

Réseaux de neurones artificiels et apprentissage profond

Julien VELCIN

Université Lumière Lyon 2

Master 2 MALIA

Remerciements

- Julien Ah-Pine
MCF d'Informatique (ICOM, Lyon 2)
- Mathieu Lefort
MCF d'Informatique (FST, Lyon 1)
- Julien Velcin
MCF d'Informatique (FSEG, Lyon 2)

Plan

Introduction générale

Le perceptron simple

Le perceptron multicouches

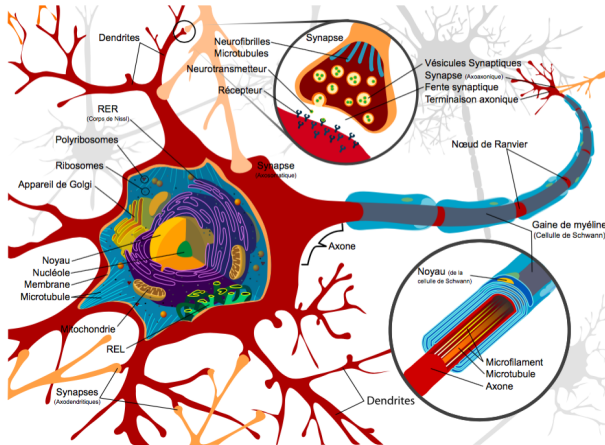
Au-delà du MLP

Références

Réseaux de neurones artificiels

- Modèles (lointainement) inspirés du fonctionnement de notre cerveau et de travaux en sciences cognitives et neurosciences
- Notre cerveau est capable de traiter des problèmes difficiles : reconnaissance de visages, de la parole, etc.
- Paradigme **connexionniste**, une vision *parallèle* et *distribuée* de l'activité du cerveau vu comme un système de traitement de l'information :
 - chaque neurone traite l'information indépendamment des autres avant d'en communiquer le résultat à d'autres neurones grâce aux synapses
 - des parties du cerveau (groupes de neurones) se spécialisent dans certaines tâches

Schéma d'un neurone biologique



Comparons les deux “machines”

Caractéristiques	Cerveau humain	Ordinateur
Support des données	neurones	circuits électroniques
Véhicule des données	neurotransmetteurs (substances chimiques)	impulsions électriques
Enregistrement des données	analogique (continu)	numérique (binaire)
Types de mémoires	à long terme à court terme registres sensoriels externes : bibliothèques	morte (ROM) vive (RAM) tampon (<i>buffer</i>) périphériques : disques, ...
Localisation, spécialisation des fonctions	aires cérébrales spécialisées (cortex visuel, ...)	circuits spécialisés (CPU, mémoires, contrôleurs, horloge, ...)
Nombre de cellules	≈ 30 000 000 000 (10 ¹⁰)	4 000 000 000 (10 ⁹)
Liaisons par cellule	≈ 10 000 (10 ⁴)	2 (cellules voisines)
Structure	réseau (non linéaire)	liste (linéaire)
Durée des impulsions	0,001 sec (10 ⁻³)	0,000 000 001 sec (10 ⁻⁹)
Vitesse de propagation	130 m/sec (10 ²)	300 000 000 m/sec (10 ⁸)
Temps d'accès (sec.)	0,1 sec (10 ⁻¹)	0,000 000 1 sec (10 ⁻⁷)
Débit (<i>bits/sec.</i>)	faible (10 ²)	fort (10 ⁸)

<http://intelligence-artificielle-tpe.e-monsite.com/pages/limites-technologiques-et-ethique-de-l-ia/cerveau-humain-et-robot.html>

Un peu d'histoire

- 1943 : neurone formel de McCulloch & Pitts
"A logical calculus of the ideas immanent in nervous activities"
- 1948 : Von Neumann – les réseaux d'automates
- 1949 : Donald Hebb – hypothèse de l'efficacité synaptique, notion de mémoire associative, premières règles locales d'apprentissage
- 1958 : Rosenblatt et Widrow - Perceptron et Adaline
- 1969 : Minsky et Papert – analyse théorique des capacités de calcul des réseaux à base de perceptron.
- Début des années 70 : stagnation des recherches sur les réseaux neuromimétiques ; report des efforts en Intelligence Artificielle symbolique (systèmes experts. . .)
- Années 70-80 : quelques avancées - perceptron multicouches (MLP), cartes auto-organisatrices (SOM), etc.
- Des mécanismes d'apprentissage performant pour le perceptron multicouches voient le jour (rétropropagation du gradient)

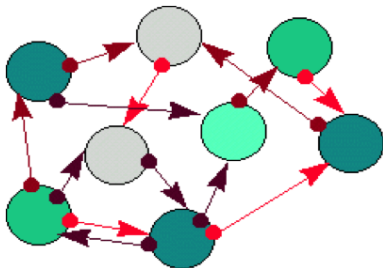
Un peu d'histoire récente

Avènement de l'apprentissage profond (*deep learning*) :

- avant 2000 : de nombreux travaux importants mais qui passent inaperçus (la faute aux SVM ?), tels que les machines de Boltzmann (1985) ou la mémoire à *long short-term memory* (LSTM) pour les réseaux récurrents (1997)
- Peu à peu, des progrès importants sont faits quant à la puissance de calculs (ex. GPUs) et les architectures profondes gagnent les compétitions
- 2013 : un réseau de neurones remporte la compétition ImageNet
- 2015 : Google annonce des taux de reconnaissance de visage de l'ordre de 99,63% avec FaceNet (réseau de neurones à 22 couches !)
- octobre 2015 : victoire d'alphaGo contre le champion européen de Go, Fan Hui
- mi-mars 2016 : victoire de DeepMind contre le champion du monde de Go, Lee Sedol (4 parties à 1)

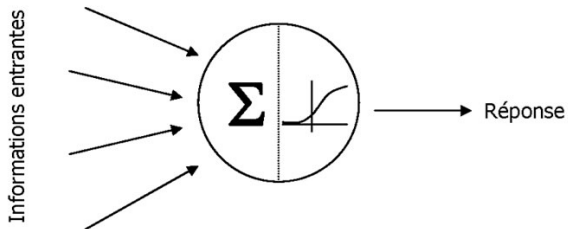
S'inspirer des modèles naturels

Un **réseau neuromimétique** est un graphe pondéré orienté dont les sommets sont appelés “neurones” et sont dotés d'un comportement d'automate simple.



Modèle de neurone artificiel

Figure 1 – Neurone artificiel (McCulloch et Pitts, 1943)



Plan

Introduction générale

Le perceptron simple

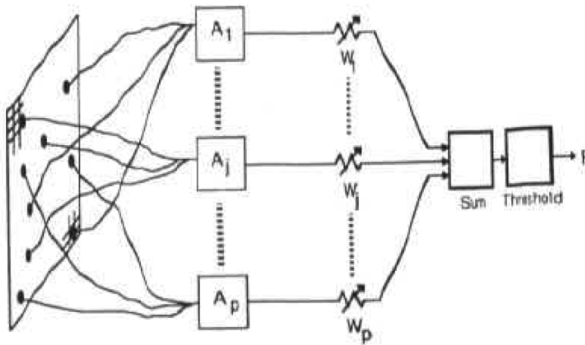
Le perceptron multicouches

Au-delà du MLP

Références

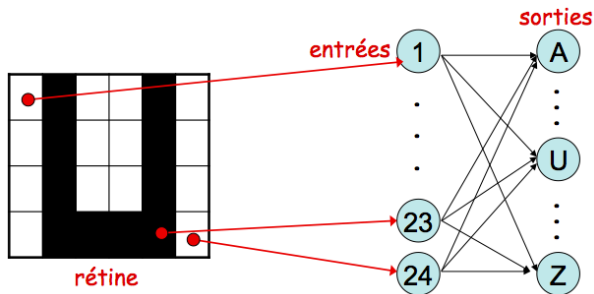
Le perceptron de Rosenblatt (1957)

Architecture simple : une entrée (la rétine) et une sortie.

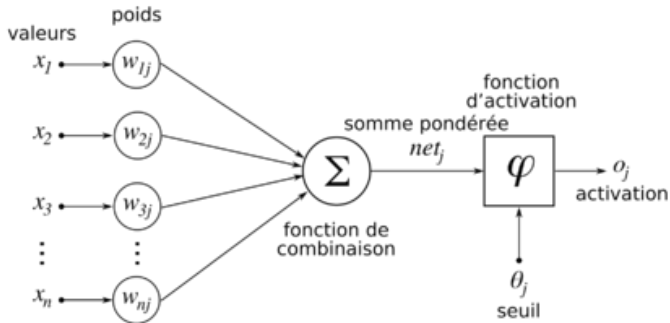


Exemple : associer des formes à des classes

Rétine de 4 x 6 éléments (image d'une lettre) avec 26 éléments en sortie



Le Perceptron



Le Perceptron

Cas à un seul neurone de sortie

Modèle de McCulloch et Pitts :

- Soit $\mathbf{x} = (x_1, \dots, x_p)$ dans \mathbb{R}^p le signal d'entrée
- Chaque x_i est associé à un poids w_i
- Le vecteur \mathbf{w} est appelé coefficients synaptiques
- On ajoute une constante w_0 appelée le biais

Le Perceptron

Cas à un seul neurone de sortie

Modèle de McCulloch et Pitts :

- Soit $\mathbf{x} = (x_1, \dots, x_p)$ dans \mathbb{R}^p le signal d'entrée
- Chaque x_i est associé à un poids w_i
- Le vecteur \mathbf{w} est appelé coefficients synaptiques
- On ajoute une constante w_0 appelée le biais

Signal post-synaptique

$$s(x) = w_0 + \sum_{i=1}^p (w_i \times x_i)$$

Le Perceptron

Cas à un seul neurone de sortie

Modèle de McCulloch et Pitts :

- Soit $\mathbf{x} = (x_1, \dots, x_p)$ dans \mathbb{R}^p le signal d'entrée
- Chaque x_i est associé à un poids w_i
- Le vecteur \mathbf{w} est appelé coefficients synaptiques
- On ajoute une constante w_0 appelée le biais

Signal post-synaptique

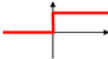
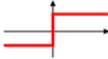
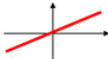
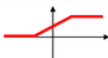
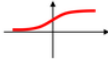
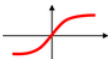
$$s(\mathbf{x}) = w_0 + \sum_{i=1}^p (w_i \times x_i)$$

$$\text{ou, si } \mathbf{x} = (1, x_1, \dots, x_p) : s(\mathbf{x}) = \sum_{i=0}^p w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

Fonction d'activation

$\hat{y}_{\mathbf{w}}(\mathbf{x}) = h(s(\mathbf{x}))$ où h peut être définie de différentes manières

Exemples de fonctions d'activation

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

Neurone à deux entrées

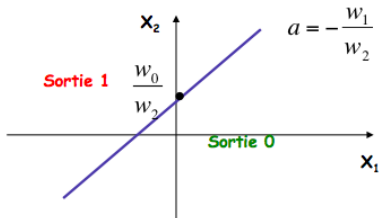
Règle d'activation linéaire

Si $s = w_1.x_1 + w_2.x_2 > w_0$ alors $\hat{y}=1$

Si $s = w_1.x_1 + w_2.x_2 \leq w_0$ alors $\hat{y}=0$

Géométriquement

Cela signifie qu'on a divisé le plan en deux régions par une droite d'équation $w_1.x_1 + w_2.x_2 - w_0 = 0$



Lien avec la régression logistique

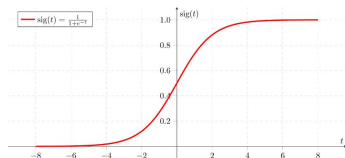
Principe

- on cherche à modéliser de façon linéaire $\ln \frac{P(Y=1|\mathbf{X}=\mathbf{x})}{1-P(Y=1|\mathbf{X}=\mathbf{x})}$:

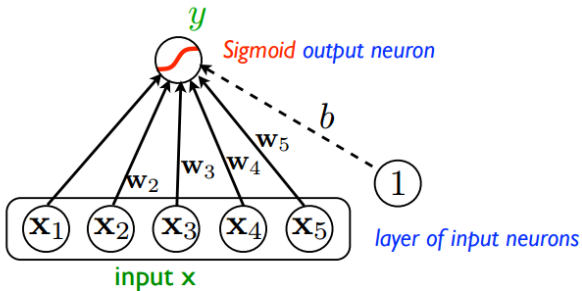
$$\ln \frac{P(Y=1|\mathbf{X}=\mathbf{x})}{1-P(Y=1|\mathbf{X}=\mathbf{x})} = \underbrace{\ln \frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})}}_{\text{logit}(\pi(\mathbf{x}))} = \beta_0 + \sum_{j=1}^p \beta_j x_j$$

ce modèle s'écrit aussi

$$\pi(\mathbf{x}) = \frac{\exp(\beta_0 + \sum_{j=1}^p \beta_j x_j)}{1 + \exp(\beta_0 + \sum_{j=1}^p \beta_j x_j)}$$

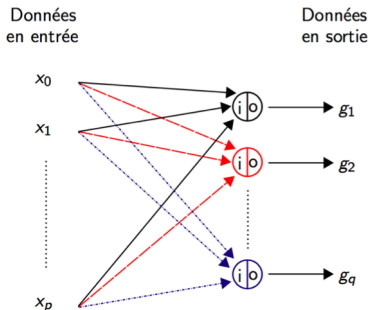


Une autre représentation bien utile



Remarques sur le perceptron

- Sans couche cachée, on retrouve les modèles linéaires
- Avec la fonction sigmoïd, il s'agit d'une régression logistique et on interprète les sorties comme des probabilités
- Pour le problème multiclasse, on traite q problèmes binaires indépendants :



A nouveau : relations entre LR et ANN

Pour la catégorisation en $q = 2$ classes, on utilise un seul perceptron de sortie et la fonction sigmoïde :

$$g(x) = h(s(x)) = \frac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x})}$$

La régression logistique polytomique est équivalente à un ANN avec q perceptrons en parallèle et des fonctions d'activation softmax :

$$g_k(x) = h(\mathbf{w}_k^T \cdot \mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \cdot \mathbf{x})}{\sum_j \exp(\mathbf{w}_j^T \cdot \mathbf{x})}$$

Le softmax permet de garantir une probabilité en sortie.

Cas de la fonction booléenne “AND”

X^1	X^2	AND
1	1	1
1	0	0
0	1	0
0	0	0

Cas de la fonction booléenne “AND”

X^1	X^2	AND
1	1	1
1	0	0
0	1	0
0	0	0

Signal post-synaptique et fonction d'Heaviside :

$$w_0 + w_1 X^1 + w_2 X^2 \stackrel{\leq}{\geq} 0$$

Cas de la fonction booléenne “AND”

X^1	X^2	AND
1	1	1
1	0	0
0	1	0
0	0	0

Signal post-synaptique et fonction d'Heaviside :

$$\begin{cases} w_0 + w_1 X^1 + w_2 X^2 \leq 0 \\ w_0 + w_1 + w_2 > 0 \\ w_0 + w_1 \leq 0 \\ w_0 + w_2 \leq 0 \\ w_0 \leq 0 \end{cases}$$

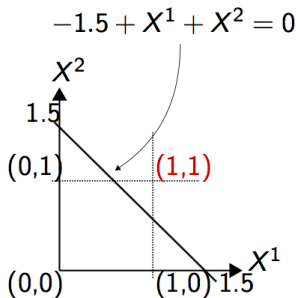
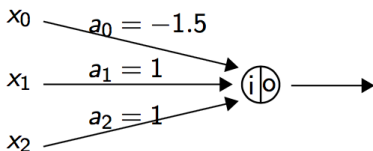
Cas de la fonction booléenne “AND”

X^1	X^2	AND
1	1	1
1	0	0
0	1	0
0	0	0

Signal post-synaptique et fonction d'Heaviside :

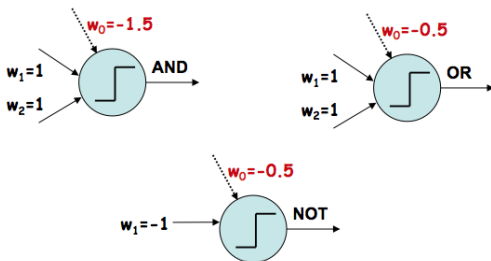
$$w_0 + w_1 X^1 + w_2 X^2 \lesseqgtr 0$$

$$\begin{cases} w_0 + w_1 + w_2 > 0 \\ w_0 + w_1 \leq 0 \\ w_0 + w_2 \leq 0 \\ w_0 \leq 0 \end{cases}$$



Note au sujet du biais w_0

L'une des motivations initiales était de pouvoir représenter des fonctions logiques :



Cas de la fonction booléenne “XOR”

X^1	X^2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

Cas de la fonction booléenne “XOR”

Signal post-synaptique et fonction d'Heaviside :

$$w_0 + w_1 X^1 + w_2 X^2 \begin{matrix} \leq \\ \geq \end{matrix} 0$$

X^1	X^2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

Cas de la fonction booléenne "XOR"

Signal post-synaptique et fonction d'Heaviside :

X^1	X^2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

$$\begin{cases} w_0 + w_1 X^1 + w_2 X^2 \leq 0 \\ w_0 + w_1 + w_2 \leq 0 \\ w_0 + w_1 > 0 \\ w_0 + w_2 > 0 \\ w_0 \leq 0 \end{cases}$$

Cas de la fonction booléenne “XOR”

Signal post-synaptique et fonction d'Heaviside :

X^1	X^2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

$$\begin{cases} w_0 + w_1 X^1 + w_2 X^2 \leq 0 \\ w_0 + w_1 + w_2 \leq 0 \\ w_0 + w_1 > 0 \\ w_0 + w_2 > 0 \\ w_0 \leq 0 \end{cases}$$

Ensemble de contraintes **incompatibles**.

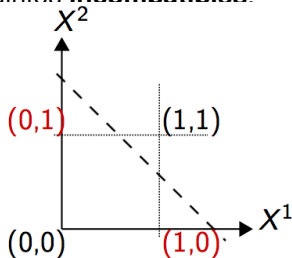
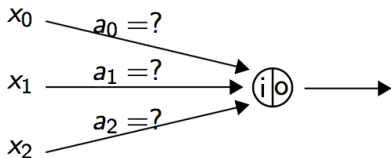
Cas de la fonction booléenne "XOR"

Signal post-synaptique et fonction d'Heaviside :

X^1	X^2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

$$\begin{cases} w_0 + w_1 X^1 + w_2 X^2 \leq 0 \\ w_0 + w_1 + w_2 \leq 0 \\ w_0 + w_1 > 0 \\ w_0 + w_2 > 0 \\ w_0 \leq 0 \end{cases}$$

Ensemble de contraintes **incompatibles**.

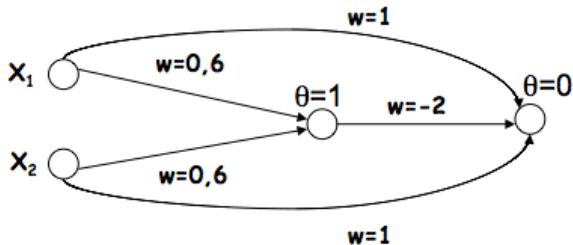


Et pourtant, une solution existe...

On peut apprendre le XOR avec un réseau à une couche cachée :

Et pourtant, une solution existe...

On peut apprendre le XOR avec un réseau à une couche cachée :



Plan

Introduction générale

Le perceptron simple

Le perceptron multicouches

Au-delà du MLP

Références

Le perceptron multicouches

- Les modèles précédents définissent des modèles linéaires avec certaines limites.

Le perceptron multicouches

- Les modèles précédents définissent des modèles linéaires avec certaines limites.
- Le perceptron multicouche (“multilayer perceptron”) est une généralisation de ces modèles :

Le perceptron multicouches

- Les modèles précédents définissent des modèles linéaires avec certaines limites.
- Le perceptron multicouche (“multilayer perceptron”) est une généralisation de ces modèles :
 - en régression il permet de traiter les cas **non linéaires** de régression

Le perceptron multicouches

- Les modèles précédents définissent des modèles linéaires avec certaines limites.
- Le perceptron multicouche (“multilayer perceptron”) est une généralisation de ces modèles :
 - en régression il permet de traiter les cas **non linéaires** de régression
 - en classification, il permet de déterminer des fonctions de décision **non linéaire** permettant de résoudre le problème “XOR” précédent par exemple

Le perceptron multicouches

- Les modèles précédents définissent des modèles linéaires avec certaines limites.
- Le perceptron multicouche (“multilayer perceptron”) est une généralisation de ces modèles :
 - en régression il permet de traiter les cas **non linéaires** de régression
 - en classification, il permet de déterminer des fonctions de décision **non linéaire** permettant de résoudre le problème “XOR” précédent par exemple
- Il consiste en l’ajout de **couches de neurones dites cachées** entre les données en entrée et les données en sortie.

Pourquoi considérer des fonctions non linéaires ?

$$y = W_2(W_1.x + b_1) + b_2$$

Pourquoi considérer des fonctions non linéaires ?

$$y = W_2(W_1 \cdot x + b_1) + b_2$$

$$y = W_2 \cdot W_1 \cdot x + W_2 \cdot b_1 + b_2$$

Pourquoi considérer des fonctions non linéaires ?

$$y = W_2(W_1.x + b_1) + b_2$$

$$y = W_2.W_1.x + W_2.b_1 + b_2$$

$$y = W.x + b$$

Pourquoi considérer des fonctions non linéaires ?

$$y = W_2(W_1.x + b_1) + b_2$$

$$y = W_2.W_1.x + W_2.b_1 + b_2$$

$$y = W.x + b$$

En conclusion

Le réseau à 2 couches est équivalent à un réseau à 1 couche linéaire

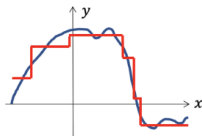
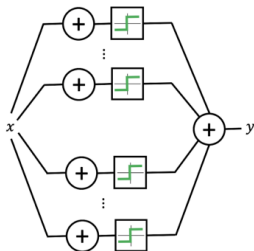
Approximation universelle

Théorème d'approximation universelle

Toute fonction continue (vérifiant certaines propriétés) peut être approximée via un MLP avec au moins une couche cachée contenant suffisamment de neurones et des fonctions d'activation non linéaires (ex. sigmoïde, mais aussi reLU).

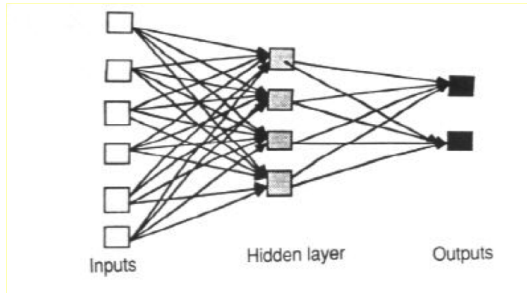
Attention : dire qu'un MLP a la “capacité” de représenter toute fonction continue ne signifie pas que l'on va être capable d' *apprendre* les paramètres du réseau de manière optimale.

Approximation universelle

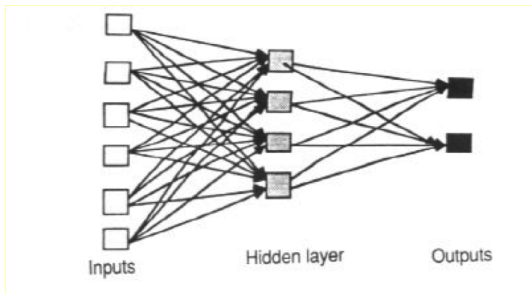


(tiré de : <https://towardsdatascience.com/geometric-foundations-of-deep-learning-94cdd45b451d>)

Le perceptron multicouches



Le perceptron multicouches

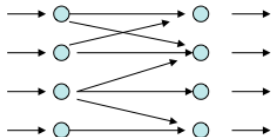


Petite démo avec TensorFlow :

<http://playground.tensorflow.org>

Architecture des réseaux de neurones

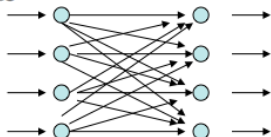
Entrées



Sorties

Connexions directes
(*feed-forward*)

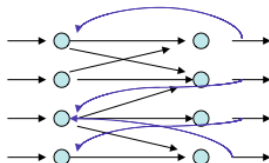
Entrées



Sorties

Connexions directes - complet

Entrées



Sorties

Connexions récurrentes

Perceptron - Apprentissage des poids

Loi de Hebb

Lorsque deux neurones sont excités conjointement, il se crée ou renforce un lien les unissant : $w'_i = w_i + \eta(\hat{y}.x_i)$

Perceptron - Apprentissage des poids

Loi de Hebb

Lorsque deux neurones sont excités conjointement, il se crée ou renforce un lien les unissant : $w'_i = w_i + \eta(\hat{y}.x_i)$

Cette loi a inspiré le perceptron de Rosenblatt qui l'adapte pour prendre en compte l'erreur observée :

$$w'_i = w_i + \eta(y - \hat{y})x_i$$

Perceptron - Apprentissage des poids

Loi de Hebb

Lorsque deux neurones sont excités conjointement, il se crée ou renforce un lien les unissant : $w'_i = w_i + \eta(\hat{y} \cdot x_i)$

Cette loi a inspiré le perceptron de Rosenblatt qui l'adapte pour prendre en compte l'erreur observée :

$$w'_i = w_i + \eta(y - \hat{y})x_i$$

Dans le perceptron de Rosenblatt, la règle passe par la fonction

d'activation : $\hat{y} = h\left(\sum_{i=0}^n w_i x_i\right)$

Avec Adaline (1960), la règle de Widrow-Hoff utilise directement la

somme pondérée des entrées : $\hat{y} = \sum_{i=0}^n w_i x_i$

Schéma général de l'algorithme pour un neurone

Pour chaque observation :

Schéma général de l'algorithme pour un neurone

Pour chaque observation :

- Phase de **propagation**
 - calculer l'activation du neurone
 - calculer la sortie de la fonction choisie

Schéma général de l'algorithme pour un neurone

Pour chaque observation :

- Phase de **propagation**
 - calculer l'activation du neurone
 - calculer la sortie de la fonction choisie
- Calcul de l'**erreur**

Schéma général de l'algorithme pour un neurone

Pour chaque observation :

- Phase de **propagation**
 - calculer l'activation du neurone
 - calculer la sortie de la fonction choisie
- Calcul de l'**erreur**
- **Mise à jour** des poids pour réduire l'erreur attendue

Schéma général de l'algorithme pour un neurone

Pour chaque observation :

- Phase de **propagation**

- calculer l'activation du neurone
- calculer la sortie de la fonction choisie

- Calcul de l'**erreur**

- **Mise à jour** des poids pour réduire l'erreur attendue

Dans le cas d'une fonction simple à seuil (*heaviside*), les règles sont simples :

- $w_{t+1} = w_t + x$ si y est positif et $\hat{y} = w_t \cdot x \leq 0$
- $w_{t+1} = w_t - x$ si y est négatif et $\hat{y} = w_t \cdot x > 0$

Illustration de l'apprentissage

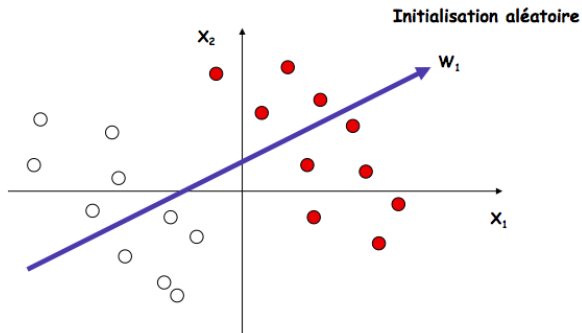


Illustration de l'apprentissage

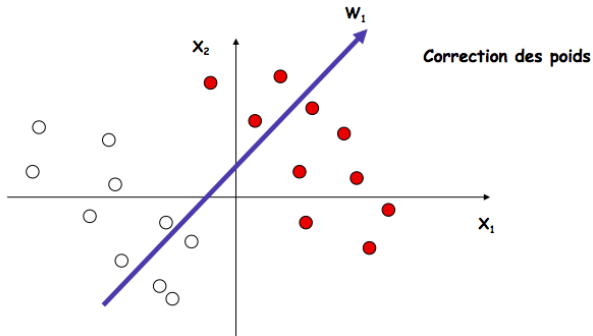


Illustration de l'apprentissage

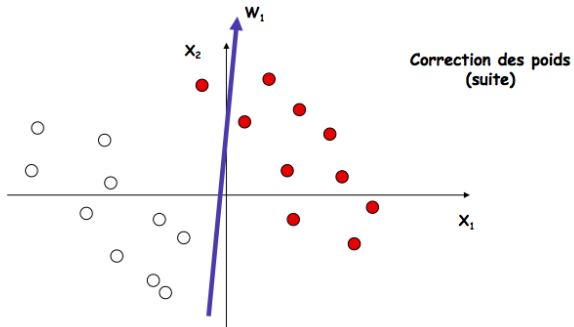
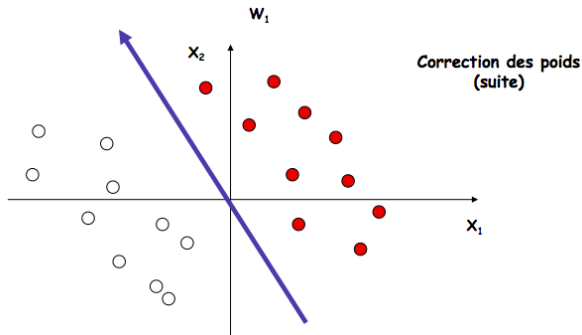
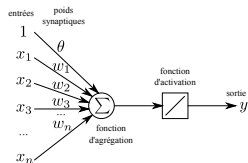


Illustration de l'apprentissage



Perceptron - Apprentissage des poids

Sortie du neurone: $\hat{y} = \sum_{i=0}^n w_i x_i$

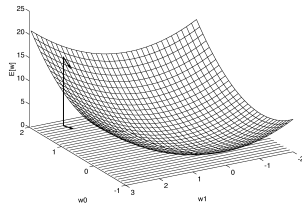
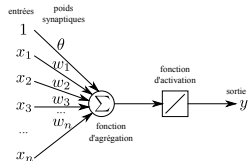


Perceptron - Apprentissage des poids

Sortie du neurone: $\hat{y} = \sum_{i=0}^n w_i x_i$

Apprentissage: On veut minimiser l'erreur

quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (y - \hat{y})^2$



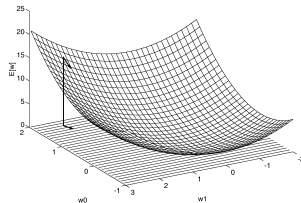
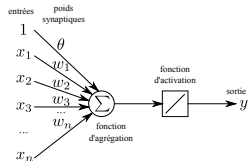
Perceptron - Apprentissage des poids

Sortie du neurone: $\hat{y} = \sum_{i=0}^n w_i x_i$

Apprentissage: On veut minimiser l'erreur

quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (y - \hat{y})^2$

$$\begin{aligned}\Delta w_i &= -\eta \frac{\partial E}{\partial w_i} \\ &= -\frac{\eta}{2} \sum_{\mathbf{x}} \frac{\partial (y - \hat{y})^2}{\partial w_i} \\ &= -\frac{\eta}{2} \sum_{\mathbf{x}} -2 \frac{\partial \hat{y}}{\partial w_i} (y - \hat{y}) = \sum_{\mathbf{x}} \eta x_i (y - \hat{y})\end{aligned}$$



Perceptron - Apprentissage des poids

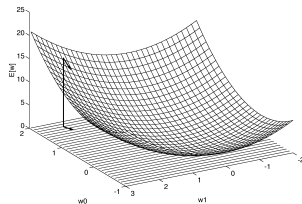
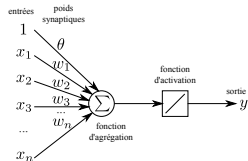
Sortie du neurone: $\hat{y} = \sum_{i=0}^n w_i x_i$

Apprentissage: On veut minimiser l'erreur

quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (y - \hat{y})^2$ par

descente de gradient stochastique (un seul exemple \mathbf{x} utilisé à la fois) :

$$\Delta \mathbf{w} = \eta \mathbf{x} (y - \hat{y})$$



Optimisation de E par descente du gradient stochastique :

$$\Delta \mathbf{w} = \eta \mathbf{x}(y - \hat{y})$$

Propriétés

- Classifieur linéaire
- Convergence garantie si η est faible (même si les données ne sont pas linéairement séparables)
- Convergence efficace (car la fonction à optimiser est quadratique)
- Convergence souvent plus rapide en mode en ligne mais pas garantie comme en mode *batch*

Pseudo-code de l'algorithme

Require: E un ensemble de données : $E = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$
 W les paramètres du réseau de neurones initialisés
 h la fonction d'activation
 η le pas d'apprentissage
repeat
 for all $e_i = (x_i, y_i)$ dans E **do**
 for all j dans $\{1, 2 \dots k\}$ **do**
 $in = w_j \cdot x_i$
 $err = y_i - h(in)$
 $w_j = w_j + \eta \cdot err \cdot h'(in) \cdot x_i$
 end for
 end for
until un certain critère de convergence

Cross-entropy vs. Mean Squared Error

$$H(p, q) = - \sum_x p(x) \log q(x)$$

Modèle num. 1 :

	\hat{y}			y			correct ?
e_1	0.3	0.3	0.4	0	0	1	oui
e_2	0.3	0.4	0.3	0	1	0	oui
e_3	0.1	0.2	0.7	1	0	0	non

Cross-entropy vs. Mean Squared Error

$$H(p, q) = - \sum_x p(x) \log q(x)$$

Modèle num. 1 :

	\hat{y}			y			correct ?
e_1	0.3	0.3	0.4	0	0	1	oui
e_2	0.3	0.4	0.3	0	1	0	oui
e_3	0.1	0.2	0.7	1	0	0	non

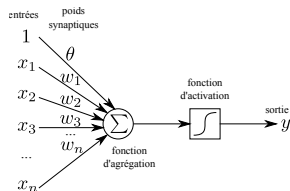
Modèle num. 2 :

	\hat{y}			y			correct ?
e_1	0.1	0.2	0.7	0	0	1	oui
e_2	0.1	0.7	0.2	0	1	0	oui
e_3	0.3	0.4	0.3	1	0	0	non

Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Sortie du neurone:

$$s = \sum_{i=0}^n w_i x_i$$
$$\hat{y} = \frac{1}{1 + e^{-s}}$$



Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Sortie du neurone:

$$s = \sum_{i=0}^n w_i x_i$$
$$\hat{y} = \frac{1}{1 + e^{-s}}$$

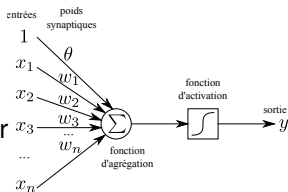
Apprentissage: On veut minimiser l'erreur

quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (y - \hat{y})^2$ par

descente de gradient stochastique:

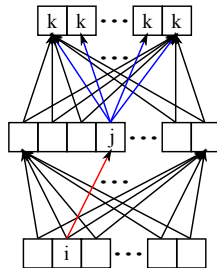
$$\Delta \mathbf{w} = \eta \mathbf{x} (y - \hat{y}) \hat{y} (1 - \hat{y})$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta \frac{\partial E}{\partial s} \frac{\partial s}{\partial w_i} = \eta \sum_{\mathbf{x}} \frac{\partial \hat{y}}{\partial s} (y - \hat{y}) \frac{\partial s}{\partial w_i} = \sum_{\mathbf{x}} \eta \hat{y} (1 - \hat{y}) (y - \hat{y}) x_i$$



Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Réseau: connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie

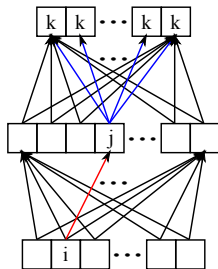


Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Réseau: connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie

Apprentissage: On veut minimiser l'erreur quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (y - \hat{y})^2$ par descente de gradient stochastique, on suppose connus les $\delta_k = -\frac{\partial E}{\partial s_k}$ de la couche supérieure

$$\begin{aligned} -\frac{\partial E}{\partial s_j} &= -\sum_k \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial s_j} \\ &= -\sum_k -\delta_k w_{jk} \hat{y}_j (1 - \hat{y}_j) \\ &= \hat{y}_j (1 - \hat{y}_j) \sum_k \delta_k w_{jk} \end{aligned}$$

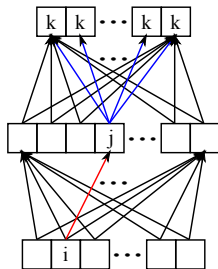


Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Réseau: connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie

Apprentissage: On veut minimiser l'erreur quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (y - \hat{y})^2$ par descente de gradient stochastique, on suppose connus les $\delta_k = -\frac{\partial E}{\partial s_k}$ de la couche supérieure

$$\begin{aligned} -\frac{\partial E}{\partial s_j} &= -\sum_k \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial s_j} \\ &= -\sum_k -\delta_k w_{jk} \hat{y}_j (1 - \hat{y}_j) \\ &= \hat{y}_j (1 - \hat{y}_j) \sum_k \delta_k w_{jk} \end{aligned}$$

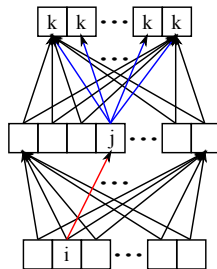


Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Réseau: connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie

Apprentissage: Pour chaque entrée reçue:

1. Calculer la sortie \hat{y} du réseau par propagation (couche par couche) de l'activité
2. Calculer l'erreur de la couche de sortie:
$$\delta_k = \hat{y}_k(1 - \hat{y}_k)(y_k - \hat{y}_k)$$
3. Rétropropager l'erreur à travers chaque couche j du réseau
$$\delta_j = \hat{y}_j(1 - \hat{y}_j) \sum_k \delta_k w_{jk}$$
4. Modifier chaque poids $\Delta w_{ij} = \eta \delta_j x_i$

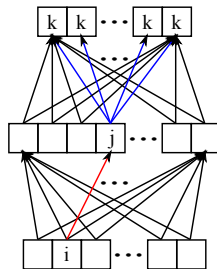


Perceptron multi-couches - Werbos & Rumelhard (1984-1986)

Réseau: connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie

Apprentissage: Pour chaque entrée reçue:

1. Calculer la sortie \hat{y} du réseau par propagation (couche par couche) de l'activité
2. Calculer l'erreur de la couche de sortie:
$$\delta_k = \hat{y}_k(1 - \hat{y}_k)(y_k - \hat{y}_k)$$
3. Rétropropager l'erreur à travers chaque couche
 j du réseau
$$\delta_j = \hat{y}_j(1 - \hat{y}_j) \sum_k \delta_k w_{jk}$$
4. Modifier chaque poids $\Delta w_{ij} = \eta \delta_j x_i$



Plan

Introduction générale

Le perceptron simple

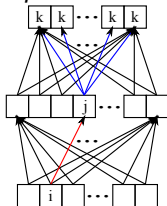
Le perceptron multicouches

Au-delà du MLP

Références

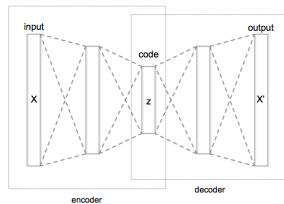
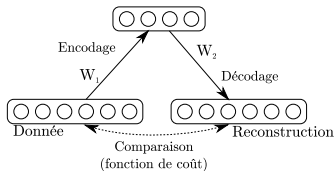
Au-delà du MLP

- *Perceptron / Multi-layer perceptron*



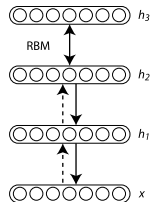
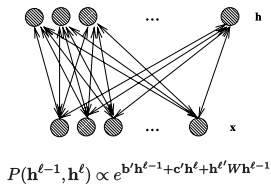
Au-delà du MLP

- *Perceptron / Multi-layer perceptron*
- *Auto-encoder / Stacked auto-encoder*



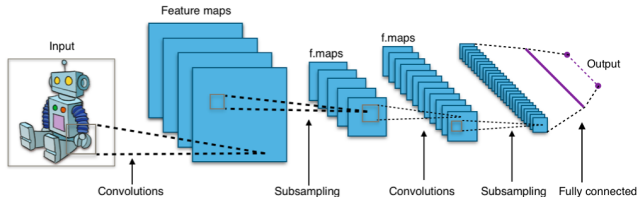
Au-delà du MLP

- *Perceptron / Multi-layer perceptron*
- *Auto-encoder / Stacked auto-encoder*
- *Restricted Boltzmann machine / Deep belief network*



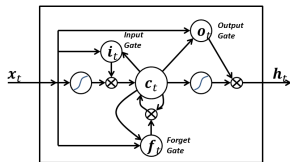
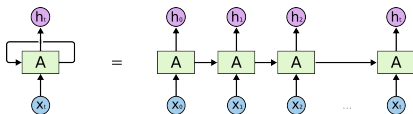
Au-delà du MLP

- *Perceptron / Multi-layer perceptron*
- *Auto-encoder / Stacked auto-encoder*
- *Restricted Boltzmann machine / Deep belief network*
- *Convolutional neural network*



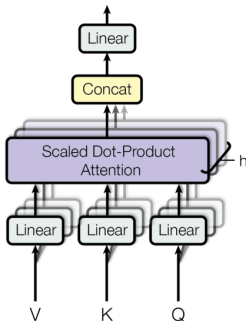
Au-delà du MLP

- *Perceptron / Multi-layer perceptron*
- *Auto-encoder / Stacked auto-encoder*
- *Restricted Boltzmann machine / Deep belief network*
- *Convolutional neural network*
- *Recurrent neural network*



Au-delà du MLP

- *Perceptron / Multi-layer perceptron*
- *Auto-encoder / Stacked auto-encoder*
- *Restricted Boltzmann machine / Deep belief network*
- *Convolutional neural network*
- *Recurrent neural network*
- *Attention models*



Plan

Introduction générale

Le perceptron simple

Le perceptron multicouches

Au-delà du MLP

Références

Quelques références

- Deep Learning. Ian Goodfellow, Yoshua Bengio and Aaron Courville. MIT Press, 2016.
- Cours de Y. Lecun à NYU : <https://cds.nyu.edu/deep-learning/>
- Speech and Language Processing (3rd ed. draft). Dan Jurafsky and James H. Martin. October 16, 2019.
- Neural network methods for natural language processing. Goldberg, Y., Synthesis Lectures on Human Language Technologies, 10(1), 1-309, 2017.