Réseaux de neurones

- Motivation
- Le perceptron
- Les algorithmes d'apprentissage pour le perceptron
- Les réseaux multi-couches
- L'algorithme de retropropagation du gradient
- Le modèle de Hopfield

Introduction

- En Intelligence Artificielle on s'intéresse au fonctionnement du cerveau
- Démarche: Connexionisme
- Cerveau: Réseau complexe d'unités de calcul élémentaire interconnectés
- On considère un modèle très simple qui ne prend pas en compte de nombreuses caractéristiques biologiques

Le cerveau

- composés de neurones
- reçoivent des signaux par les dendrites
- alimentent un axone
- Les contactes entre deux neurones se font avec des synapses (jusqu'à 1000 par connexion)
- Effet de seuil
- Il y a à peu près 100 milliards de neurones
- 10^{15} connexions

Réseaux de neurones formels

- On essaie de modéliser le fonctionnement du cerveau
- Classification des réseaux:
  - Un réseau est composé de cellules élémentaires
  - Une cellule peut manipuler des valeur réelles ou (0 et 1) ou (-1 et 1)
  - Plusieurs fonctions pour le calcul de sortie
  - L'architecture du réseau peut être sans ou avec rétroaction
  - La dynamique du réseau peut être synchrone ou asynchrone (aléatoire ou pas)
- Applications:
  - Apprentissage

Le perceptron

- Un perceptron linéaire à seuil prend en entrée n valeurs $x_1, \ldots, x_n$ et calcule une sortie o.
- Il est défini par $n+1$ constantes:
  - Les coefficients synaptiques $w_1, \ldots, w_n$ et le seuil (ou biais) θ
- o est calculé par: $$o = \begin{cases} 1 & \text{si } \sum_{i=1}^{n} w_i x_i > \theta \\ 0 & \text{sinon} \end{cases}$$
- Les entrées peuvent être à valeurs dans $\{0,1\}$ ou réelles, les poids peuvent être entiers ou réels
- Pour simplifier les notations nous remplaçons le seuil par une entrée supplémentaire qui prend comme valeur d'entrée $x_0 = 1$. On associe le poids w_0 qui correspond à $-\theta$.
- On calcule d'abord $\sum_{i=0}^{n} w_i x_i$ qu'on appelle le potentiel post-synaptique et en suite on applique une fonction d'activation. Ici c'est la fonction de Heaviside (qui donne 1 si $\sum_{i=0}^{n} w_i x_i > 0$ et 0 sinon)

Interprétation géométrique

- Soit S un ensemble d'exemples dans $\mathbb{R}^n \times \{0,1\}$.
  - Soit $S_0 = \{s \in \mathbb{R}^n \mid (s,0) \in S\}$
  - Soit $S_1 = \{s \in \mathbb{R}^n \mid (s,1) \in S\}$.
- S est dit linéairement séparable s'il existe un hyperplan H de $\mathbb{R}^n$ tel que les ensemble S_0 et S_1 soient situés de part et d'autre de ce hyperplan.
- Théorème: Un perceptron linéaire à seuil à n entrées divise l'espace des entrées $\mathbb{R}^n$ en deux sous-espaces délimités par un hyperplan.
- Théorème: Tout ensemble linéairement séparable peut être discriminé par un perceptron.
- Le XOR ne peut pas être calculé par un perceptron linéaire à seuil.

Algorithme d'apprentissage par correction d'erreur

- Soit S un ensemble d'apprentissage de $\{0,1\}^n \times \{0,1\}$ ou $\mathbb{R}^n \times \{0,1\}$
- Problème: Trouver les poids d'un perceptron qui classifie correctement l'ensemble S
- Notations:
  - $\vec{x}$ une description, un élément de $\mathbb{R}^n$ ou $\{0,1\}^n$
  - S est un ensemble de couples $(\vec{x}, c)$
  - $(\vec{x}^s, c^s)$: le s-ème élément de S
  - o^s la sortie du perceptron quand on présente $\vec{x}^s$
- Rappel: Il existe une $n+1$-ème entrée x_0 de valeur 1.

Algorithme d'apprentissage par correction d'erreur

Entrée: un échantillon S de $\mathbb{R}^n \times \{0,1\}$ ou $\{0,1\}^n \times \{0,1\}$
  Initialisation aléatoire des poids w_i pour i de 0 à n
  Répéter
    Prendre un exemple $(\vec{x}, c)$ dans S
    Calculer la sortie o du perceptron pour l'entrée $\vec{x}$
    Pour i de 0 à n
      $w_i \leftarrow w_i + (c - o)x_i$
    fin Pour
  fin Répéter
Sortie: Un perceptron P défini par $(w_0, w_1, \ldots, w_n)$

- Dans quel ordre présente-on les exemples ?
- Critères d'arrêt ?
- Théorème: Si S est linéairement séparable et si les exemples sont présentés équitablement, la procédure d'apprentissage par correction d'erreur converge vers un perceptron linéaire à seuil qui calcule S.

Exemple d'apprentissage par correction d'erreur

	w_0	w_1	w_2	Entrée	Σ	o	c	w_0	w_1	w_2
1	0	0	0	1 0 0	0	0	0	0	0	0
2	0	0	0	1 0 1	0	0	1	1	0	1
3	1	0	1	1 1 0	1	1	1	1	0	1
4	1	0	1	1 1 1	2	1	1	1	0	1
5	1	0	1	1 0 0	1	1	0	0	0	1
6	0	0	1	1 0 1	1	1	1	0	0	1
7	0	0	1	1 1 0	0	0	1	1	1	1
8	1	1	1	1 1 1	3	1	1	1	1	1
9	1	1	1	1 0 0	1	1	0	0	1	1

Problèmes et Questions

- Qu'est-ce que se passe si S n'est pas linéairement séparable ?
- Est-ce qu'on peut borner la valeur des poids et le seuil ?
- Combien de pas faut-il pour converger ?
- Est-ce que la solution trouvée est robuste ?
- Est-ce que l'algorithme est tolérant aux bruits ?

Apprentissage par descente du gradient

- Idée: On définit une fonction d'erreur qu'on essaie de minimiser avec la méthode de descente du gradient.
- Un perceptron linéaire prend en entrée un vecteur $\vec{x}$ de n valeurs et calcule une sortie o. Il est défini par un vecteur $\vec{w}$ de constantes. La sortie est définie par:
$$o = \vec{x}.\vec{w} = \sum_{i=1}^{n} w_i x_i$$
- L'erreur d'un perceptron P défini par $\vec{w}$ sur un échantillon S d'exemples $(\vec{x}^s, c^s)$ est donné par:
$$E(\vec{w}) = \frac{1}{2} \sum_{(\vec{x}^s, c^s)} (c^s - o^s)^2$$
- L'erreur est 0 ssi le perceptron classifie correctement l'ensemble S
- Il faut déterminer un $\vec{w}$ qui minimise $E(\vec{w})$.

Descente du gradient

- Étant donné une fonction f on construit une suite (x_n)
- Cette suite devrait s'approcher du minimum.
- On part de x_0 quelconque et on définit $x_{n+1} = x_n + \Delta x_n$ avec $\Delta x_n = -\epsilon f'(x_n)$, où ϵ est une valeur "bien choisi".
- rien ne garantit que le minimum trouvé est un minimum global

Algorithme d'apprentissage par descente du gradient

- $E(\vec{w})$ est une fonction de n variables. La méthode du gradient s'applique.
- On calcule la dérivée partielle de E par rapport à w_i:

$$\begin{aligned}
\frac{\partial(\vec{w})}{\partial w_i}E &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_S (c^s - o^s)^2 \\
&= \frac{1}{2}\sum_S \frac{\partial}{\partial w_i}(c^s - o^s)^2 \\
&= \frac{1}{2}\sum_S 2(c^s - o^s)\frac{\partial}{\partial w_i}(c^s - o^s) \\
&= \sum_S (c^s - o^s)\frac{\partial}{\partial w_i}(c^s - \vec{w}.\vec{x}^s) \\
&= \sum_S (c^s - o^s)(-x_i^s)
\end{aligned}$$

- On définit $$\Delta w_i = -\epsilon \times \frac{\partial(\vec{w})}{\partial w_i} = \epsilon \sum_S (c^s - o^s)(x_i^s)$$

L'algorithme d'apprentissage par descente de gradient

On n'a plus la notion de seuil (poids w_0)

Entrée: un échantillon S de $\mathbb{R}^n \times \{0,1\}$; ϵ
Initialisation aléatoire des poids w_i pour i de 1 à n
Répéter
  Pour tout i $\Delta w_i \leftarrow 0$ **fin Pour**
  Pour tout exemple $(\vec{x}^s, c^s)$ de S
    Calculer la sortie o^s
    Pour tout i $\Delta w_i \leftarrow \Delta w_i + \epsilon(c^s - o^s)x_i^s$ **fin Pour**
  fin Pour
  Pour tout i $w_i \leftarrow w_i + \Delta w_i$ **fin Pour**
fin Répéter
Sortie: Un perceptron P défini par $(w_1, \ldots, w_n)$

L'algorithme d'apprentissage de Widrow-Hoff

Entrée: un échantillon S de $\mathbb{R}^n \times \{0,1\}$ ou $\{0,1\}^n \times \{0,1\}$
  Initialisation aléatoire des poids w_i pour i de 1 à n
  Répéter
    Prendre un exemple $(\vec{x}, c)$ dans S
    Calculer la sortie o du perceptron pour l'entrée $\vec{x}$
    Pour i de 1 à n
      $w_i \leftarrow w_i + \epsilon(c - o)x_i$
    fin Pour
  fin Répéter
Sortie: Un perceptron P défini par $(w_1, \ldots, w_n)$

Les réseaux multi-couches

- On peut facilement faire le XOR
- Un réseau de neurones à couches cachées est définie par une architecture vérifiant les propriétés:
  - Les cellules sont réparties dans des couches $C_0, C_1, \ldots, C_q$.
  - La première couche C_0 (rétine) est composée des cellules d'entrée (correspondent aux n variables d'entrée).
  - Les couches $C_1, \ldots, C_{q-1}$ sont les couches cachées.
  - La couche C_q est composée des cellules de décision.
  - Les entrées d'une cellule d'une couche C_i sont toutes les cellules de la couche C_{i-1} seulement.
- La dynamique du réseau est synchrone par couche.
- Perceptron multi-couches (PMC) linéaires à seuil.
- Chaque fonction booléenne peut être calculée par un PMC linéaire à seuil avec une seule couche cachée.

Utilisation en apprentissage

- Comment choisir l'architecture du réseau ?
- Quel algorithme d'apprentissage utilisé ?

L'algorithme de retropropagation du gradient

- On choisit une fonction d'activation dérivable qui s'approche de la fonction de Heaviside:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

- Une cellule élémentaire à n entrée réelles $\vec{x} = (x_1, \ldots, x_n)$ est définie par les poids synaptiques réels $\vec{w} = (w_1, \ldots, w_n)$ et la sortie est calculée par:

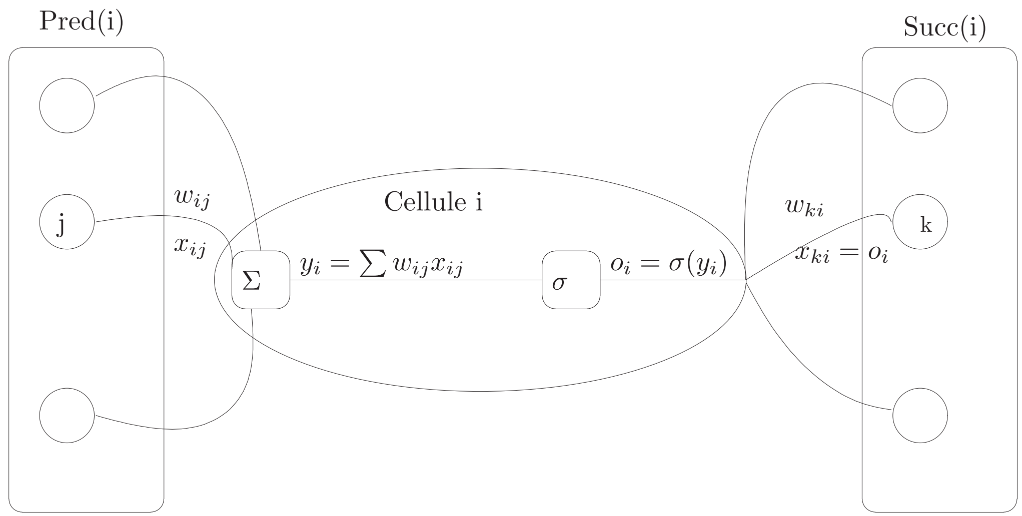
$$o(\vec{x}) = \frac{1}{1+e^{-y}} \text{ avec } y = \vec{x}\vec{w} = \sum_{i=1}^{n} w_i x_i$$

- L'erreur du PMC sur un échantillon S est définie par:

$$E(\vec{w}) = \frac{1}{2} \sum_{(\vec{x^s},\vec{c^s})\in S} \sum_{k=1}^{p} (c_k^s - o_k^s)^2$$

où o_k^s est la k-ième composante du vecteur de sortie $\vec{o}$ calculée par le PMC sur l'entrée $\vec{x}$.

Algorithme de retropropagation du gradient (Notation)



Calcul de $\frac{\partial E(\vec{w})}{\partial w_{ij}}$

On minimise l'erreur du réseau sur un exemple en appliquant la méthode de descente du gradient. Pour cela, il faut calculer

$$\frac{\partial E(\vec{w})}{\partial w_{ij}} = \frac{\partial E}{\partial w_{ij}}$$

- w_{ij} ne peut influencer la sortie du réseau qu'à travers y_i:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} x_{ij}$$

- Il suffit donc de calculer

$$\frac{\partial E}{\partial y_i}$$

- Il y a deux cas: la cellule i est
  - une cellule de sortie
  - une cellule interne

i est une cellule de sortie

- y_i n'influence la sortie du réseau qu'à travers de o_i:

$$\frac{\partial E}{\partial y_i} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial y_i}$$

$$\frac{\partial E}{\partial o_i} = \frac{\partial}{\partial o_i}\frac{1}{2}\sum_{k=1}^{p}(c_k - o_k)^2 = \frac{\partial}{\partial o_i}\frac{1}{2}(c_i - o_i)^2 = -(c_i - o_i)$$

$$\frac{\partial o_i}{\partial y_i} = \frac{\partial \sigma(y_i)}{\partial y_i} = \sigma(y_i)(1-\sigma(y_i)) = o_i(1-o_i)$$

- On obtient donc:

$$\frac{\partial E}{\partial y_i} = -(c_i - o_i)o_i(1-o_i)$$

i est une cellule interne

- y_i influence le réseau par tous les calculs des cellules de $Succ(i)$

$$\frac{\partial E}{\partial y_i} = \sum_{k\in Succ(i)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial y_i} = \sum_{k\in Succ(i)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial o_i}\frac{\partial o_i}{\partial y_i} = \sum_{k\in Succ(i)} \frac{\partial E}{\partial y_k} w_{ki} o_i (1-o_i)$$

$$= o_i(1-o_i)\sum_{k\in Succ(i)} \frac{\partial E}{\partial y_k} w_{ki}$$

- On choisit

$$\Delta w_{ij} = -\epsilon \frac{\partial E(\vec{w})}{\partial w_{ij}}$$

- On définit

$$\delta_i = -\frac{\partial E}{\partial y_i}$$

- Cellule de sortie: $\delta_i = o_i(1-o_i)(c_i - o_i)$

- Cellule interne: $\delta_i = o_i(1-o_i)\sum_{k\in Succ(i)} \delta_k w_{ki}$

Algorithme de retropropagation du gradient

Entrée: un PMC avec une couche d'entrée C_0, $q-1$ couches cachées $C_1, \ldots, C_{q-1}$ une couche de sortie C_q, n cellules.

Initialisation aléatoire des poids w_i dans $[-0.5, 0.5]$ pour i de 1 à n
Répéter
  Prendre un exemple $(\vec{x}, c)$ dans S et calculer $\vec{o}$
  Pour toute cellule de sortie i $\delta_i \leftarrow o_i(1-o_i)(c_i - o_i)$ **FinPour**
  Pour chaque couche de $q-1$ à 1
    Pour chaque cellule i de la couche courante
      $\delta_i \leftarrow o_i(1-o_i)\sum_{k\in Succ(i)} \delta_k w_{ki}$
    fin Pour
  fin Pour
  Pour tout poids $w_{ij} \leftarrow w_{ij} + \epsilon\delta_i x_{ij}$ **fin Pour**
fin Répéter

Sortie: Un PMC défini par la structure initiale choisie et les w_{ij}

Remarques

- C'est une extension de l'algorithme de Widrow-Hoff
- Peu de problèmes avec les minima locaux
- On peut utiliser un *moment* (momentum)
  - On fixe une constante $\alpha \in [0, 1[$
  - La règle de modification des poids devient

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}(t)$$

$$\Delta w_{ij}(t) = \epsilon\delta_i x_{ij} + \alpha\Delta w_{ij}(t-1)$$

  où t est un compteur des itérations.

- Quel critère d'arrêt ?
- Ordre de présentation des exemples ?
- Comment choisir les paramètres ?

Modèle de Hopfield

- Exemple d'un réseau avec rétroaction
- utilisé comme mémoire associative
  - Un certain nombre de formes est stocké dans un réseau
  - On fournit une partie d'une de ses formes
  - Le réseau reconstitue la forme complète

Modèle de Hopfield - Définitions

- Réseau avec N neurones
- tous les neurones sont connectés entre eux
- chaque neurone à une sortie -1 ou 1
- la sortie o_j est donnée par $S(\sum_i w_i^j o_i)$ où les w_i^j sont les poids du réseau et S la fonction $S(x) = 1$ si $x \geq 0$, -1 sinon.
- On fait évoluer le réseau soit de façon synchrone (on calcule tous les o_j simultanément) ou de façon asynchrone (on calcule les o_j un par un dans un ordre donné ou aléatoire)

Mémorisation d'un vecteur binaire

- On veut mémoriser un vecteur $\vec{x}^1 = (x_1^1, x_2^1, \ldots, x_n^1) \in \{-1, 1\}^N$
- On aimerait que si on fait évoluer le réseau, les sorties des neurones ne changent pas pour ce vecteur (stabilité)
- On doit avoir pour tout j: $S(\sum_i w_i^j x_i^1) = x_j^1$
- Cela est vrai, si on choisit w_i^j avec le même signe que $x_j^1 x_i^1$
- On choisit: $w_i^j = \frac{1}{N} x_j^1 x_i^1$ (règle de Hebb)

Mémorisation d'un vecteur binaire

- Chaque vecteur $\vec{y}$ "proche" de $\vec{x}$ dans l'espace de Hamming ($\{-1, 1\}^N$) est "attiré" vers $\vec{x}$.
- $S(\sum_i w_i^j y_i) = S(\sum_i x_j^1 x_i^1 y_i) = S(x_j^1) S(\sum_i x_i^1 y_i) = x_j^1 S(\sum_i x_i^1 y_i)$
- Ce terme est égal à x_j^1 si la somme $\sum_i x_i^1 y_i$ est positive, c.-à-d. si le nombre de x_i^1 égaux au nombre de y_i est supérieur à $N/2$.
- Pour tout vecteur initial situé à une distance de Hamming (le nombre de composantes différentes) inférieure à $N/2$ de $\vec{x}^1$, le réseau évoluera vers $\vec{x}^1$.
- Pour tout vecteur initial situé à une distance de Hamming supérieure à $N/2$ de $\vec{x}^1$, le réseau évoluera vers $-\vec{x}^1$.
- On mémorise le vecteur et son complément

Exemple

- On veut mémoriser le vecteur $\vec{x}^1 = (-1, -1, 1, 1, 1)$
- On calcule les vecteurs de poids $\vec{w}^j = (w_1^j, w_2^j, w_3^j, w_4^j, w_5^j)$ associés à chaque neurone j
- $\vec{w}^1 = \vec{w}^2 = \frac{1}{5}(1, 1, -1, -1, -1), \vec{w}^3 = \vec{w}^4 = \vec{w}^5 = \frac{1}{5}(-1, -1, 1, 1, 1)$
- On vérifie que $S(\sum_i w_i^j x_i^1) = x_j^1$ pour tout j
- On peut prendre n'importe quel vecteur de dimension 5 et le faire évoluer

Mémoriser plusieurs vecteurs

- On veut mémoriser P vecteurs $\vec{x}^1, \ldots, \vec{x}^P$
- On utilise la règle de Hebb généralisé: $w_i^j = \frac{1}{N} \sum_{k=1}^{P} x_j^k x_i^k$
- Est-ce qu'avec cette règle on satisfait la règle de stabilité ?
- On aimerait pour tout k et j : $S(\sum_i w_i^j x_i^k) = x_j^k$
- $S(\sum_i w_i^j x_i^k) = S(\sum_i \frac{1}{N} \sum_l x_j^l x_i^l x_i^k) = S(\frac{1}{N} \sum_i x_j^k x_i^k x_i^k + \frac{1}{N} \sum_i \sum_{l \neq k} x_j^l x_i^l x_i^k) = S(x_j^k + \frac{1}{N} \sum_i \sum_{l \neq k} x_j^l x_i^l x_i^k)$
- Une condition suffisante pour que ce terme soit égale à x_j^k est: $1 > |\frac{1}{N} \sum_i \sum_{l \neq k} x_j^l x_i^l x_i^k)|$

Exemple

- On veut mémoriser le vecteur $\vec{x}^1 = (-1, -1, 1, 1, 1)$ et le vecteur $\vec{x}^1 = (1, -1, 1, -1, 1)$
- On calcule les vecteurs de poids $\vec{w}^j$ associé à chaque neurone j
- $\vec{w}^1 = \frac{1}{5}(2, 0, 0, -2, 0), \vec{w}^2 = \frac{1}{5}(0, 2, -2, 0, -2), \vec{w}^3 = \frac{1}{5}(0, -2, 2, 0, 2), \vec{w}^4 = \frac{1}{5}(-2, 0, 0, 2, 0), \vec{w}^5 = \frac{1}{5}(0, -2, 2, 0, 2)$
- On vérifie que $S(\sum_i w_i^j x_i^1) = x_j^1$ et $S(\sum_i w_i^j x_i^2) = x_j^2$ pour tout j
- On peut prendre n'importe quel vecteur de dimension 5 est le faire évoluer