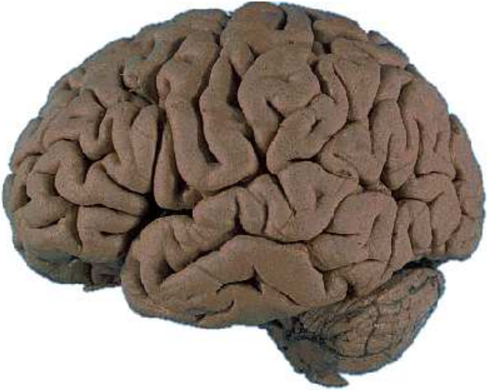
Apprentissage Automatique Numérique
Perceptron
Perceptron multi-couches

Loïc Barrault

Laboratoire d'Informatique de l'Université du Maine (LIUM)
loic.barrault@lium.univ-lemans.fr

14 novembre 2016

Le Cerveau Humain



Est-ce qu'on peut créer des machines intelligentes qui ont un fonctionnement similaire à celui de cerveau humain ?

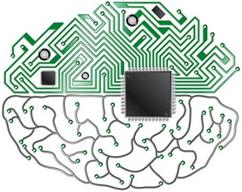
Le Cerveau Humain

Caractéristiques :

- 10 billions = 10.10^{12} cellules nerveuses (neurones)
- Chacune connectée à 10 000 autres via les synapses
- Faible dégradation en cas de dommages partiels
- Certaines tâches peuvent être reprises par d'autres zones
- Apprentissage à partir des expériences
- Calcul lent (100 Hz), mais massivement parallèle
  $\rightarrow$ très efficace

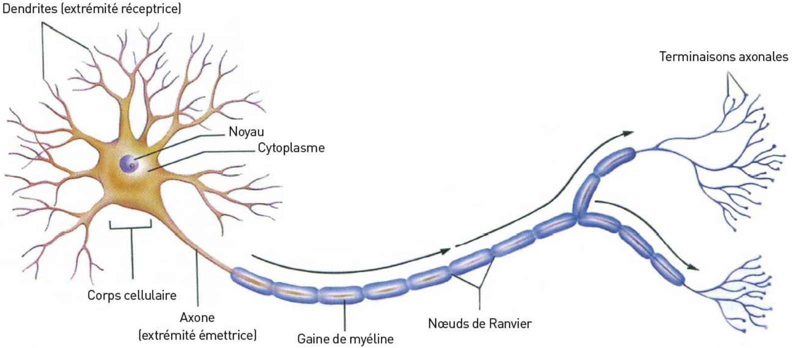
  Exemple : perception visuelle très complexe en 100ms
  (c-a-d en 10 opérations !)

Comparaison cerveau humain / ordinateur



	cerveau humain	ordinateur
élément de calcul	10^{14} synapses	$7 * 10^9$ transistors
taille d'un élément	10^{-6} m	3^{-9} m
besoin d'énergie	30 W	45-130 W (CPU)
vitesse de calcul	100Hz	3,5GHz
type de calcul	parallèle distribué	multi-core centralisé
tolérance aux fautes	oui	non
apprentissage	oui	un peu
conscience	normalement	pas encore

La Cellule Nerveuse



La Cellule Nerveuse

- Réception des stimulations des autres cellules via les synapses
- Ces stimulations sont additionnées
- Lorsque cette somme dépasse un seuil la cellule envoie une stimulation électrique le long de son axone (dépolarisation)
- Pendant une certaine période la cellule ne peut envoyer de nouvelles stimulations période de réfraction
- Les bouts de l'axone touchent presque le corps ou les dendrites d'autres cellules
- La transmission de l'impulsion électrique se fait par des neuro-transmetteurs
- La transmission dépend de la quantité de neuro-transmetteurs disponibles, du nombre et de l'arrangement des synapses, de l'absorption des neuro-transmetteurs par des récepteurs, ...

Apprentissage dans le cerveau humain

Principes :

- Modification de la force des connexions
- Ajout ou suppression de connexions
- Aucune supervision n'est nécessaire
- ...

Principe de Hebb

Si un axone de la cellule A excite la cellule B de façon répétitive ou persistante, un processus de croissance ou des changements métaboliques se mettent en place de sorte que la stimulation de la cellule B par la cellule A augmente.

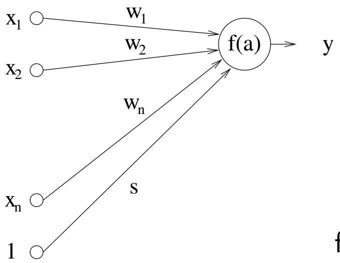
Hebb: "Neurons that fire together, wire together"

- Cellules actives en même temps
  $\rightarrow$ renforcer les connexions
- Cellules pas actives en même temps
  $\rightarrow$ affaiblir les connexions

$\Rightarrow$ Processus local il n'y a pas de supervision globale

Le perceptron

Petite unité de calcul vaguement inspirée par le fonctionnement supposé du cerveau humain

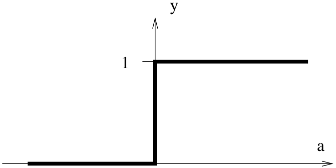


entrées:	x_i
poids:	w_i
seuil:	s
activité:	$a = \sum_i w_i x_i + s$
sortie:	$y = f(a)$
fonction d'activité:	$f = seuil(a)$

une entrée imaginaire qui vaut toujours 1 permet d'ajouter le seuil s au vecteur de poids $\Rightarrow$ facilite la notation.

Le Perceptron et les fonctions logiques

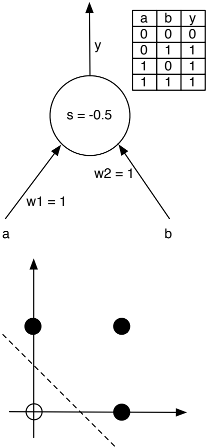
- Les fonctions logiques :
  - entrées: binaires 0 ou 1
  - sortie: binaire 0 ou 1
- fonction d'activité du perceptron: f=seuil(a)
- : ex. fonction seuil



$\Rightarrow$ on peut facilement déterminer les poids w_i et le seuil s pour réaliser des fonctions logiques OU, ET et NON

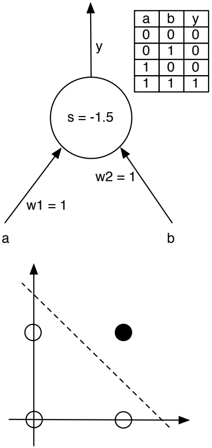
Le Perceptron et les fonctions logiques

y = a OU b



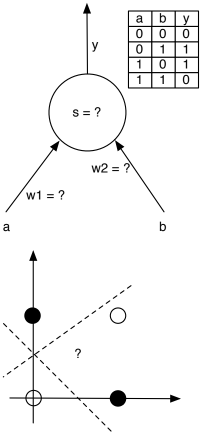
a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

y = a ET b



a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

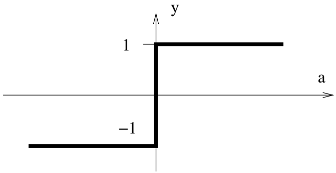
y = a XOR b



a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

Le Perceptron pour la classification

- entrées: vecteur $\mathbf{x}$ de dimension d, valeurs réelles
- sortie: classe A si $a \geq 0$, classe B si $a < 0$.
- fonction d'activité: f=sign(a)
- fonction signe :



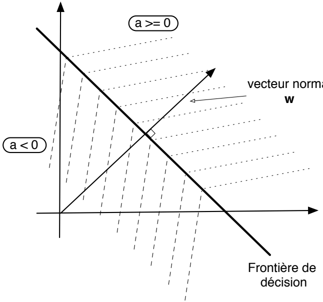
Problèmes :

- est-ce qu'on peut résoudre tout problème de classification ?
- comment déterminer les valeurs des poids et du seuil ?

Le perceptron

- frontière de décision :

$$\sum_i w_i x_i + s = 0$$



Le perceptron ne peut résoudre que des problèmes qui sont linéairement séparables

Minsky & Pappert, 1969 :

"Le perceptron ne sert à rien puisqu'il ne sait même pas résoudre le ou exclusif."

Le perceptron : apprentissage

- Objectif : avoir un algorithme qui détermine les paramètres du perceptron pour un problème de classification donné.
- Moyen : une base d'apprentissage avec des exemples typiques $\mathbf{x}$ et des réponses désirées $\mathbf{c}$ ($\rightsquigarrow$ classe) :
$\{(\mathbf{x}^1, \mathbf{c}^1), \ldots, (\mathbf{x}^N, \mathbf{c}^N)\}$ où $\mathbf{x} \in \mathbb{R}^p$ et $\mathbf{c} = \pm 1$

$\Rightarrow$ trouver $\mathbf{w}$ et s par une méthode automatique tels que:

$$\begin{aligned} \mathbf{w}^t\mathbf{x} + s &>= 0 && \text{pour tous les } \mathbf{x} \text{ de la classe A (+1)} \\ \mathbf{w}^t\mathbf{x} + s &< 0 && \text{pour tous les } \mathbf{x} \text{ de la classe B (-1)} \end{aligned}$$

Le Perceptron : simplification de notation

- Incorporer le seuil s dans le vecteur des poids :

$$\mathbf{w}^t\mathbf{x} + s \lessgtr 0 \Longleftrightarrow (s, w_1, \ldots, w_p) \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{pmatrix} = \bar{\mathbf{w}}^t\bar{\mathbf{x}} \lessgtr 0$$

- Inverser tous les exemples de la classe B:

  les exemples de la classe B sont correctement classés si

$$\bar{\mathbf{w}}^t\bar{\mathbf{x}} < 0 \qquad \Longleftrightarrow \qquad \bar{\mathbf{w}}^t(-\bar{\mathbf{x}}) >= 0$$

$\Rightarrow$ classification correcte si

$$\begin{aligned} \bar{\mathbf{w}}^t\hat{\mathbf{x}} &>= 0 \text{ pour tous les } \mathbf{x} \\ \text{avec } \hat{\mathbf{x}} &= \begin{cases} \bar{\mathbf{x}} & \text{si } x \in \text{ classe A} \\ -\bar{\mathbf{x}} & \text{si } x \in \text{ classe B} \end{cases} \end{aligned}$$

Le perceptron : algorithme d'apprentissage

Règle du perceptron (correction d'erreur) :

Répéter tant qu'il y a des exemples mal classés :

- classer l'exemple courant $\mathbf{x}$:

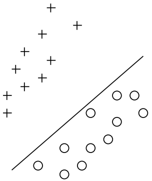
  si réponse correcte ($\bar{\mathbf{w}}^t \hat{\mathbf{x}} >= 0$) alors $\hat{\mathbf{w}}^{t+1} = \hat{\mathbf{w}}^t$
  si réponse fausse ($\bar{\mathbf{w}}^t \hat{\mathbf{x}} < 0$) alors $\hat{\mathbf{w}}^{t+1} = \hat{\mathbf{w}}^t + \mathbf{x}$

Variantes :

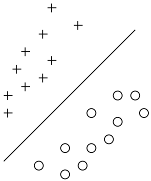
1) a. recommencer avec le 1er ex. après chaque changement de $\hat{\mathbf{w}}$
   b. présentation cyclique des exemples
2) a. changement de $\hat{\mathbf{w}}$ à chaque erreur
   b. cumul des changements de tous les exemples mal classés et une seule mise à jour de $\hat{\mathbf{w}}$ par passe à travers la base d'apprentissage

Le perceptron : inconvénients

- L'algorithme ne converge que si le problème est linéairement séparable. Sinon le comportement n'est pas défini. Il n'est pas certain qu'il trouve une solution approchée.
- Qualité de la solution non garantie



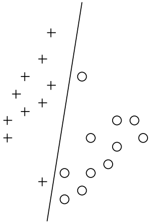
mauvaise solution ?

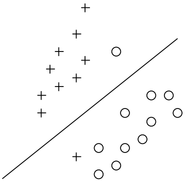


meilleure solution

Le perceptron : inconvénients

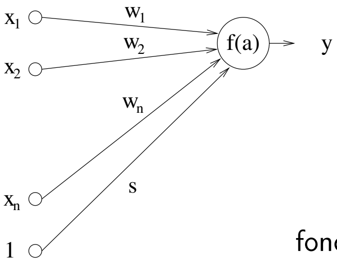
- Dans certains problèmes, il peut être préférable de commettre quelques erreurs, plutôt de donner une solution sur des cas trop spécifiques (probablement erronés).


solution trop spécifique


solution préfèrable

Adaline

- L'**Ada**ptive **Li**near **Ne**uron a été développé dans le contexte du traitement du signal (1960).
- Neurone qui possède des valeurs d'activation continue et une fonction d'activation linéaire.



entrées:	x_i
poids:	w_i
seuil:	s
activité:	$a = \sum_i w_i x_i + s$
sortie:	$y = f(a)$
fonction d'activité:	$f(a) : y = a$ (linéaire)

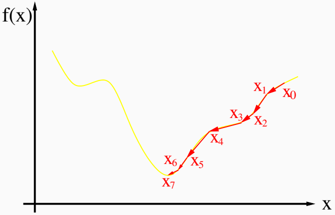
Adaline: algorithme d'apprentissage

Règle de Widrow et Hoff – Delta règle

Principe :

Descente de gradient sur une fonction d'erreur f

$$w_{i,j}(t+1) = w_{i,j}(t) + \lambda\,(a - \mathbf{c})\,x_j$$



Minimiser $f(x)$:

Choisir point de départ x_0

Procédure itérative :

- faire un petit pas dans la direction de la plus grande pente (gradient négatif)
- $x_{t+1} = x_t - \lambda \dfrac{\partial f(x)}{\partial x}$

Adaline: algorithme d'apprentissage

Principe :

Descente de gradient sur une mesure d'erreur quadratique

$$E = \frac{1}{2} \sum_{ex\ e} (a^e - \mathbf{c}^e)^2$$

Calcul pour un exemple (on ignore la somme sur les exemples) :

$$\begin{aligned} E &= \frac{1}{2}(a - c)^2 \\ w_i &= w_i - \lambda \Delta w_i \end{aligned}$$

$$\text{avec } \Delta w_i = \frac{\partial E}{\partial w_i}$$

Adaline: algorithme d'apprentissage

Suite :

$$
\begin{aligned}
E &= \frac{1}{2}(a - \mathbf{c})^2 \\
w_i &= w_i - \lambda \Delta w_i
\end{aligned}
$$

$$
\begin{aligned}
\text{avec } \Delta w_i &= \frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial w_i} \\
&= \frac{\partial}{\partial a}\left(\frac{1}{2}(a - \mathbf{c})^2\right)\frac{\partial}{\partial w_i}\left(\sum_j w_j x_j\right) \\
&= (a - \mathbf{c})\, x_i
\end{aligned}
$$

Comparaison perceptron/Adaline

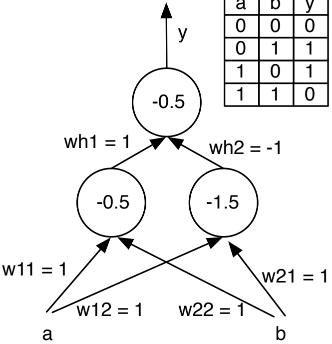
	Règle du perceptron	Règle de Widrow-Hoff
critère minimisé	$E = \sum_{\substack{\text{exemples } e \\ \text{mal classés}}} \left(-\bar{\mathbf{w}}^t \hat{\mathbf{x}}^e\right)$	$E = \sum_e \left(\bar{\mathbf{w}}^t \bar{\mathbf{x}}^e - \mathbf{c}^e\right)^2$
mise à jour des poids	$\Delta\bar{\mathbf{w}} = \hat{\mathbf{x}}$ si $\bar{\mathbf{w}}^t \hat{\mathbf{x}} < 0$	$\Delta\mathbf{w} = \epsilon\left(\bar{\mathbf{w}}^t \bar{\mathbf{x}} - \mathbf{c}\right)\mathbf{x}$ avec $\epsilon \to 0$
corrections des poids	de taille fixe	la taille est fonction de l'erreur
convergence	linéairement séparable : $\to$ convergence pas linéairement séparables : $\to$ comportement indéfini	minimisation d'une fonction qui tient compte du comportement désiré

Perceptron et Adaline : Conclusion

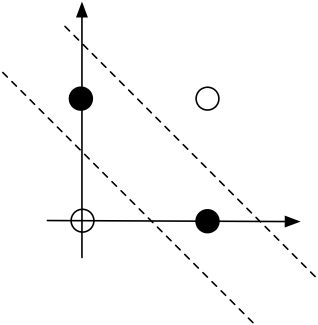
- Le perceptron et l'Adaline sont deux modèles de systèmes adaptatifs basés sur des simples automates linéaires.
- Ils peuvent apprendre à l'aide d'une base d'exemples à construire un classifieur grâce à une procédure d'apprentissage qui modifie leurs poids.
- La règle d'apprentissage de l'Adaline se comporte mieux que la règle du Perceptron. Elle converge toujours vers une solution qui minimise l'erreur entre les sorties réelles et les sorties désirées.
- Mais, l'erreur quadratique ne minimise pas forcément le nombre de mauvaises classifications.
- En plus, ces modèles sont intrinsèquement limités à des simples problèmes linéairement séparables.

Motivation pour les perceptrons multi-couches

Ajouter une couche dans un réseau à seuil permet de résoudre le problème du XOR. (Il existe de nombreuses solutions à ce problème).

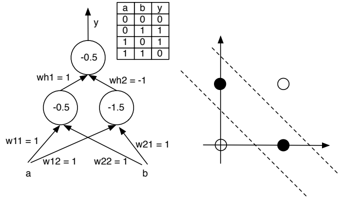


a	b	y
0	0	0
0	1	1
1	0	1
1	1	0



a XOR b = (a OU b) ET NON (a ET b)

Le perceptron multi-couches



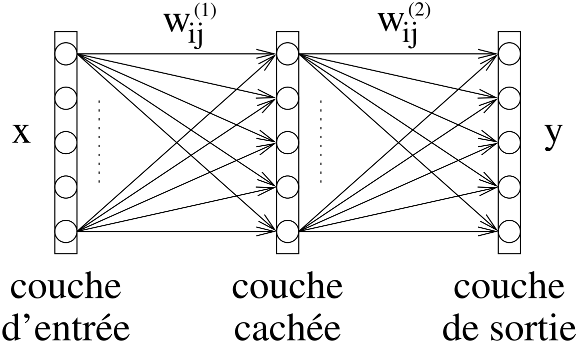
a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

- Lorsqu'on utilise des fonctions d'activité non-linéaires un réseau de neurones multi-couches permet de calculer toute fonction non-linéaire de $\mathbb{R}^n \rightarrow \mathbb{R}^m$.

$\rightarrow$ reconnaissance : comment évaluer $\mathbf{y}$ pour une entrée $\mathbf{x}$ donnée

$\rightarrow$ apprentissage : comment déterminer $\mathbf{W}$ pour obtenir le comportement désiré ?

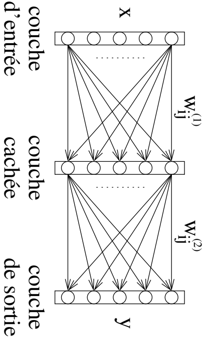
Le perceptron multi-couches : reconnaissance



Le perceptron multi-couches : reconnaissance

- Évaluation des activités couche par couche :

$$
\begin{aligned}
y_i^2 &= f\left(\sum_j w_{ij}^1 \; x_j^1\right) \\
y_i^3 &= f\left(\sum_j w_{ij}^2 \; y_j^2\right) \\
&\vdots \\
y_i^c &= f\left(\sum_j w_{ij}^{c-1} \; y_j^{c-1}\right)
\end{aligned}
$$



$\Rightarrow$ propagation de l'entrée **x** vers la sortie **y**

Le perceptron multi-couches : apprentissage

- utiliser une base d'apprentissage avec des exemples typiques et des réponses désirées : $\{(\mathbf{x}^1, \mathbf{c}^1), \ldots (\mathbf{x}^N, \mathbf{c}^N)\}$
- minimiser un critère de différence entre $\mathbf{y}$ et $\mathbf{c}$:

$$J = \sum_{\text{ex. } e} E(\mathbf{y}^e, \mathbf{c}^e)$$

par une méthode numérique d'optimisation,
p.ex. par descente de gradient :

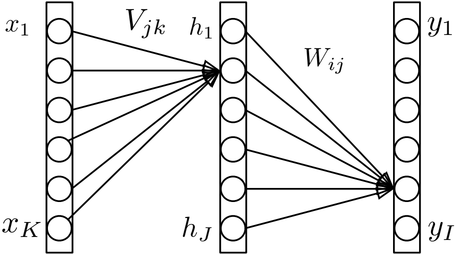
$$w_{ij}^{nouv} = w_{ij}^{avant} - \lambda \frac{\partial J}{\partial w_{ij}}$$

$\Rightarrow$ comment calculer $\partial E / \partial w_{ij}$?

a) couche de sortie : facile

b) couche cachée : problématique puisqu'on n'a plus de réponses désirées

Rétro-propagation du gradient (Backpropagation)



$$
\begin{aligned}
h_j &= f(z_j) \\
z_j &= \sum_k v_{jk} x_k \\
y_i &= f(a_i) \\
a_i &= \sum_j w_{ij} h_j
\end{aligned}
$$

Rétro-propagation du gradient (Backpropagation)

Couche de sortie :

$$\frac{\partial E}{\partial w_{ij}} = \underbrace{\frac{\partial E}{\partial a_i}}_{\delta_i} \frac{\partial a_i}{\partial w_{ij}} = \delta_i \, h_j$$

$$\text{avec } \delta_i = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial a_i} = \frac{\partial E}{\partial y_i} f'(a_i)$$

Couche cachée :

$$\frac{\partial E}{\partial v_{jk}} = \underbrace{\frac{\partial E}{\partial z_j}}_{\gamma_j} \frac{\partial z_j}{\partial v_{jk}} = \gamma_j \, x_k$$

$$\text{avec } \gamma_j = \sum_i \frac{\partial E}{\partial a_i} \frac{\partial a_i}{\partial h_j} \frac{\partial h_j}{\partial z_j} = \sum_i \delta_i \, w_{ij} \, f'(z_j) = f'(z_j) \sum_i \delta_i w_{ij}$$

$\Rightarrow$ calcul itératif de la sortie vers l'entrée

Backpropagation : fonctions d'erreurs

Pour chaque type de problème :

- Quelle fonction d'activation ?
- Quelle fonction d'erreur ?

Régression :

$\rightarrow$ estimer une valeur dans un ensemble continu de réel

- fonction d'activation linéaire + erreur euclidienne

$$y_i = a_i \qquad \partial y_i / \partial a_i = 1$$

$$E(\mathbf{y}, \mathbf{c}) = \frac{1}{2} \sum_i (y_i - c_i)^2 \qquad \partial E / \partial y_i = (y_i - c_i)$$

Backpropagation : fonctions d'erreurs

Classification : sigmoïde + erreur euclidienne

- réponses désirées ± 0.6 pour éviter la saturation

$$y_i = \tanh(a_i) \qquad \partial y_i / \partial a_i = 1 - y_i^2$$

$$E(\mathbf{y}, \mathbf{c}) = \frac{1}{2} \sum_i (y_i - c_i)^2 \qquad \partial E / \partial y_i = (y_i - c_i)$$

Probabilités *a posteriori* : softmax + cross-entropie

$$y_i = \frac{e^{a_i}}{\sum_k e^{a_k}} \qquad \partial y_i / \partial a_k = \delta_{ik} y_i - y_i y_k$$

$$E(\mathbf{y}, \mathbf{c}) = \sum_i c_i \log y_i \qquad \partial E / \partial y_i = \frac{c_i}{y_i}$$

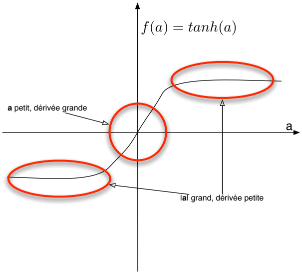
Déroulement d'un apprentissage

1. Normaliser les données
2. Initialiser les poids $\mathbf{W}$
3. Répéter
   - Choisir un exemple $(\mathbf{x}, \mathbf{c})$
   - Propager l'exemple $\mathbf{x}$ à travers le réseau $\rightarrow \mathbf{y}$
   - Évaluer la fonction d'erreur $E(\mathbf{y}, \mathbf{c})$
   - Rétro-propager le gradient d'erreur $\rightarrow \nabla w_{ij}$
   - Mise à jour des poids W
   - Modifier éventuellement les paramètres d'apprentissage

   $\rightarrow$ cf. λ plus loin.

   Jusqu'à convergence

Backpropagation: normalisation et initialisation



Si l'activité $|a|$ est très grande
$\rightarrow f'(a)$ est faible
$\rightarrow$ convergence est lente

- Normalisation des entrées du réseau :
  soustraire la moyenne et diviser par la variance
- Initialisation des poids :
  avec des valeurs aléatoires dans $[-1/\sqrt{f}, 1/\sqrt{f}]$
  où f est le *fan-in* (nombre de poids arrivant à ce neurone)

$$\Rightarrow a = \sum_i w_i x_i \text{ est relativement petit}$$

Backpropagation : choix des exemples

Théorie :

minimiser $J = \sum_{e \in E} E(\mathbf{y}^e, \mathbf{c}^e)$ avec E les exemples d'apprentissage

Méthode *batch* :

- présenter **tous** les exemples et cumuler les $\partial E / \partial w_{ij}$
- puis faire une mise à jour des poids

– problème : convergence est très lente

Backpropagation : choix des exemples

Théorie :

minimiser $J = \sum_{e \in E} E(\mathbf{y}^e, \mathbf{c}^e)$ avec E les exemples d'apprentissage

Méthode *stochastique* :

- mise à jour des poids après **chaque** exemple (choix aléatoire !)

+ on profite des redondances entre les exemples

– E peut augmenter, mais cela permet éventuellement de s'échapper d'un minimum local

Backpropagation : paramètres d'apprentissage

$$w_{ij}^{nouv} = w_{ij}^{avant} - \lambda \frac{\partial E}{\partial w_{ij}}$$

Théorie :

- λ trop petit $\rightarrow$ convergence est très lente
- λ trop grand $\rightarrow$ oscillations ou divergence
- Calcul exact possible mais trop coûteux

Pratique :

- décroissance exponentielle : $\lambda' = \frac{\lambda}{1 + nbr\_iter}$
- il y a plein d'autres heuristiques

Réseaux de Neurones en pratique

Architecture du réseau

- Dimension de l'entrée généralement donnée par le problème
- Nombre de sorties = nombre de classes
- Combien de couches cachées et combien de neurones ?

  $\Rightarrow$ c'est la magie noire des RdN ... !!
  - Pas de règles prédéfinie pour déterminer le nombre optimal
  - Il faut essayer différentes structures et garder celle qui donne les meilleurs résultats sur les données de développement

Réseaux de neurones en pratique

Apprentissage

- Il est important de mélanger les exemples

  $\rightarrow$ pas une classe après l'autre !
- Le choix des paramètres d'apprentissage
  - λ + heuristique de mise à jour
  - initialisation
- On veut apprendre les caractéristiques du problème et pas tous les moindres détails des données d'apprentissage
  - Quand faut-il arrêter l'apprentissage (convergence) ?
  - Calculer le taux d'erreur sur les données de développement après chaque itération

  $\Rightarrow$ Arrêter l'apprentissage lorsqu'il diminue plus

  $\rightarrow$ conserver un pouvoir de généralisation important

  $\rightarrow$ éviter le sur-apprentissage