

Perceptron multicouche

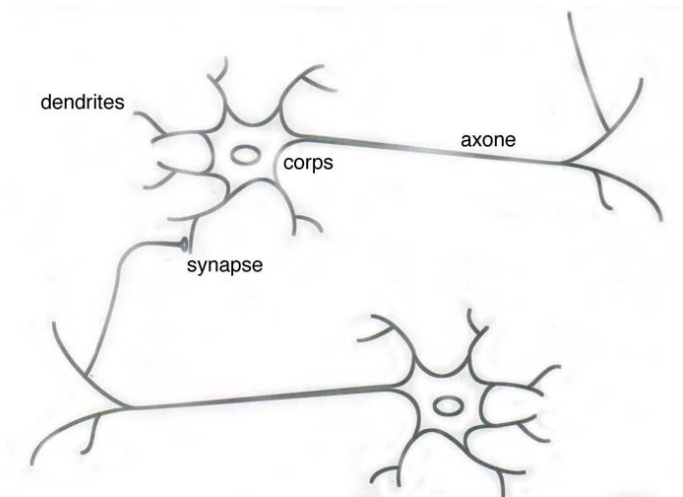
Apprentissage et reconnaissance – GIF-4101 / GIF-7005
Professeur : Christian Gagné



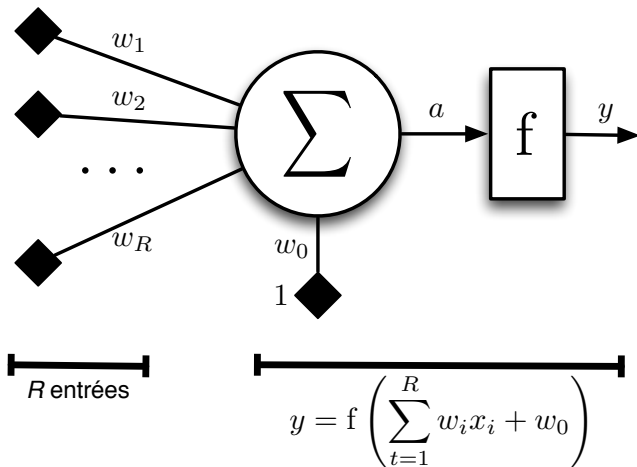
Semaine 12 : 23 novembre 2016

- Cerveau : siège de l'intelligence naturelle
 - ▶ Calculs parallèles et distribués
 - ▶ Apprentissage et généralisation
 - ▶ Adaption et contexte
 - ▶ Tolérant aux fautes
 - ▶ Faible consommation d'énergie
- Machine computationnelle biologique !

Neurone biologique



Modèle de neurone artificiel



- Chaque neurone est un discriminant linéaire avec une fonction de transfert f

$$y = f \left(\sum_i w_i x_i + w_0 \right) = f(\mathbf{w}^T \mathbf{x} + w_0)$$

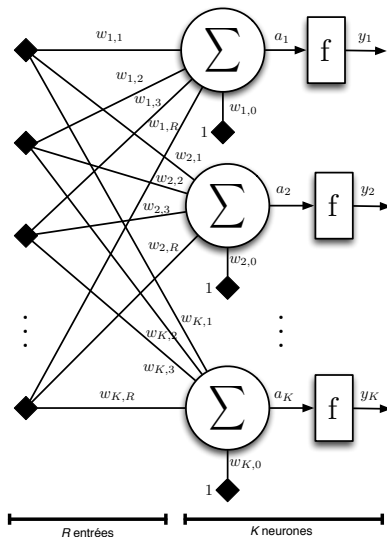
- Exemples de fonctions de transfert

- ▶ Fonction linéaire : $f_{lin}(a) = a$
- ▶ Fonction sigmoïde : $f_{sig}(a) = \frac{1}{1+\exp(-a)}$
- ▶ Fonction seuil : $f_{seuil}(a) = 1$ si $a \geq 0$ et $f_{seuil}(a) = 0$ autrement

- Plusieurs neurones connectés ensemble forment un réseau de neurones

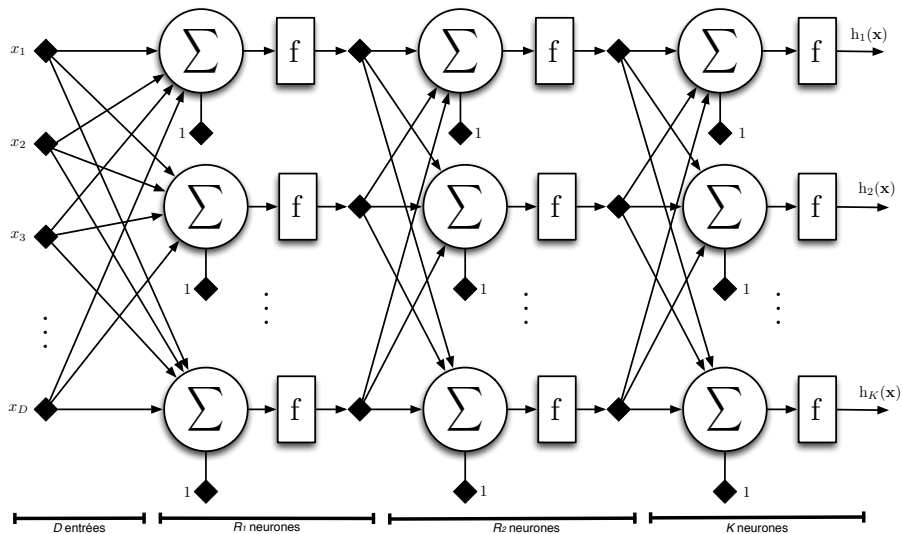
- ▶ Réseau à une couche : neurones connectés sur les entrées
- ▶ Réseau à plusieurs couches : certains neurones sont connectés sur les sorties d'autres neurones

Réseau de neurones (une couche)



- Réseau à une couche : ensemble de discriminants linéaires
 - ▶ Incapable de classer correctement des données non linéairement séparables
- Réseau à plusieurs couches (perceptron multicouche)
 - ▶ Discriminants linéaires (neurones) cascadés à la sortie d'autres discriminants linéaires
 - ▶ Capable de classer des données non linéairement séparables
 - ▶ Ensemble de classifieurs simples
 - ▶ Chaque couche fait une projection dans un nouvel espace
- Lors du traitement de données, l'information se propage des entrées vers les sorties

Perceptron multicouche



Problème du XOR

- Problème du XOR

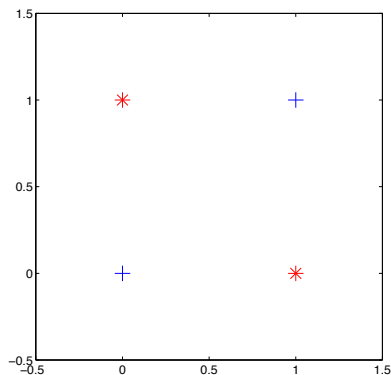
$$\mathbf{x}_1 = [0 \ 0]^T \quad r_1 = 0$$

$$\mathbf{x}_2 = [0 \ 1]^T \quad r_2 = 1$$

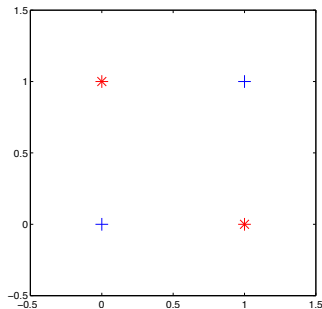
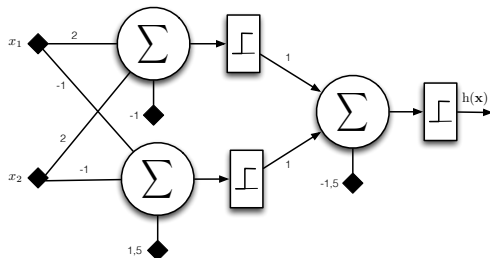
$$\mathbf{x}_3 = [1 \ 0]^T \quad r_3 = 1$$

$$\mathbf{x}_4 = [1 \ 1]^T \quad r_4 = 0$$

- Exemple de données non linéairement séparables

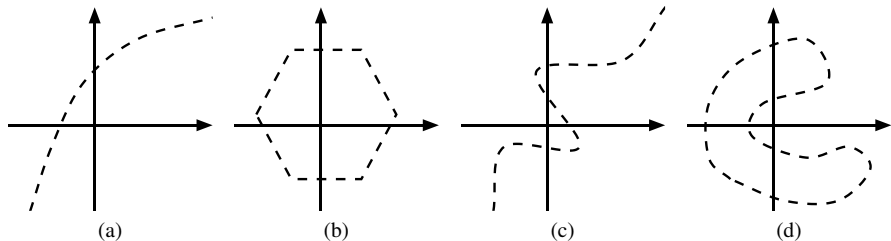


Réseau pour le problème du XOR



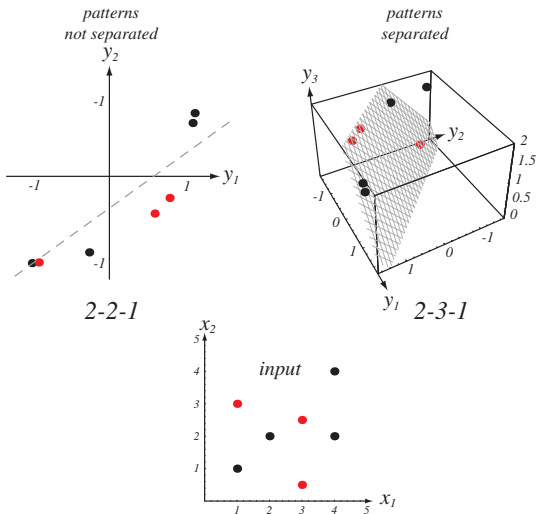
- Selon la topologie de réseau utilisé, différentes frontières de décisions sont possibles
 - ▶ Réseau avec une couche cachée et une couche de sortie : frontières convexes
 - ▶ Deux couches cachées ou plus : frontières concaves
 - ★ Le réseau de neurones est alors un approximateur universel
- Nombre de poids (donc de neurones) détermine directement la complexité du classifieur
 - ▶ Détermination de la bonne topologie est souvent *ad hoc*, par essais et erreurs

Formes de frontières de décision



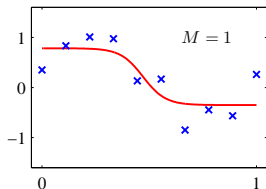
Exemples de frontières de décision : (a) convexe ouverte ; (b) convexe fermée ; (c) concave ouverte ; et (d) concave fermée

Nombre de neurones sur la couche cachée

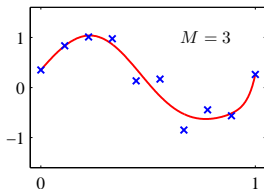


Tiré de R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, Wiley Interscience, 2001.

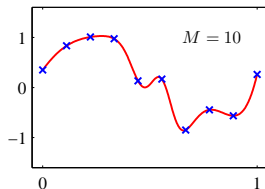
Nombre de neurones sur la couche cachée



1-1-1



1-3-1

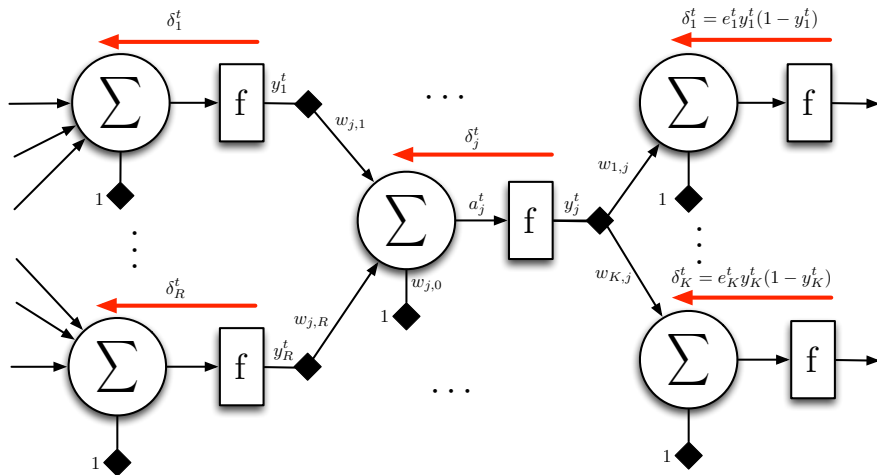


1-10-1

Tiré de C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

- Apprentissage avec le perceptron multicouche : déterminer les poids \mathbf{w}, w_0 de tous les neurones
- Rétropropagation des erreurs
 - ▶ Apprentissage par descente du gradient
 - ▶ Couche de sortie : correction guidée par l'erreur entre les sorties désirées et obtenues
 - ▶ Couches cachées : correction selon les sensibilités (influence du neurone sur l'erreur dans la couche de sortie)

Rétropropagation des erreurs



- Valeur y_j^t du neurone j pour la donnée \mathbf{x}^t

$$y_j^t = f(a_j^t) = f\left(\sum_{i=1}^R w_{j,i} y_i^t + w_{j,0}\right)$$

- ▶ f : fonction d'activation du neurone
- ▶ $a_j^t = \sum_{i=1}^R w_{j,i} y_i^t + w_{j,0}$: sommation pondérée des entrées du neurone
- ▶ $w_{j,i}$: poids du lien connectant le neurone j au neurone i de la couche précédente
- ▶ $w_{j,0}$: biais du neurone j
- ▶ y_i^t : sortie du neurone i de la couche précédente pour la donnée \mathbf{x}^t
- ▶ R : nombre de neurones sur la couche précédente

Erreur de la couche de sortie

- Un ensemble de données $\mathcal{X} = \{\mathbf{x}^t, \mathbf{r}^t\}_{t=1}^N$, avec $\mathbf{r}^t = [r_1^t \ r_2^t \ \dots \ r_K^t]^T$, où $r_j^t = 1$ si $\mathbf{x}^t \in C_j$, autrement $r_j^t = 0$
- Erreur observée pour donnée \mathbf{x}^t sur neurone j de la couche de sortie

$$e_j^t = r_j^t - y_j^t$$

- Erreur quadratique observée pour donnée \mathbf{x}^t sur les K neurones de la couche de sortie (un neurone par classe)

$$E^t = \frac{1}{2} \sum_{j=1}^K (e_j^t)^2$$

- Erreur quadratique moyenne observée pour les données du jeu \mathcal{X}

$$E = \frac{1}{N} \sum_{t=1}^N E^t$$

Correction de l'erreur pour la couche de sortie

- Correction des poids par descente du gradient de l'erreur quadratique moyenne

$$\Delta w_{j,i} = -\eta \frac{\partial E}{\partial w_{j,i}} = -\frac{\eta}{N} \sum_{t=1}^N \frac{\partial E^t}{\partial w_{j,i}}$$

- L'erreur du neurone j dépend des neurones de la couche précédente
 - ▶ Développement en utilisant la règle du chaînage des dérivées
($\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \frac{\partial y}{\partial x}$)

$$\begin{aligned} \frac{\partial E^t}{\partial w_{j,i}} &= \frac{\partial E^t}{\partial e_j^t} \frac{\partial e_j^t}{\partial y_j^t} \frac{\partial y_j^t}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{j,i}} \\ \frac{\partial E^t}{\partial w_{j,0}} &= \frac{\partial E^t}{\partial e_j^t} \frac{\partial e_j^t}{\partial y_j^t} \frac{\partial y_j^t}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{j,0}} \end{aligned}$$

Calcul des dérivées partielles

- Développement avec fonction d'activation sigmoïde ($y_j^t = \frac{1}{1+\exp(-a_j^t)}$)

$$\frac{\partial E^t}{\partial e_j^t} = \frac{\partial}{\partial e_j^t} \frac{1}{2} \sum_{l=1}^K (e_l^t)^2 = e_j^t$$

$$\frac{\partial e_j^t}{\partial y_j^t} = \frac{\partial}{\partial y_j^t} r_j^t - y_j^t = -1$$

$$\begin{aligned} \frac{\partial y_j^t}{\partial a_j^t} &= \frac{\partial}{\partial a_j^t} \frac{1}{1 + \exp(-a_j^t)} = \frac{\exp(-a_j^t)}{[1 + \exp(-a_j^t)]^2} \\ &= \frac{1}{1 + \exp(-a_j^t)} \frac{\exp(-a_j^t) + 1 - 1}{1 + \exp(-a_j^t)} = y_j^t(1 - y_j^t) \end{aligned}$$

$$\frac{\partial a_j^t}{\partial w_{j,i}} = \frac{\partial}{\partial w_{j,i}} \sum_{l=1}^R w_{j,l} y_l^t + w_{j,0} = y_i^t$$

$$\frac{\partial a_j^t}{\partial w_{j,0}} = \frac{\partial}{\partial w_{j,0}} \sum_{l=1}^R w_{j,l} y_l^t + w_{j,0} = 1$$

Apprentissage pour la couche de sortie

- Apprentissage des poids de la couche de sortie

$$\begin{aligned}\Delta w_{j,i} &= -\frac{\eta}{N} \sum_{t=1}^N \frac{\partial E^t}{\partial w_{j,i}} = -\frac{\eta}{N} \sum_{t=1}^N \frac{\partial E^t}{\partial e_j^t} \frac{\partial e_j^t}{\partial y_j^t} \frac{\partial y_j^t}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{j,i}} \\ &= \frac{\eta}{N} \sum_{t=1}^N e_j^t y_j^t (1 - y_j^t) y_i^t\end{aligned}$$

- Apprentissage des biais de la couche de sortie

$$\begin{aligned}\Delta w_{j,0} &= -\frac{\eta}{N} \sum_{t=1}^N \frac{\partial E^t}{\partial w_{j,0}} = -\frac{\eta}{N} \sum_{t=1}^N \frac{\partial E^t}{\partial e_j^t} \frac{\partial e_j^t}{\partial y_j^t} \frac{\partial y_j^t}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{j,0}} \\ &= \frac{\eta}{N} \sum_{t=1}^N e_j^t y_j^t (1 - y_j^t)\end{aligned}$$

Règle du delta

- Poser un delta δ_j^t , qui correspond au *gradient local* du neurone j pour la donnée \mathbf{x}^t

$$\delta_j^t = e_j^t y_j^t (1 - y_j^t)$$

$$\Delta w_{j,i} = \frac{\eta}{N} \sum_{t=1}^N \delta_j^t y_i^t$$

$$\Delta w_{j,0} = \frac{\eta}{N} \sum_{t=1}^N \delta_j^t$$

- Formulation utile pour correction de l'erreur sur les couches cachées

Correction de l'erreur pour les couches cachées

- Gradient de l'erreur pour les couches cachées

$$\frac{\partial E^t}{\partial w_{j,i}} = \frac{\partial E^t}{\partial y_j^t} \frac{\partial y_j^t}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{j,i}}$$

- Seul $\frac{\partial E^t}{\partial y_j^t}$ change, $\frac{\partial y_j^t}{\partial a_j^t}$ et $\frac{\partial a_j^t}{\partial w_{j,i}}$ sont les mêmes que sur la couche de sortie
 - ▶ Erreur pour un neurone de la couche cachée dépend de l'erreur des neurones k de la couche suivante (rétropropagation des erreurs)

$$E^t = \frac{1}{2} \sum_k (e_k^t)^2$$

$$\frac{\partial E^t}{\partial y_j^t} = \frac{\partial}{\partial y_j^t} \frac{1}{2} \sum_k (e_k^t)^2 = \sum_k e_k^t \frac{\partial e_k^t}{\partial y_j^t}$$

Correction de l'erreur pour les couches cachées

$$\begin{aligned}\frac{\partial E^t}{\partial y_j^t} &= \frac{\partial}{\partial y_j^t} \frac{1}{2} \sum_k (e_k^t)^2 = \sum_k e_k^t \frac{\partial e_k^t}{\partial y_j^t} \\&= \sum_k e_k^t \frac{\partial e_k^t}{\partial a_k^t} \frac{\partial a_k^t}{\partial y_j^t} \\&= \sum_k e_k^t \frac{\partial (r_k^t - y_k^t)}{\partial a_k^t} \frac{\partial (\sum_l w_{k,l} y_l^t + w_{k,0})}{\partial y_j^t} \\&= \sum_k e_k^t [-y_k^t (1 - y_k^t)] w_{k,j} \\ \delta_k^t &= e_k^t [y_k^t (1 - y_k^t)] \\ \frac{\partial E^t}{\partial y_j^t} &= - \sum_k \delta_k^t w_{k,j}\end{aligned}$$

Correction de l'erreur pour les couches cachées

- Correction de l'erreur correspondante

$$\frac{\partial E^t}{\partial w_{j,i}} = \frac{\partial E^t}{\partial y_j^t} \frac{\partial y_j^t}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{j,i}}$$

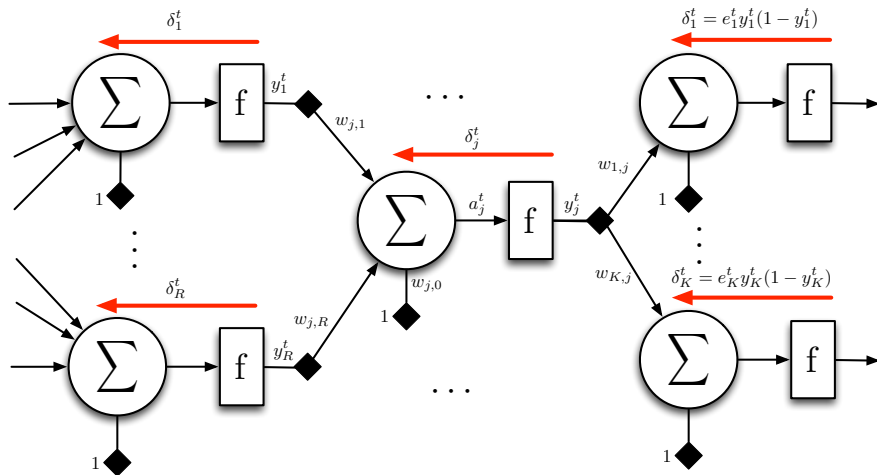
$$= - \left[\sum_k \delta_k^t w_{k,j} \right] y_j^t (1 - y_j^t) y_i^t$$

$$\delta_j^t = y_j^t (1 - y_j^t) \sum_k \delta_k^t w_{k,j}$$

$$\Delta w_{j,i} = -\eta \frac{\partial E}{\partial w_{j,i}} = -\frac{\eta}{N} \sum_{t=1}^N \frac{\partial E^t}{\partial w_{j,i}} = \frac{\eta}{N} \sum_{t=1}^N \delta_j^t y_i^t$$

$$\Delta w_{j,0} = -\eta \frac{\partial E}{\partial w_{j,0}} = -\frac{\eta}{N} \sum_{t=1}^N \frac{\partial E^t}{\partial w_{j,0}} = \frac{\eta}{N} \sum_{t=1}^N \delta_j^t$$

Rétropropagation des erreurs



- Apprentissage par lots

- ▶ Guidé par l'erreur quadratique moyenne ($E = \frac{1}{N} \sum_t E^t$)
- ▶ Correction des poids une fois à chaque époque, en calculant l'erreur pour tout le jeu de données
- ▶ Relative stabilité de l'apprentissage

- Apprentissage en ligne

- ▶ Correction des poids pour chaque présentation de données, donc N corrections de poids par époque
- ▶ Guidé par l'erreur quadratique de chaque donnée (E^t)
- ▶ Requiert la permutation de l'ordre de traitement à chaque époque pour éviter les mauvaises séquences
- ▶ Apprentissage en ligne est plus rapide que par lots, mais avec risque de plus grandes instabilités

Saturation des neurones

- Plage opératoire des neurones avec fonction sigmoïde autour de 0
 - ▶ Pour valeurs de a faibles $f_{sig}(a) \rightarrow 0$, et pour valeurs de a élevée, $f_{sig}(a) \rightarrow 1$

$$f_{sig}(1) = 0,7311, \quad f_{sig}(5) = 0,9933, \quad f_{sig}(10) \approx 1$$

- Pour valeurs grandes/petites, disons $x < -10$ ou $x > 10$, gradient pratiquement nul
 - ▶ Apprentissage extrêmement lent
- Valeurs d'entrées, les \mathbf{x}^t , doivent être normalisées au préalable dans $[-1, 1]$
 - ▶ Typiquement, normalisation selon valeurs min et max du jeu de données pour chaque dimension
 - ▶ Appliquer la même normalisation aux données évaluées (ne pas recalculer la normalisation)

- En classement, valeurs désirées $r_i^t \in \{0, 1\}$
 - ▶ Souffre également du problème de saturation des neurones avec fonction sigmoïde
 - ▶ On vise à approximer les r_i^t avec les neurones de la couche de sortie

$$f_{sig}(a) = 0 \Rightarrow a \rightarrow -\infty, \quad f_{sig}(a) = 1 \Rightarrow a \rightarrow \infty$$

- Solution : transformer les valeurs désirées en valeurs $\tilde{r}_i^t \in \{0,05, 0,95\}$
 - ▶ Si $\mathbf{x}^t \in C_i$ alors $\tilde{r}_i^t = 0,95$
 - ▶ Autrement $\tilde{r}_i^t = 0,05$

- Les poids et biais d'un perceptron multicouche sont initialisés aléatoirement
 - ▶ Typiquement, on initialise les poids et biais uniformément dans $[-0,5, 0,5]$

$$w_{j,i} \sim \mathcal{U}(-0,5, 0,5), \forall i,j$$

- Perceptron multicouche est donc un algorithme stochastique
 - ▶ D'une exécution à l'autre, on n'obtient pas nécessairement les mêmes résultats

Algorithme de rétropropagation

- ➊ Normaliser données d'entraînement $x_j^t \in [-1,1]$ et sortie désirées $\tilde{r}_j^t \in \{0,05, 0,95\}$
- ➋ Initialiser les poids et biais aléatoirement, $w_{i,j} \in [-0,5, 0,5]$
- ➌ Tant que le critère d'arrêt n'est pas atteint, répéter :
 - ➊ Calculer les sorties observées en propageant les données vers l'avant
 - ➋ Calculer les erreurs observées sur la couche de sortie

$$e_j^t = \tilde{r}_j^t - y_j^t, \quad j = 1, \dots, K, \quad t = 1, \dots, N$$

- ➍ Ajuster les poids et biais en rétropropageant l'erreur observée

$$w_{j,i} = w_{j,i} + \Delta w_{j,i} = w_{j,i} + \frac{\eta}{N} \sum_t \delta_j^t y_i^t$$

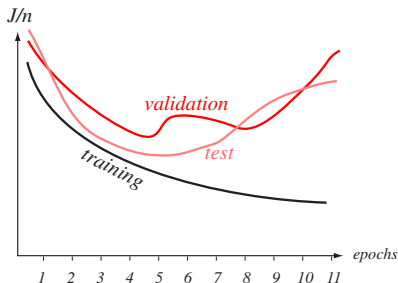
$$w_{j,0} = w_{j,0} + \Delta w_{j,0} = w_{j,0} + \frac{\eta}{N} \sum_t \delta_j^t$$

où le gradient local est défini par :

$$\delta_j^t = \begin{cases} e_j^t y_j^t (1 - y_j^t) & \text{si } j \in \text{couche de sortie} \\ y_j^t (1 - y_j^t) \sum_k \delta_k^t w_{k,j} & \text{si } j \in \text{couche cachée} \end{cases}$$

Surapprentissage et critère d'arrêt

- Nombre d'époques : facteur déterminant pour le surapprentissage
- Critère d'arrêt : lorsque l'erreur sur l'ensemble de validation augmente (généralisation)
- Requiert utilisation d'une partie des données de l'ensemble pour la validation



Tiré de R.O. Duda, P.E. Hart, D.G. Stork,
Pattern Classification, Wiley Interscience, 2001.

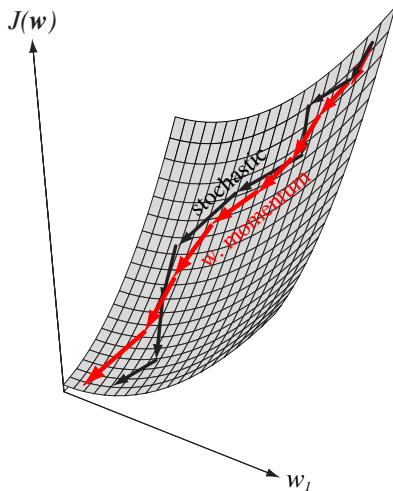
- Règle du delta généralisée

$$w_{j,i}(n) = w_{j,i}(n-1) + \frac{\eta}{N} \sum_t \delta_j^t y_i^t + \alpha \Delta w_{j,i}(n-1)$$

$$w_{j,0}(n) = w_{j,0}(n-1) + \frac{\eta}{N} \sum_t \delta_j^t + \alpha \Delta w_{j,0}(n-1)$$

- Facteur $\Delta w_{j,i}(n-1)$ est la correction effectuée au poids/biais à l'époque précédente
- Paramètre $\alpha \in [0,5,1]$ est nommé *momentum*
- Donne une « inertie » à la descente du gradient, en incluant une correction provenant des itérations précédentes
- Avec momentum, le facteur $\Delta w_{j,i}(n-1)$ dépend lui-même de la correction de l'itération précédente $\Delta w_{j,i}(n-2)$, et ainsi de suite

Momentum



Tiré de R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, Wiley Interscience, 2001.

- Algorithme de rétropropagation développé ici pour fonction de transfert sigmoïde, pour le classement
 - ▶ D'autres fonctions de transfert peuvent être utilisées
 - ★ Fonction linéaire : $f_{lin}(a) = a$
 - ★ Fonction tangente hyperbolique : $f_{tanh}(a) = \tanh(a)$
 - ★ Fonction ReLU (*rectified linear unit*) : $f_{ReLU}(a) = \max(0, a)$
 - ▶ En fait, toutes fonctions continues dérivables sur \mathbb{R} peuvent être utilisées
- Perceptron multicouche approprié pour de la régression
 - ▶ Topologie conseillée : une couche cachée avec fonction sigmoïde et une couche de sortie avec fonction linéaire
 - ▶ Critère de l'erreur quadratique moyenne approprié pour la régression

Méthode du deuxième ordre

- La descente du gradient est une méthode du premier ordre (dérivées premières)
- Possibilité de faire mieux avec des méthodes du deuxième ordre
- Méthode de Newton
 - ▶ Basée sur l'expansion de la série de Taylor du deuxième ordre, $\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$ un point dans le voisinage de \mathbf{x}

$$F(\mathbf{x}') = F(\mathbf{x} + \Delta\mathbf{x}) \approx F(\mathbf{x}) + \nabla F(\mathbf{x})^T \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \nabla^2 F(\mathbf{x}) \Delta\mathbf{x} = \hat{F}(\mathbf{x})$$

- ▶ Recherche un plateau dans l'erreur quadratique $\hat{F}(\mathbf{x})$

$$\begin{aligned} \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{x}} &= \nabla F(\mathbf{x}) + \nabla^2 F(\mathbf{x}) \Delta\mathbf{x} = 0 \\ \Delta\mathbf{x} &= -(\nabla^2 F(\mathbf{x}))^{-1} \nabla F(\mathbf{x}) \end{aligned}$$

- ▶ Calcul de l'inverse de la matrice Hessienne $((\nabla^2 F(\mathbf{x}))^{-1})$ coûteux en calculs
- ▶ Méthode du gradient conjugué évite le calcul de l'inverse de la matrice Hessienne

- Fonctions de base radiale (RBF : *Radial Basis Functions*)

$$\phi_i(\mathbf{x}) = \exp \left[-\frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{2s_i^2} \right]$$

- Consiste en une fonction gaussienne centrée sur \mathbf{m}_i avec une influence locale paramétrée par s_i
 - ▶ À strictement parler, ce n'est pas une densité de probabilité de loi multinormale ($\int_{-\infty}^{\infty} \phi_i(\mathbf{x}) d\mathbf{x} \neq 1$)
- Idée : chaque fonction gaussienne capture un groupe de données dans un certain voisinage
- Avec R fonctions gaussiennes, projection dans un espace à R dimensions

$$\phi = [\phi_1 \ \dots \ \phi_R]^T : \mathbb{R}^D \rightarrow \mathbb{R}^R$$

Discrimination avec fonctions gaussiennes

- Discrimination avec R fonctions gaussiennes (K classes)

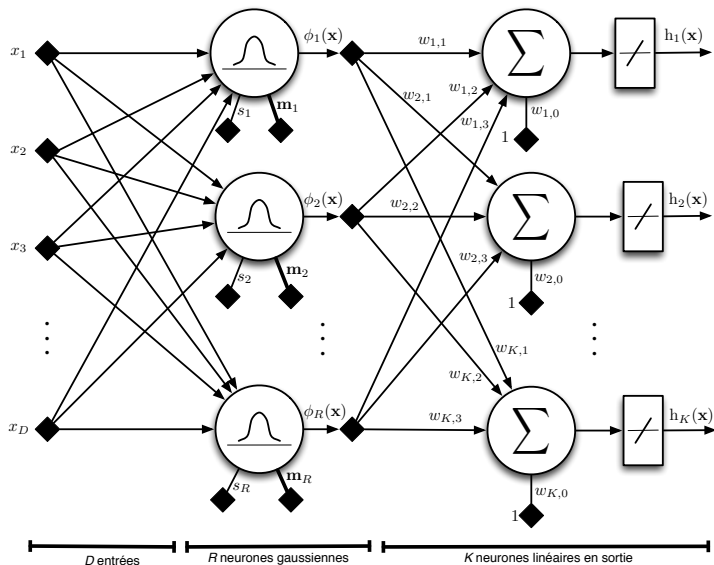
$$h_j(\mathbf{x}) = \sum_{i=1}^R w_{j,i} \phi_i(\mathbf{x}) + w_{j,0} = \sum_{i=1}^R w_{j,i} \exp \left[-\frac{\|\mathbf{x} - \mathbf{m}_i\|^2}{2s_i^2} \right] + w_{j,0}$$

- Paramètres du discriminant à estimer

- ▶ Positions \mathbf{m}_i des fonctions gaussiennes
- ▶ Étalement s_i des fonctions
 - ★ Fréquent de le partager entre les fonctions gaussiennes, $s_i = s, \forall i$
- ▶ Poids $w_{j,i}$ des fonctions gaussiennes
 - ★ Poids $w_{j,i}$ lie la j -ième classe à la i -ième fonction gaussienne
 - ★ Peut être fixé à des constantes, $w_i = \pm 1$, selon l'association entre fonctions gaussiennes et classes
- ▶ Biais $w_{j,0}$ des sorties
 - ★ Avec poids égaux, ex. $w_{j,i} = \pm 1$, biais peut être nul, $w_{j,0} = 0$

- Réseau RBF peut être vu comme un cas particulier d'un perceptron multicouche
 - ▶ Une couche cachée avec fonction gaussienne
 - ▶ Couche de sortie avec fonction linéaire
- Développement des équations pour mettre à jour les \mathbf{m}_i , \mathbf{w}_j , $w_{j,0}$ et même s_i avec descente du gradient est une instance de l'algorithme de rétropropagation des erreurs
 - ▶ Corrige d'abord les poids \mathbf{w}_j et $w_{j,0}$ sur la couche de sortie
 - ▶ Corrige ensuite les positions des centres \mathbf{m}_i et étalements s_i

Réseau RBF comme réseau de neurones



Apprentissage avec descente du gradient (\mathbf{w}_j et $w_{j,0}$)

- Critère d'erreur quadratique

$$E(\mathbf{w}, w_0 | \mathcal{X}) = \frac{1}{N} \sum_{t=1}^N \frac{1}{2} \sum_{j=1}^K (e_j^t)^2 = \frac{1}{2N} \sum_{t=1}^N \sum_{j=1}^K \left[r_j^t - \left(\sum_{i=1}^R w_{j,i} \phi_i(\mathbf{x}^t) + w_{j,0} \right) \right]^2$$

- Dérivée partielle pour \mathbf{w}_i et $w_{i,0}$

$$\frac{\partial E}{\partial w_{j,i}} = -\frac{1}{N} \sum_{t=1}^N \left[r_j^t - \left(\sum_{i=1}^R w_{j,i} \phi_i(\mathbf{x}^t) + w_{j,0} \right) \right] \phi_i(\mathbf{x}^t) = -\frac{1}{N} \sum_{t=1}^N e_j^t \phi_i(\mathbf{x}^t)$$

$$\frac{\partial E}{\partial w_{j,0}} = -\frac{1}{N} \sum_{t=1}^N \left[r_j^t - \left(\sum_{i=1}^R w_{j,i} \phi_i(\mathbf{x}^t) + w_{j,0} \right) \right] = -\frac{1}{N} \sum_{t=1}^N e_j^t$$

- Descente du gradient pour $w_{j,i} = w_{j,i} + \Delta w_{j,i}$, $i = 0, \dots, K$

$$\Delta w_{j,i} = -\eta \frac{\partial E}{\partial w_{j,i}} = \frac{\eta}{N} \sum_{t=1}^N e_j^t \phi_i(\mathbf{x}^t), \quad \Delta w_{j,0} = -\eta \frac{\partial E}{\partial w_{j,0}} = \frac{\eta}{N} \sum_{t=1}^N e_j^t$$

Apprentissage avec descente du gradient (\mathbf{m}_i)

- Dérivée partielle pour $\mathbf{m}_i = [m_{i,1} \ m_{i,2} \ \cdots \ m_{i,D}]^T$

$$\begin{aligned}\frac{\partial \phi_i(\mathbf{x}^t)}{\partial m_{i,k}} &= \frac{\partial \exp \left[-\frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{2s_i^2} \right]}{\partial m_{i,k}} = \frac{(x_k^t - m_{i,k})}{s_i^2} \exp \left[-\frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{2s_i^2} \right] \\ &= \frac{(x_k^t - m_{i,k})}{s_i^2} \phi_i(\mathbf{x}^t)\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial m_{i,k}} &= -\frac{1}{N} \sum_{t=1}^N \sum_{j=1}^K w_{j,i} \left[r_j^t - \left(\sum_{i=1}^R w_{j,i} \phi_i(\mathbf{x}^t) + w_{j,0} \right) \right] \frac{\partial \phi_i(\mathbf{x}^t)}{\partial m_{i,k}} \\ &= -\frac{1}{N} \sum_{t=1}^N \sum_{j=1}^K e_j^t w_{j,i} \frac{(x_k^t - m_{i,k})}{s_i^2} \phi_i(\mathbf{x}^t)\end{aligned}$$

- Apprentissage : $m_{i,k} = m_{i,k} + \Delta m_{i,k}$, $i = 1, \dots, R$, $k = 1, \dots, D$

$$\Delta m_{i,k} = -\eta \frac{\partial E}{\partial m_{i,k}} = \frac{\eta}{N} \sum_{t=1}^N \sum_{j=1}^K e_j^t w_{j,i} \frac{(x_k^t - m_{i,k})}{s_i^2} \phi_i(\mathbf{x}^t)$$

Apprentissage avec descente du gradient (s_i)

- Dérivée partielle pour s_i

$$\begin{aligned}\frac{\partial \phi_i(\mathbf{x}^t)}{\partial s_i} &= \frac{\partial \exp \left[-\frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{2s_i^2} \right]}{\partial s_i} = 2 \frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{2s_i^3} \exp \left[-\frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{2s_i^2} \right] \\ &= \frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{s_i^3} \phi_i(\mathbf{x}^t) \\ \frac{\partial E}{\partial s_i} &= -\frac{1}{N} \sum_{t=1}^N \sum_{j=1}^K w_{j,i} \left[r_j^t - \left(\sum_{i=1}^R w_{j,i} \phi_i(\mathbf{x}^t) + w_{j,0} \right) \right] \frac{\partial \phi_i(\mathbf{x}^t)}{\partial s_i} \\ &= -\frac{1}{N} \sum_{t=1}^N \sum_{j=1}^K e_j^t w_{j,i} \frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{s_i^3} \phi_i(\mathbf{x}^t)\end{aligned}$$

- Apprentissage : $s_i = s_i + \Delta s_i$, $i = 1, \dots, R$

$$\Delta s_i = -\eta \frac{\partial E}{\partial s_i} = \frac{\eta}{N} \sum_{t=1}^N \sum_{j=1}^K e_j^t w_{j,i} \frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{s_i^3} \phi_i(\mathbf{x}^t)$$

- Apprentissage en bloc de réseau RBF
 - ▶ Apprentissage en bloc de \mathbf{w}_j , $w_{j,0}$, \mathbf{m}_j et s_j par descente du gradient peut être relativement lourd, computationnellement parlant
 - ▶ Convergence lente vers résultats satisfaisants
- Apprentissage hybride
 - ▶ Fixer $s_j = s$ et apprendre les positions \mathbf{m}_j par clustering (ex. K -means)
 - ▶ Ensuite, apprendre \mathbf{w}_j et $w_{j,0}$ par descente du gradient

- Depuis version 0.18 (fin septembre 2016), MLP est disponible dans scikit-learn
 - ▶ Plus nécessaire d'utiliser scikit-neuralnetwork pour les devoirs !
 - ▶ Scikit-learn utilise certaines avancées des réseaux profonds (mais pas toutes)
 - ▶ Pas d'accélération GPU pour les calculs, rigidité des modèles utilisables
- `neural_network.MLPClassifier` : perceptron multicouche pour le classement
 - ▶ Minimise entropie croisée pour du classement avec des méthodes basées sur le gradient

$$E_{entr} = - \sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t)$$

- `neural_network.MLPRegressor` : perceptron multicouche pour la régression
 - ▶ Minimise l'erreur quadratique avec des méthodes basées sur le gradient

Paramètres de MLPClassifier et MLPRegressor

- `hidden_layer_sizes` (tuple) : nombre de neurones sur chaque couche cachée (défaut : (100,))
- `activation` (string) : 'identity' (linéaire), 'logistic' (sigmoïde), 'tanh' et 'relu' (défaut : 'relu')
- `solver` (string) : 'lbfgs' (quasi-Newton), 'sgd' (descente du gradient stochastique), 'adam' (sgd avec détermination automatique du taux d'apprentissage) (défaut : 'adam')
- `alpha` (float) : paramètre de la régularisation L_2 des poids (défaut : 0,0001)
- `batch_size` (int) : taille des lots pour chaque mise à jour (défaut : $\min(200, N)$)
- `learning_rate_init` (float) : taux d'apprentissage initial (défaut : 0,001)
- `learning_rate` (string) : 'constant', 'invscaling' ($\text{learning_rate_init} / \text{pow}(t, \text{power_t})$), 'adaptive' (taux actuel réduit lorsque apprentissage stagne) (défaut : 'constant')
- `max_iter` (int) : nombre maximal d'époques (défaut : 200)
- `tol` (float) : tolérance, arrêt de l'apprentissage si gain < tolérance pour plus de deux époques (défaut : 10^{-4})
- `momentum` (float) : momentum pour la descente du gradient (défaut : 0,9)
- `early_stopping` (bool) : arrêt lorsque erreur sur ensemble de validation ne baisse plus (défaut : False)
- `validation_fraction` (float) : portion des données utilisées pour la validation avec l'*early stopping* (défaut : 0,1)