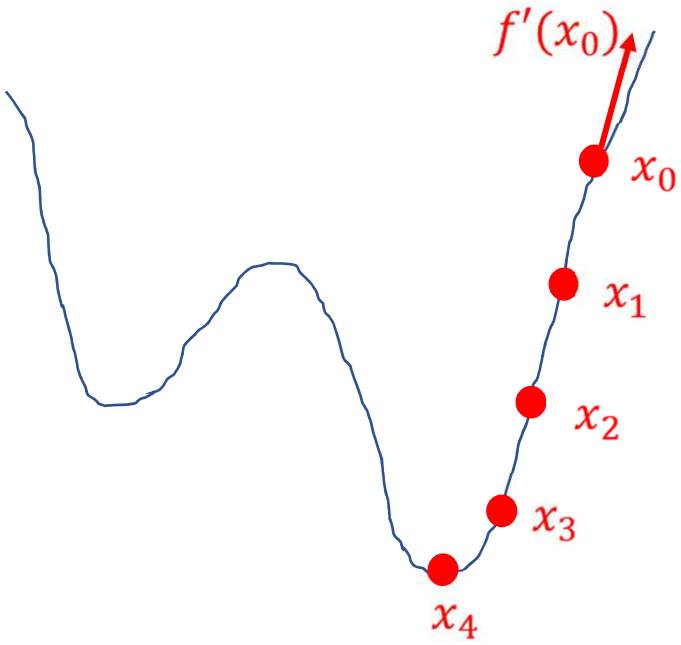
$$E(\boldsymbol{\omega}) = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik} \ln\left(p_k(\boldsymbol{x}_i, \boldsymbol{\omega})\right)$$

Exemple de 3 classes

y_i	$p_1(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_2(\boldsymbol{x}_i, \boldsymbol{\omega})$	$p_3(\boldsymbol{x}_i, \boldsymbol{\omega})$	$\hat{y}_i$	y_{i1}	y_{i2}	y_{i3}	écarts quadratiques
1	0,6	0,3	0,1	1	1	0	0	$(0,6-1)^2+(0,3-0)^2+(0,1-0)^2=0,26$
1	0,4	0,5	0,1	2	1	0	0	$(0,4-1)^2+(0,5-0)^2+(0,1-0)^2=0,62$
2	0,2	0,45	0,35	2	0	1	0	$(0,2-0)^2+(0,45-1)^2+(1,0-1)^2=0,465$
3	0,1	0,55	0,45	2	0	0	1	$(0,1-0)^2+(0,55-2)^2+(2,0-2)^2=0,615$
3	0	0,1	0,9	3	0	0	1	$(0-0)^2+(0,1-2)^2+(2,0-2)^2=0,02$
								$E(\boldsymbol{\omega})=0,99$

Ajustement du réseaux de neurones

Descente du gradient



Algorithme pour minimiser une fonction f

- 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

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

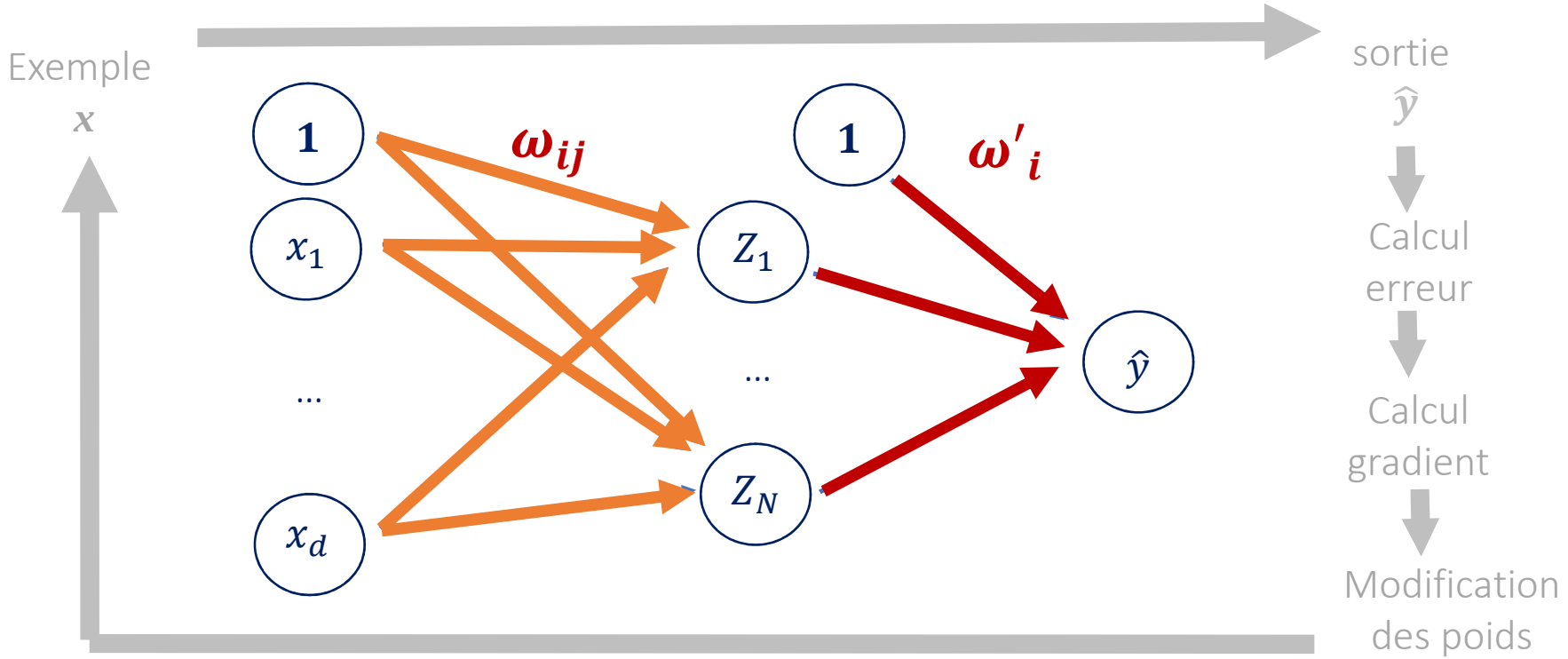
Ajustement du réseaux de neurones

Principe de la rétro-propagation du gradient

Les poids sont donc ajustés par une méthode de descente du gradient

$$\boldsymbol{\omega}^{(q+1)} = \boldsymbol{\omega}^{(q)} - \alpha \nabla E(\boldsymbol{\omega}^{(q)}).$$

Toute la difficulté réside dans le calcul du gradient. La méthode de rétro-propagation permet un calcul efficace de $\nabla E(\boldsymbol{\omega}^{(q)})$.



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

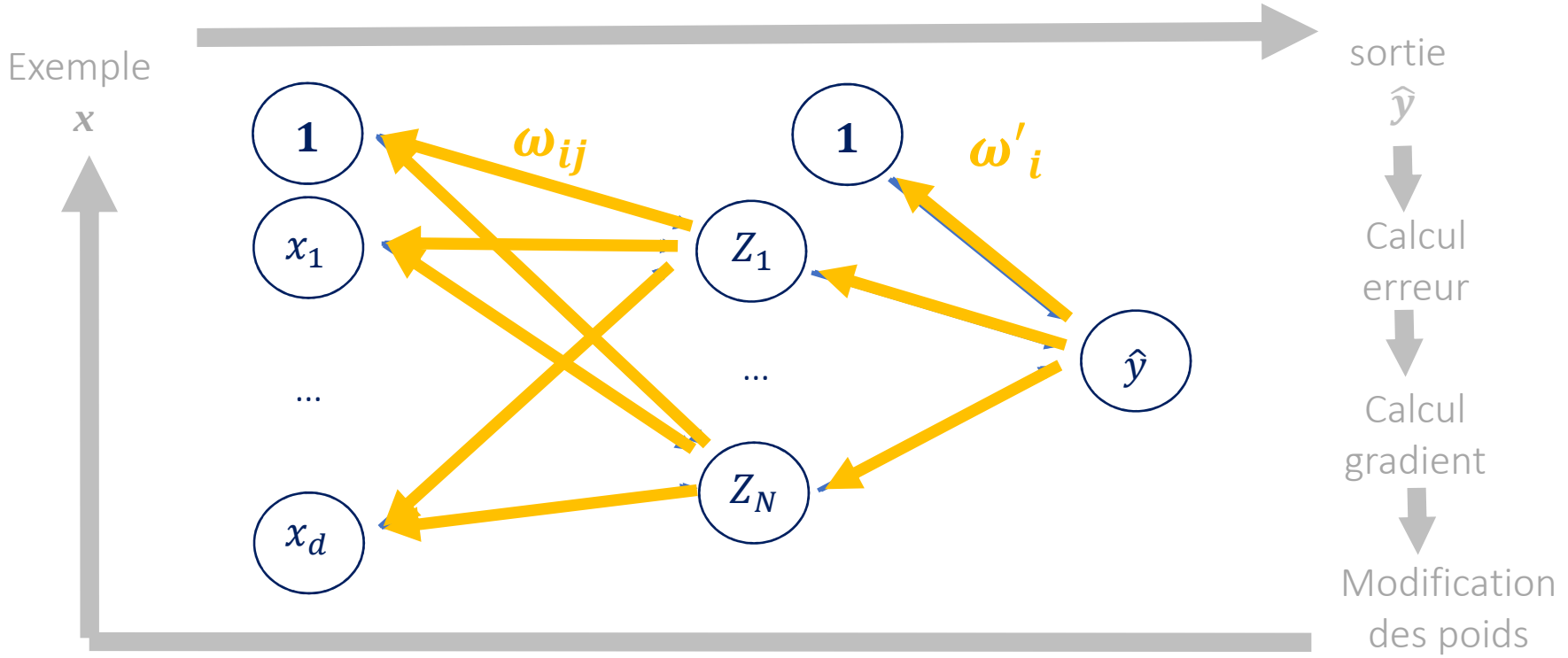
Ajustement du réseaux de neurones

Principe de la rétro-propagation du gradient

Les poids sont donc ajustés par une méthode de descente du gradient

$$\boldsymbol{\omega}^{(q+1)} = \boldsymbol{\omega}^{(q)} - \alpha \nabla E(\boldsymbol{\omega}^{(q)}).$$

Toute la difficulté réside dans le calcul du gradient. La méthode de rétro-propagation permet un calcul efficace de $\nabla E(\boldsymbol{\omega}^{(q)})$.

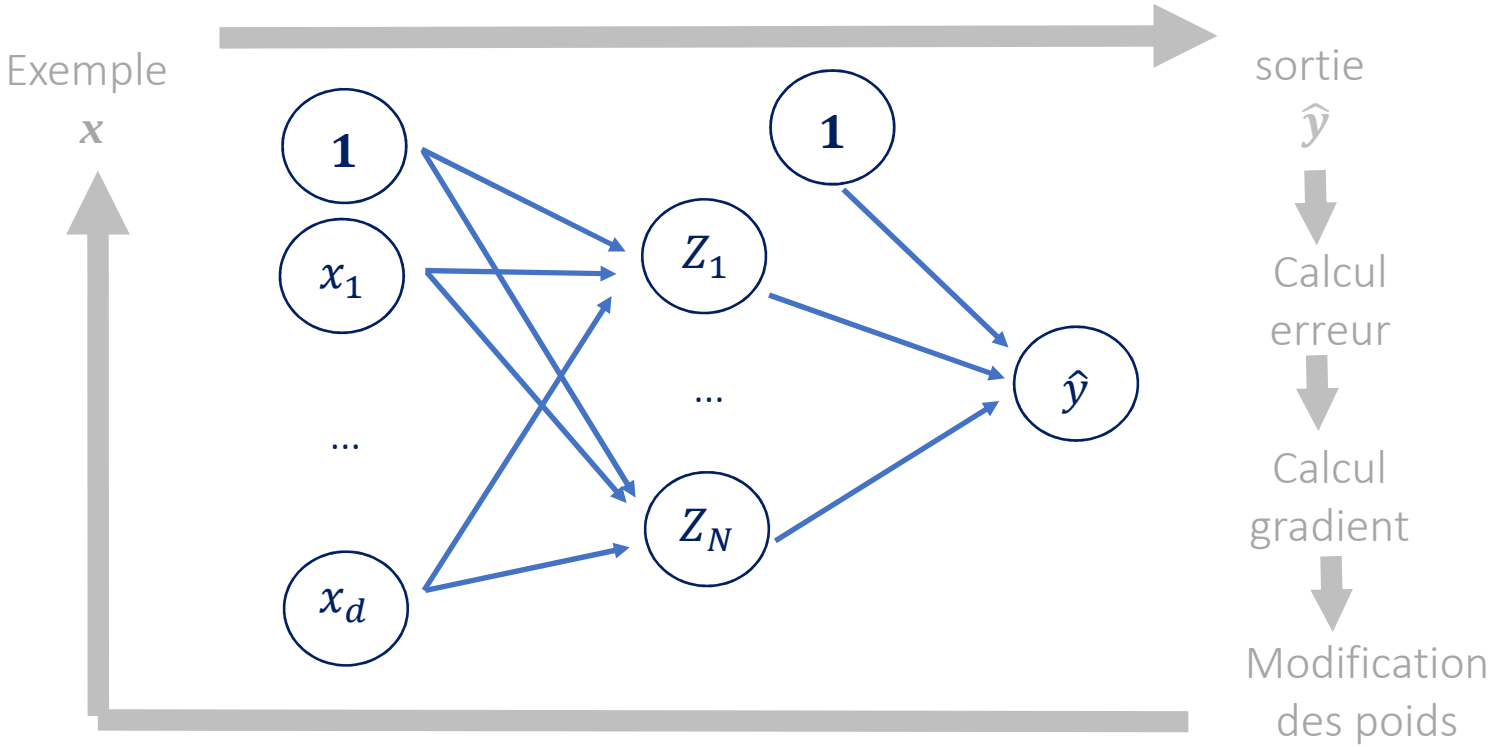


Tant que l'erreur>ε

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

Ajustement du réseaux de neurones

Principe de la rétro-propagation du gradient



Cycle (Epoch) = parcours entier de la base d'apprentissage

Plusieurs cycles sont nécessaires à la convergence

- Version « *online* » : ligne par ligne
- Version « *batch* » : toutes les lignes à la fois
- Version « *mini batch* » : par paquets de lignes

Ajustement du réseaux de neurones

Convergence de l'algorithme

Taux d'apprentissage

- α trop petit : convergence lente
- α trop grand : les poids oscillent et ne 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 grands [-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
- La convergence de l'algorithme dépend de l'initialisation aléatoire des poids

Problème des valeurs du gradient

Pour certaines fonctions de coût, le gradient risque de devenir

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

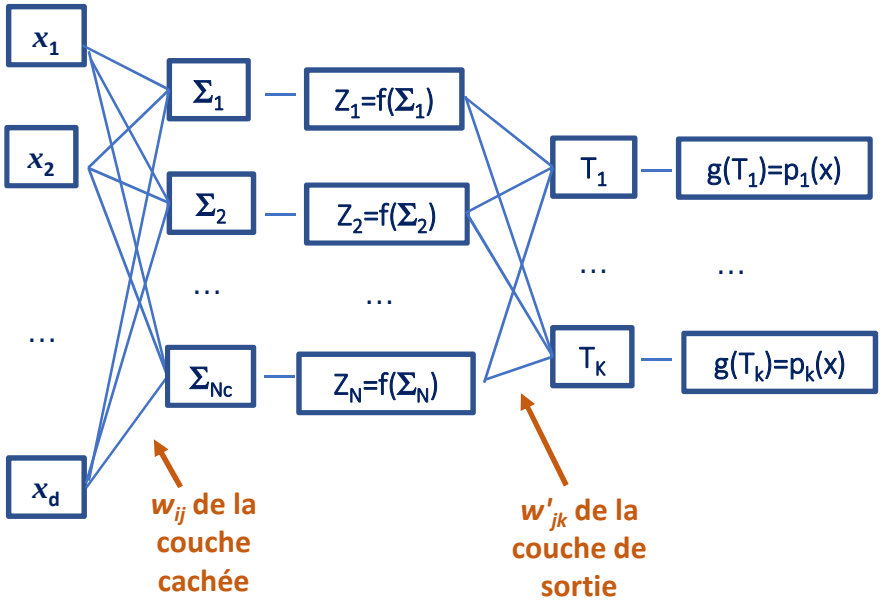
$$f(x) = x^+ = \max(0, x)$$

Ajustement du réseaux de neurones

Descente du gradient : cas perceptron multicouches

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

- ceux de la couche cachée, ω_{ij}, de l'entrée x_i vers le neurones Z_j
- ceux de la couche de sortie, ω'_{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
              Δω'_jk ← -αδ_k z_j
            • Pour chaque neurone caché j
              Calculer δ_j
              Δω_ij ← -αδ_j x_i
       Ajuster les poids :
              ω'_jk ← ω_jk+Δω_jk
              ω_ij ← ω_ij+Δω_ij
```

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

Son expression sera démontrée en cours

Ajustement du réseaux de neurones

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

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

La fonction de coût devient alors

$$E'(\boldsymbol{\omega}) = E(\boldsymbol{\omega}) + \tau\Omega$$

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 minimisent 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

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

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

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 (x,y). Notons y_k=1 si y∈ C_k et 0 sinon

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

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

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

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

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

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