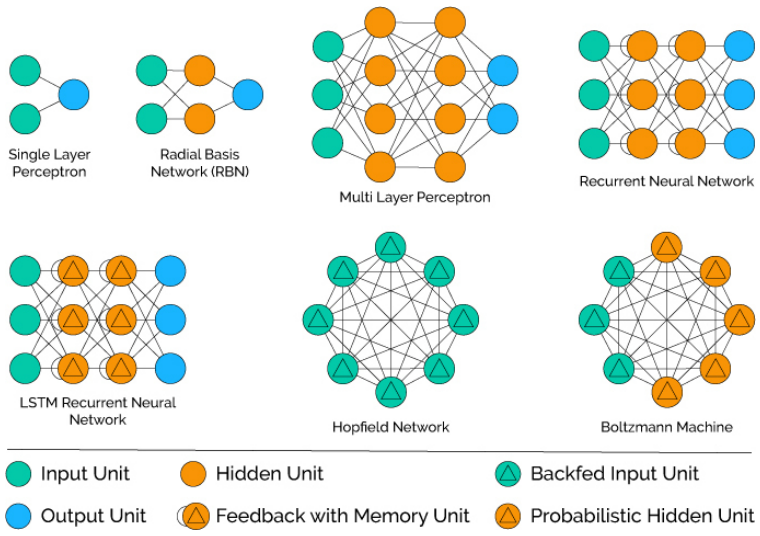
Les réseaux de neurones



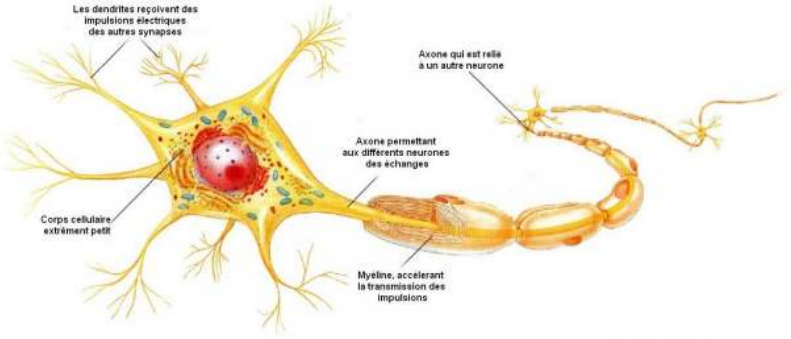
Les réseaux de neurones

- **Connexionisme** : modélise les phénomènes mentaux ou comportementaux comme des processus émergents de réseaux d'unités simples interconnectées.
- Modélisation mathématique du cerveau humain
- Réseaux de neurones formels = réseaux d'unités de calcul élémentaires interconnectées
- 2 axes de recherche :
  - étude et modélisation des phénomènes naturels d'apprentissage (biologie, physiologie du cerveau)
  - algorithmes pour résoudre des problèmes complexes
- Applications
  - statistiques : analyse de données / prévision / classification
  - robotique : contrôle et guidage de robots ou de véhicules autonomes
  - imagerie / reconnaissance de formes
  - traitement du signal
  - simulation de l'apprentissage

Outils / Bibliothèques

- Outils commerciaux :
  1. Matlab : “neural networks" toolbox
     `http://www.mathworks.com/products/neuralnet/`
  2. Wolfram Mathematica
     `https://www.wolfram.com/mathematica/`
  3. NeuralDesigner
     `https://www.neuraldesigner.com/`
- Outils open-source / gratuits :
  1. JOONE : bibilothèque JAVA open source (licence LGPL)
     `http://www.jooneworld.com/`
  2. Scilab : ANN toolbox
     `http://www.scilab.org/`
     `https://atoms.scilab.org/toolboxes/ANN_Toolbox`
  3. Scikit-Learn : Neural network models
  4. tensorFlow + Keras : deep learning

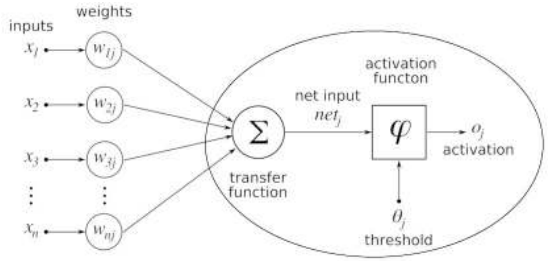
Le neurone biologique



Comparison of standard computer (CIRCA 2020) and human brain

	Computer	**Human Brain**
Computational units	1 CPU, 10^{10} Gates	10^{11} Neurons
Storage units	10^{10} bytes RAM, 10^{13} bytes disk	10^{11} Neurons, 10^{14} Synapses
Cycle time	10^{-9} Sec	10^{-3} sec
Bandwidth	10^{9} bits/sec	10^{14} bits/Sec
Neuron updates/sec	10^{5}	10^{14}
power	100 watts	10 watts

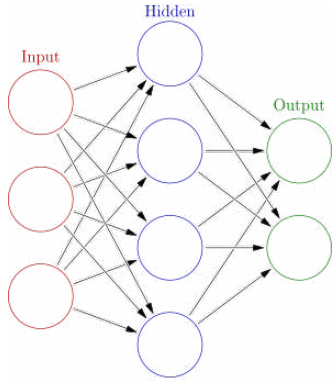
Un neurone artificiel simple : le perceptron



- modèle abstrait TRES simplifié d'un neurone biologique
- élément de base servant à effectuer des calculs, appelé nœud ou unité/unit
- à chaque entrée x est associé un poids w qui peut être modifié
- les entrées x correspondent à des signaux provenant d'axones d'autres neurones
  x_0 - les "biais" sont des entrées "spéciales", avec un poids w_0
- Les poids w correspondent aux modulations synaptiques (*i.e.* qqch comme la force/le nombre des neurotransmetteurs)
- La somme correspond au soma de la cellule
- la fonction d'activation correspond à l'activation du signal envoyé le long de l'axone
- Ainsi, la sortie $y = f(z)$ correspond au signal sur l'axone.

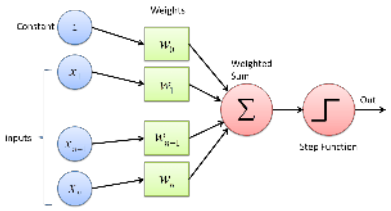
Réseaux de neurones artificiels : feedforward nets

- Réseaux interconnectés composés d'unités simples ("neurones artificiels").
  - le poids w_{ij} est le poids de la $i^{\text{ème}}$ entrée dans l'unité j.
  - Les neurones de sortie permettent de déterminer la classe. (décision qui dépend de l'entrée et des poids)
  - S'il y a plusieurs unités de sorties qui sont activées, on choisit alors l'unité avec la plus grande valeur.
- L'apprentissage consiste à ajuster les poids du réseau...
  - pour que la sortie attendue soit produite dès qu'une instance de l'ensemble d'apprentissage est passée en entrée du réseau.

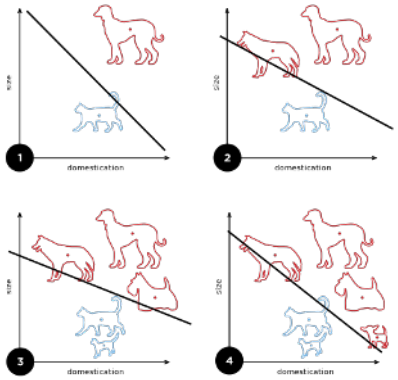


Unité neuronale de base (Single Perceptron Unit)

- Commençons par regarder le **perceptron** (unité neuronale de base) :



- soit $x_0 = 1$ et $w_0 = b$
  alors l'équation de l'hyperplan est
  $$w.x = 0$$
  c.a.d.
  $$\sum_{j=0}^{n} w_j x_j = 0$$
- $h(x) = sign(\sum_{j=1}^{n} w_j x_j + b)$
  $h(x) = sign(w.x + b)$

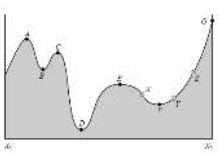


Algorithme pour le Perceptron - Rosenblatt, 1956

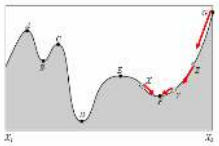
1. Initialiser les poids et le seuil. Les poids peuvent être initialisés à 0 ou à de petites valeurs aléatoires.
2. Pour chaque exemple j de l'ensemble d'entraînement D, effectuer les opérations suivantes :
   - Calculer la sortie :
     $$\begin{aligned} y_j(t) &= f[\mathbf{w}(t) \cdot \mathbf{x}_j] \\ &= f[w_0(t)x_{j,0} + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}] \end{aligned}$$
   - mettre à jour les poids :
     $w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))x_{j,i}$, pour toutes les caractéristiques $0 \leq i \leq n$
     r est appelé le taux d'apprentissage (learning rate).
3. Pour un apprentissage "offline", la seconde étape peut être répétée jusqu'à ce que l'erreur $\frac{1}{s}\sum_{j=1}^{s} |d_j - y_j(t)|$ soit inférieure à un seuil prédéfini (par l'utilisateur) γ, où jusqu'à un nombre prédéterminé d'itérations (s est la taille de l'ensemble d'apprentissage).

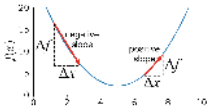
Descente de Gradient - 1D

- il est facile de trouver un minimum local. Il est difficile trouver un minimum absolu

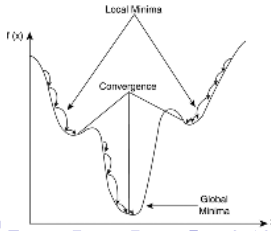


- idée :
  On part d'un point X_0 donné par l'utilisateur
  On descend le long de la plus grande pente locale



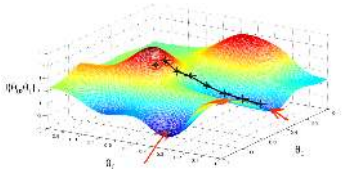


- On connaît F et $F'(X)$ - On part d'un point de départ X_0 - On calcule la suite $X_{k+1} = x_k - \alpha_k g_k$ où :
  $g_k = F'(x_k)$
  α_k est le pas de descente

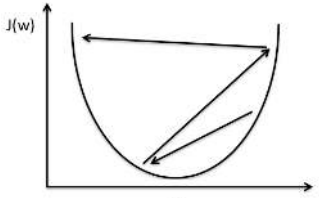


Descente de Gradient - cas multidimensionnel

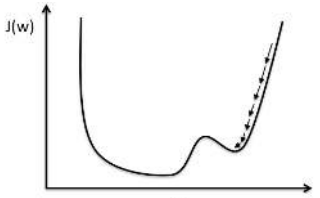
- vecteur gradient :
  $$\nabla_w f = \left[\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_n}\right]^t$$
- pour minimiser $f(w)$
  - choisir un point w initial
  - Changer w en $w - \alpha \nabla_w f$
  - jusqu'à ce que f se stabilise ($\nabla_w f \approx 0$)



- Cette technique trouve un minimum local sauf si la fonction est globalement convexe...
- si f n'est pas linéaire, le taux α doit être choisi avec soin :
  - trop petit → convergence TRÈS lente
  - trop grand → oscillation



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

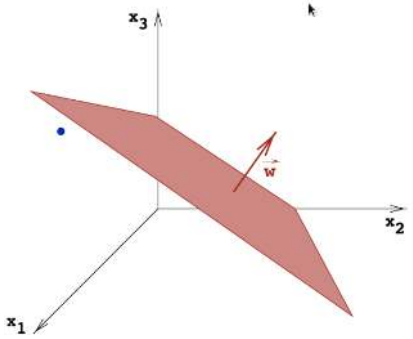
Perceptron : Algorithme d'apprentissage

Erreur sur un positif

Erreur de classification de $(x, +1)$

$b + <w, x> \; < \; 0$

- Comment peut-on modifier b et w pour remédier à cette erreur ?
  → augmenter $b + <w, x>$



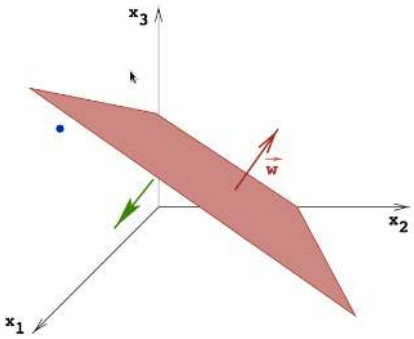
Perceptron : Algorithme d'apprentissage

Erreur sur un positif

Erreur de classification de $(x, +1)$

$b + <w, x> \; < \; 0$

- Comment peut-on modifier b et w pour remédier à cette erreur ?
  → augmenter $b + <w, x>$
  - augmenter b



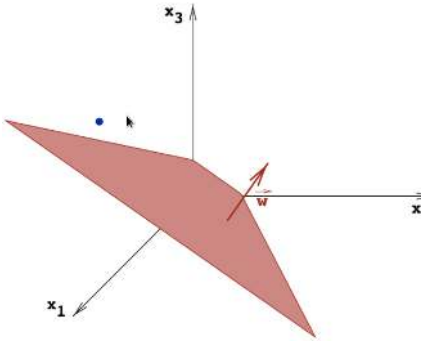
Perceptron : Algorithme d'apprentissage

Erreur sur un positif

Erreur de classification de $(x, +1)$

$b + <w, x> \ < \ 0$

- Comment peut-on modifier b et w pour remédier à cette erreur ?
  $\rightarrow$ augmenter $b + <w, x>$
  - augmenter b



CÔTE D'AZUR

Perceptron : Algorithme d'apprentissage

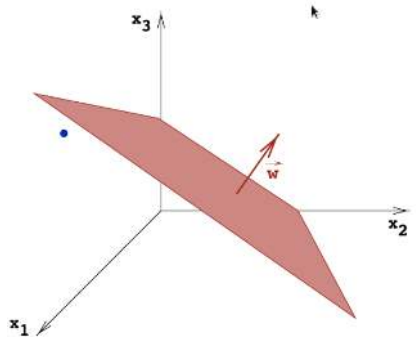
AI for bio
J-P Comet
Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learling
Evaluation
Data Mining
Project

Erreur sur un positif

Erreur de classification de $(x, +1)$

$b + <w, x> \ < \ 0$

- Comment peut-on modifier b et w pour remédier à cette erreur ?
  $\rightarrow$ augmenter $b + <w, x>$
  - augmenter b



175/227

CÔTE D'AZUR

Perceptron : Algorithme d'apprentissage

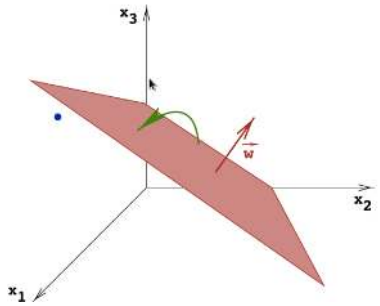
AI for bio
J-P Comet
Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learling
Evaluation
Data Mining
Project

Erreur sur un positif

Erreur de classification de $(x, +1)$

$b + <w, x> \ < \ 0$

- Comment peut-on modifier b et w pour remédier à cette erreur ?
  $\rightarrow$ augmenter $b + <w, x>$
  - augmenter b
  - Si $x_i > 0$ augmenter w_i
  - Si $x_i < 0$ diminuer w_i



175/227

CÔTE D'AZUR

Perceptron : Algorithme d'apprentissage

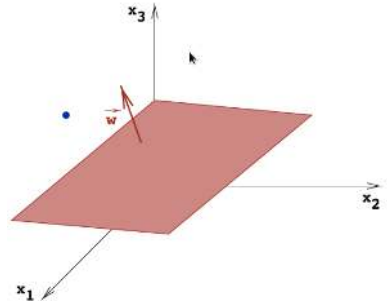
AI for bio
J-P Comet
Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learling
Evaluation
Data Mining
Project

Erreur sur un positif

Erreur de classification de $(x, +1)$

$b + <w, x> \ < \ 0$

- Comment peut-on modifier b et w pour remédier à cette erreur ?
  $\rightarrow$ augmenter $b + <w, x>$
  - augmenter b
  - Si $x_i > 0$ augmenter w_i
  - Si $x_i < 0$ diminuer w_i



175/227

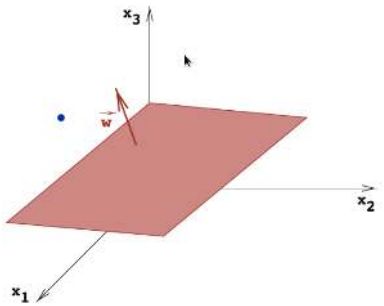
Perceptron : Algorithme d'apprentissage

Erreur sur un positif

Erreur de classification de $(x, +1)$

$b + <w, x> \ < \ 0$

- Comment peut-on modifier b et w pour remédier à cette erreur ?
  $\rightarrow$ augmenter $b + <w, x>$
  - augmenter b
  - Si $x_i > 0$ augmenter w_i
  - Si $x_i < 0$ diminuer w_i



- Algorithme : ajouter x à w et 1 à b

Erreur sur un négatif

Erreur de classification de $(x, -1)$

$b + <w, x> \ > \ 0$

Perceptron : données complétées

- Pour simplifier les calculs, on utilise souvent (toujours !) une astuce mathématique pour ne pas traîner tout le temps le scalaire b.
- Idée : Rajouter une dimension
  si nos données sont de dimension d, donc dans $\mathbb{R}^d$, on va faire comme si elles étaient dans $\mathbb{R}^{d+1}$ et qu'elles avaient toutes 1 en dernière coordonnée.
- Du coup, on considère que w est aussi de dimension $d+1$ et la dernière coordonnées de w est b.
- Mathématiquement

$$\begin{aligned} f(x) = sgn(b + <w, x>) &= sgn(b + \sum_{i=1}^{d} w_i x_i) \\ &= sgn(+b.1 + \sum_{i=1}^{d} w_i x_i) \\ &= sgn(\sum_{i=1}^{d+1} w_i x_i) \text{ avec } w_{d+1} = b \text{ et } x_{d+1} = 1 \\ &= sgn(<(w_1, ..., w_d, b), (x_1, ..., x_d, 1)>) \end{aligned}$$

- On dit que les données sont complétées si elles sont dans $\mathbb{R}^{d+1}$

Perceptron : apprentissage *via* descente de gradient

- Maximiser la marge des points mal classés

$$\begin{aligned} f(w) &= \sum_{i \text{ mal classé}} y^i w.x^i \\ \nabla_w f &= \sum_{i \text{ mal classé}} y^i x^i \end{aligned}$$

- Entraînement Off-line : calculer le gradient comme la somme sur tous les points de l'ensemble d'apprentissage
- Entraînement On-line : approximer le gradient par l'un des termes dans la somme : $y^i x^i$
  L'entraînement On-line peut être vu comme une approximation aléatoire du gradient en chaque point (souvent appelé "gradient stochastique") ce qui peut aider à sortir d'un minimum local.

Perceptron : Algorithme - Rosenblatt, 1956

- choisir un vecteur de poids initial (y compris b)
- Répéter jusqu'à ce que tous les points soient correctement classés :
  - Répéter pour chaque point :
    - Calculer la "marge" $(y^i w.x^i)$ pour le point i
    - si la marge > 0, le point est correctement classé
    - sinon changer les poids pour augmenter la marge ; changer le poids proportionnellement à $y^i x^i$

Noter que si $y^i = 1$
si $x^i_j > 0$, alors w_j augmente
si $x^i_j < 0$, alors w_j diminue
et similairement pour $y^i = -1$

Garantit de trouver un hyperplan solution s'il en existe un - si les données sont séparables linéairement - sinon, l'algorithme boucle ...

Algorithme d'apprentissage du Perceptron

Algorithme d'apprentissage du Perceptron
Entrée : $S = (x_1, y_1), ..., (x_n, y_n)$, un échantillon $\mathbb{R}^{d+1} \times \{-1, +1\}$ complété et linéairement séparable

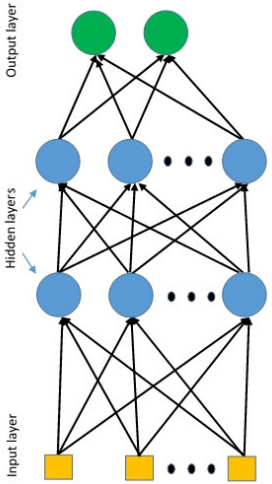
$w = w_0 \in \mathbb{R}^{d+1}$
Répéter
  Pour $i = 1$ **à** n
    Si $y_i.<w, x_i> \leq 0$ **alors**
      $w = w + y^i x^i$
  FinPour
Jusqu'à ce qu'il n'y ait plus d'erreurs

Sortie : w

- L'algorithme du Perceptron est une procédure **on-line**, par **correction d'erreurs (error-driven)**.
- L'algorithme est **correct :** lorsqu'il converge, l'hyperplan résultat sépare bien les données fournies en entrée
- L'algorithme est **complet :** si l'ens d'apprentissage est linéairement séparable, l'algorithme **converge.**
- Dans le pire des cas, le nombre d'itérations est égal à $(n+1)^2 2^{(n+1)log(n+1)}$. **Complexité exponentielle !**
- En pratique, on se limite souvent à un nombre d'itérations fixé par avance.
- Très mauvaise tolérance au bruit...

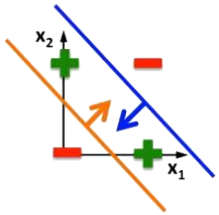
Perceptron Multi-couche

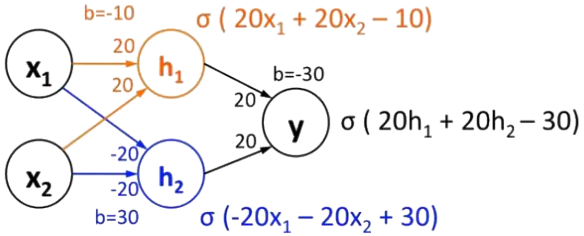
- Et si le problème n'est pas linéairement séparable ?
- Solution : Combiner de multiples séparateurs linéaires.
- Introduction d'unités dites "cachées" dans les RN les rend beaucoup plus puissants :
- ils ne sont plus limités à des problèmes linéairement séparables.
- Les premières "couches" (layers) transforment le problème en problèmes traitables plus facilement par les couches d'après.



Multi-Layer Perceptron (MLP) : le problème XOR

Linear classifiers cannot solve this





σ(20*0 + 20*0 – 10) ≈ 0
σ(20*1 + 20*1 – 10) ≈ 1
σ(20*0 + 20*1 – 10) ≈ 1
σ(20*1 + 20*0 – 10) ≈ 1

σ (-20*0 – 20*0 + 30) ≈ 1
σ (-20*1 – 20*1 + 30) ≈ 0
σ (-20*0 – 20*1 + 30) ≈ 1
σ (-20*1 – 20*0 + 30) ≈ 1

σ (20*0 + 20*1 – 30) ≈ 0
σ (20*1 + 20*0 – 30) ≈ 0
σ (20*1 + 20*1 – 30) ≈ 1
σ (20*1 + 20*1 – 30) ≈ 1

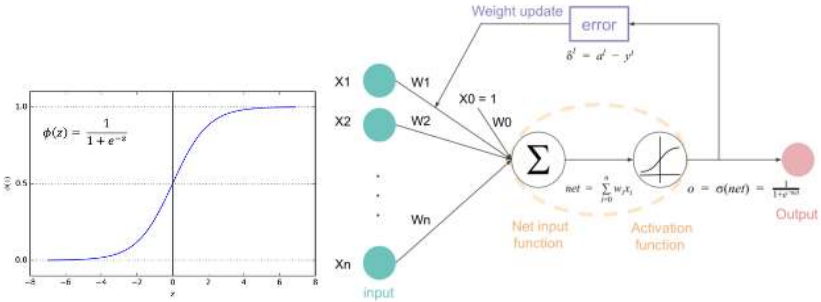
Multi-Layer Perceptron

- Comment apprendre les paramètres du MLP ?
- La descente de Gradient !
- Cependant, la fonction du réseau est discontinue.
  Un petit changement des poids (biais) peut causer un changement drastique de la sortie...
  Difficile d'utiliser la descente de gradient...
- Quelle est la source de cette discontinuité ?
  - sortie du perceptron : $y(x) = \sigma(\sum_j w_j x_j - b)$
    où $\sigma(z)$ est la fonction escalier
- Peut-on remplacer cette fonction σ par une fonction dérivable ?
  $\Rightarrow$ Utiliser une fonction à seuil souple (smooth threshold function)

Soft Threshold

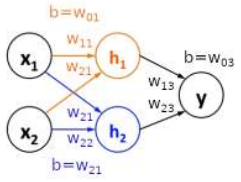
- Peut on entraîner un MLP avec des techniques de descente de gradient ?
- Non car la sortie n'est pas une fonction continue des paramètres du système (poids w et seuil b).
- Dans un perceptron, peu importe la distance d'un point à la frontière de décision, la fonction de sortie sera 0 ou 1...
- Il faut introduire une sortie continue (fonction continue par rapport aux poids et seuils)

- **Unité Sigmoïde**
  - Utilisée communément dans les réseaux neuronaux
  - la fonction dite **logistique** :
  - la valeur z est aussi appelée l'"activation" d'un neurone.



Entraînement du réseau (Training)

- Propriété TRÈS importante : la sigmoïde est différentiable.
  $\Rightarrow$ On peut utiliser des méthodes basées sur le gradient pour la minimisation de la phase d'apprentissage.
- La sortie d'un MLP avec unités sigmoïdales est une fonction de 2 vecteurs, les entrées (x) et les poids (w).
  $\Rightarrow$ Pendant la phase d'entraînement, les instances de l'ensemble d'entraînement sont considérés fixées.
- La sortie de cette fonction (y) varie continuement avec les poids.



$$y = s\Big(\underbrace{w_{13}\,s\big(\overbrace{w_{11}x_1 + w_{21}x_2 + w_{01}}^{z_1}\big) + w_{23}\,s\big(\overbrace{w_{12}x_1 + w_{22}x_2 + w_{02}}^{z_2}\big) + w_{03}}_{z_3}\Big)$$

Entraînement du réseau (Training)

- $y(x, w)$ où w est un vecteur de poids et x est un vecteur d'entrée
- y^m est la sortie désirée.
  Erreur sur l'ensemble d'entraînement pour un vecteur de poids donné :

$$E = \frac{1}{2}\sum_i (y(x^m, w) - y^m)^2$$

le coefficient $\frac{1}{2}$ n'est là que pour simplifier les dérivées.

- Notre objectif : trouver un vecteur de poids qui minimise l'erreur

Descente de Gradient

- Version On-line : on considère à chaque fois seulement l'erreur d'une seule donnée en entrée

$$E = \frac{1}{2}(y(x^m, w) - y^m)^2$$

- On suit la descente de gradient. Le gradient de l'erreur d'entraînement est calculé comme une fonction des poids.

pour simplifier, on note $y(x^m, w)$ simplement par y.

$$\nabla_w E = (y(x^m, w) - y^m)\nabla_w y(x^m, w)$$

où $\nabla_w y(x^m, w) = \left[\frac{\partial y}{\partial w_1}, ..., \frac{\partial y}{\partial w_n}\right]^t$

- On modifie w comme suit :

$$w \leftarrow w - \eta \nabla_w E \quad \Leftrightarrow \quad w \leftarrow w - \eta(y(x^m, w) - y^m)\left[\frac{\partial y}{\partial w_1}, ..., \frac{\partial y}{\partial w_n}\right]^t$$

Descente de Gradient - une seule Unité (1)

Gradient de la sortie : $\nabla_w y(x^m, w) = \left[\frac{\partial y}{\partial w_1}, ..., \frac{\partial y}{\partial w_n}\right]^t$

$$z = \sum_{i=0}^{n} w_i x_i^m \qquad y = s(z) = \frac{1}{1+e^{-z}}$$

$$\begin{aligned}\frac{\partial y}{\partial w_i} &= \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} \\ &= \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_i} \\ &= \frac{\partial s(z)}{\partial z} x_i^m\end{aligned}$$

$$w \leftarrow w - \eta(y(x^m, w) - y^m)\left[\frac{\partial y}{\partial w_1}, ..., \frac{\partial y}{\partial w_n}\right]^t$$

$$w_i \leftarrow w_i - \eta(y - y^m)\frac{\partial s(z)}{\partial z} x_i^m$$

En notant :

$$\delta = \frac{\partial E}{\partial z} = (y - y^m)\frac{\partial s(z)}{\partial z}$$

on a : $\Delta w_i = -\eta \delta x_i^m$ Delta rule

Descente de Gradient - une seule Unité (2)

Dérivée de la sigmoïde :

$$\begin{aligned}s(z) &= \frac{1}{1+e^{-z}} \\ \frac{ds(z)}{dz} &= \frac{d}{dz}\left[(1+e^{-z})^{-1}\right] \\ &= [-(1+e^{-z})^{-2}][-e^{-z}] \\ &= \left[\frac{1}{1+e^{-z}}\right]\left[\frac{e^{-z}}{1+e^{-z}}\right] \\ &= s(z)(1-s(z)) \\ &= y(1-y)\end{aligned}$$

En suivant l'équation du transparent précédent :

$$\begin{aligned}\Delta w_i &= -\eta \delta x_i^m \\ &= -\eta(y - y^m)\frac{\partial s(z)}{\partial z} x_i^m \\ &= -\eta(y - y^m)y(1-y)x_i^m \\ &= -\eta y(1-y)(y - y^m)x_i^m\end{aligned}$$

Descente de Gradient - Unités de sortie

$\nabla_w y(x^m, w) = \left[\frac{\partial y}{\partial w_1}, ..., \frac{\partial y}{\partial w_n}\right]^t$

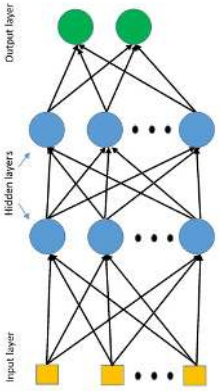
$z = \sum_{i=0}^{n} w_i x_i \qquad y = s(z) = \frac{1}{1+e^{-z}}$

$$\begin{aligned}\frac{\partial y}{\partial w_i} &= \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} \\ &= \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_i} \\ &= \frac{\partial s(z)}{\partial z} x_i\end{aligned}$$

cas de base

$$\begin{aligned}\frac{\partial y}{\partial w_{ji}} &= \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_{ji}} \\ &= \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_{ji}} \\ &= \frac{\partial s(z)}{\partial z} w_j \frac{\partial y_j}{\partial w_{ji}}\end{aligned}$$

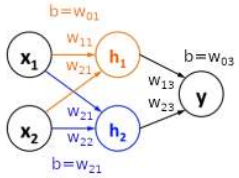
Récursion



Gradient de l'erreur

$$E = \frac{1}{2}\sum_i (y(x^m, w) - y^{i^*})^2$$

$$y = s\Big(\underbrace{w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 + w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 + w_{02}}^{z_2}) + w_{03}}_{z_3}\Big)$$



CÔTE D'AZUR

AI for bio

J-P Comet

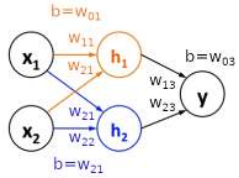
Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learing
Evaluation
Data Mining
Project

Gradient de l'erreur

$$E = \frac{1}{2}\sum_i (y(x^m, w) - y^{i^*})^2$$

$$y = s\Big(\underbrace{w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 + w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 + w_{02}}^{z_2}) + w_{03}}_{z_3}\Big)$$

$$\frac{\partial E}{\partial w_j} = (y - y^{i^*})\frac{\partial y}{\partial w_j}$$



CÔTE D'AZUR

AI for bio

J-P Comet

Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learing
Evaluation
Data Mining
Project

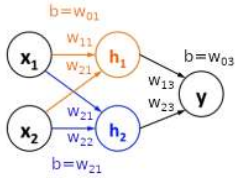
Gradient de l'erreur

$$E = \frac{1}{2}\sum_i (y(x^m, w) - y^{i^*})^2$$

$$y = s\Big(\underbrace{w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 + w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 + w_{02}}^{z_2}) + w_{03}}_{z_3}\Big)$$

$$\frac{\partial E}{\partial w_j} = (y - y^{i^*})\frac{\partial y}{\partial w_j}$$

$$\frac{\partial y}{\partial w_{13}} = \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_{13}} = \frac{\partial y}{\partial z_3}y_1$$



CÔTE D'AZUR

AI for bio

J-P Comet

Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learing
Evaluation
Data Mining
Project

Gradient de l'erreur

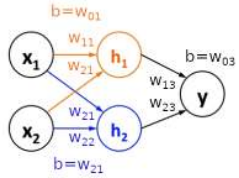
$$E = \frac{1}{2}\sum_i (y(x^m, w) - y^{i^*})^2$$

$$y = s\Big(\underbrace{w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 + w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 + w_{02}}^{z_2}) + w_{03}}_{z_3}\Big)$$

$$\frac{\partial E}{\partial w_j} = (y - y^{i^*})\frac{\partial y}{\partial w_j}$$

$$\frac{\partial y}{\partial w_{13}} = \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_{13}} = \frac{\partial y}{\partial z_3}y_1$$

$$\frac{\partial y}{\partial w_{11}} = \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_{11}} = \frac{\partial y}{\partial z_3}\left(w_{13}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{11}}\right) = \frac{\partial y}{\partial z_3}\left(w_{13}\frac{\partial y_1}{\partial z_1}x_1\right)$$



Apprentissage des poids d'un MLP (1)

- $z_b = \sum_a y_a w_{a\to b}$ le potentiel du neurone b
- $y_b = \sigma(z_b)$ où $\sigma(.)$ est la fonction d'activation du neurone b.
- $E = \frac{1}{2}(\text{sortie attendue} - y_b)^2$
- Pour tout neurone b, on a :

$$\frac{\partial E}{\partial w_{a\to b}} = \frac{\partial E}{\partial z_b}\frac{\partial z_b}{\partial w_{a\to b}} = \frac{\partial E}{\partial z_b} y_a$$

- Reste à calculer ce terme $\frac{\partial E}{\partial z_b}$.
  - pour le neurone de sortie : $\frac{\partial E}{\partial z_b} = \frac{\partial E}{\partial y_b}\frac{\partial y_b}{\partial z_b}$
    - $\frac{\partial E}{\partial y_b}$ peut se réécrire :

$$\begin{aligned}\frac{\partial E}{\partial y_b} &= \frac{\partial}{\partial y_b}\frac{1}{2}(\text{sortie attendue} - y_b)^2 \\ &= -(\text{sortie attendue} - y_b)\end{aligned}$$

    - $\frac{\partial y_b}{\partial z_b}$: c'est simplement la dérivée de la fonction d'activation.

  Ainsi, $\frac{\partial E}{\partial z_b} = -y_b(1-y_b)(\text{sortie attendue} - y_b)$ pour un neurone de sortie.

Apprentissage des poids d'un MLP (2)

Pour un neurone d'une couche cachée :
- on suppose que l'on a calculé $\frac{\partial E}{\partial z_b}$ pour la couche $c+1$.
- utile pour calculer $\frac{\partial E}{\partial z_b}$ pour les neurones de la couche c

Ainsi, on commence par calculer ce terme pour la couche juste avant la couche de sortie, puis remonter ainsi vers la première couche cachée.
Soit donc un neurone b d'une couche cachée :

$$\frac{\partial E}{\partial z_b} = \sum_{k:\text{neuronne de la couche d'après}} \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial z_b} = \sum_k \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial y_b}\frac{\partial y_b}{\partial z_b}$$

dans laquelle :

- $\frac{\partial y_b}{\partial z_b}$: dérivée de la fonction d'activation du neurone b ;
- $\frac{\partial z_k}{\partial y_b}$:

$$\begin{aligned} z_k &= \sum_{b':\text{neurones de la couche précédente}} w_{b'\to k} y_{b'} \\ z_k &= w_{b\to k} y_b + \sum_{b':\text{autres neurones de la couche précédente}} w_{b'\to k} y_{b'} \end{aligned}$$

  y_b apparaît une seule fois. Donc $\boxed{\frac{\partial z_k}{\partial y_b} = w_{b\to k}}$

- $\frac{\partial E}{\partial z_k}$: c'est la dérivée de E par rapport au potentiel du neurone k qui se situe dans la couche suivante, donc plus proche de la sortie. Ce terme a déjà été calculé précédemment.

Apprentissage des poids d'un MLP (3)

- Ainsi on a :

Neurone de sortie :

$$\frac{\partial E}{\partial z_b} = -y_b(1-y_b)\times(\text{sortie attendue} - y_b)$$

Neurone caché :

$$\begin{aligned}\frac{\partial E}{\partial z_b} &= \sum_k \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial y_b}\frac{\partial y_b}{\partial z_b} \\ &= \sum_k \frac{\partial E}{\partial z_k}.w_{b\to k}.y_b(1-y_b) \\ &= y_b(1-y_b)\sum_k \frac{\partial E}{\partial z_k}.w_{b\to k}\end{aligned}$$

- or $w \leftarrow w - \eta\nabla_w E$

$$\begin{aligned} w_{i\to b} &\leftarrow w_{i\to b} - \eta\frac{\partial E}{\partial w_{i\to b}} \\ w_{i\to b} &\leftarrow w_{i\to b} - \eta\frac{\partial E}{\partial z_b}\frac{\partial z_b}{\partial w_{i\to b}} = w_{i\to b} - \eta\frac{\partial E}{\partial z_b} y_i \end{aligned}$$

- Règle du δ :
  Si on pose $\delta_b = \frac{\partial E}{\partial z_b}$, alors : $w_{i\to b} \leftarrow w_{i\to b} - \eta\delta_b y_i$

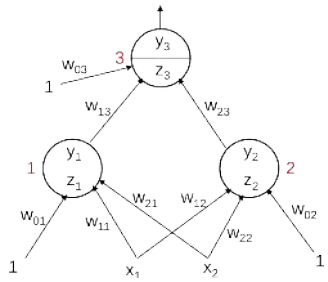
Algorithme de rétro-propagation du gradient de l'erreur

Algorithme: Algorithme de rétro-propagation du gradient de l'erreur (version stochastique) pour un perceptron multi-couches ayant $P+1$ entrées (P attributs + le biais), $q+1$ couches numérotées de 0 (C_0 : couche d'entrée) à q (C_q : couche de sortie) et une seule sortie ; notation : $s(x_i)$ est la sortie du PMC pour la donnée x_i, $y_{l,k}$ est la sortie de la ke unité (entrée ou neurone) de la couche l, $w_{l,k,m}$ est le poids de la connexion entre l'unité k de la couche l et le neurone m de la couche $l+1$ ($k=0$ correspond au biais), $|C_l|$ est le nombre d'unités composant la couche C_l. L'algorithme donné ici correspond à des neurones à fonction d'activation logistique $\frac{1}{1+e^{-x}}$.

Nécessite : les N instances d'apprentissage X
Nécessite : taux d'apprentissage $\alpha \in]0,1]$
initialiser les w_i
while *critère d'arrêt non rempli* **do**
  mettre à jour α
  mélanger les exemples
  for *tout exemple* x_i **do**
    $s(x_i) \leftarrow$ sortie du réseau pour l'exemple x_i
    $\delta_{q,1} \leftarrow -s(x_i)(1-s(x_i))(y_i - s(x_i))$
    for *toutes les couches cachées : l décroissant de $q-1$ à 1* **do**
      for *tous les neurones k de la couche C_l* **do**
        $\delta_{l,k} \leftarrow y_{l,k}(1-y_{l,k})\sum_{m\in\{0,\dots,|C_{l+1}|\}} w_{l,k,m}\delta_{l+1,m}$
      end
    end
    // mise à jour des poids
    for *toutes les couches l croissant de 0 à $q-1$* **do**
      for *toutes les unités k de la couche l, k variant de 1 à $|C_l|$* **do**
        for *tous les neurones m connectés sur la sortie du neurone k de la couche l, m variant de 1 à $|C_l|+1$* **do**
          $w_{l,k,m} \leftarrow w_{l,k,m} - \alpha\delta_{l+1,m}y_{l,k}$
        end
      end
    end
  end
end

Backpropagation : Exemple

- Initialiser les poids et calculer les z_i et y_i
- Calcul des δ_i :
  $\delta_3 = y_3(1-y_3)(y_3 - y^m)$
  $\delta_2 = y_2(1-y_2)\delta_3 w_{23}$
  $\delta_1 = y_1(1-y_1)\delta_3 w_{13}$
- Mise à jour des poids :



$$w_{03} = w_{03} - \eta\delta_3(1) \quad w_{02} = w_{02} - \eta\delta_2(1) \quad w_{01} = w_{01} - \eta\delta_1(1)$$
$$w_{13} = w_{13} - \eta\delta_3 y_1 \quad w_{12} = w_{12} - \eta\delta_2 x_1 \quad w_{11} = w_{11} - \eta\delta_1 x_1$$
$$w_{23} = w_{23} - \eta\delta_3 y_2 \quad w_{22} = w_{22} - \eta\delta_2 x_2 \quad w_{21} = w_{21} - \eta\delta_1 x_2$$

- Dérivation de δ_2 et δ_1

$$\begin{aligned}\delta_2 &= \frac{\partial E}{\partial z_2} = \frac{\partial E}{\partial z_3}\frac{\partial z_3}{\partial z_2} = \delta_3\frac{\partial z_3}{\partial z_2} \\ &= \delta_3\frac{\partial(w_{13}y_1 + w_{23}y_2)}{\partial z_2} \\ &= \delta_3 w_{23}\frac{\partial y_2}{\partial z_2} = \delta_3 w_{23}\frac{\partial s(z_2)}{\partial z_2} \\ &= \delta_3 w_{23} y_2(1-y_2)\end{aligned}$$

$$\begin{aligned}\delta_1 &= \frac{\partial E}{\partial z_1} = \frac{\partial E}{\partial z_3}\frac{\partial z_3}{\partial z_1} = \delta_3\frac{\partial z_3}{\partial z_1} \\ &= \delta_3\frac{\partial(w_{13}y_1 + w_{23}y_2)}{\partial z_1} \\ &= \delta_3 w_{13}\frac{\partial y_1}{\partial z_1} = \delta_3 w_{13}\frac{\partial s(z_1)}{\partial z_1} \\ &= \delta_3 w_{13} y_1(1-y_1)\end{aligned}$$

Codage des entrées et sorties

- Pour les réseaux de neurones, toutes les valeurs des attributs doivent être encodées d'une manière standardisée, entre 0 et 1, même pour des variables catégorielles.
- Pour les variables continues, on applique simplement la normalisation entre le min et le max :
  $X^* = [X - min(X)]/[max(X) - min(X)]$
- Pour les variables catégorielles, on utilise des indicatrices.
  - *e.g. attribut d'état matrimonial*, pouvant valoir les valeurs *célibataire, marié, divorcé*.
  - les enregistrements des célibataires auront alors 1 pour célibataire, et 0 pour le reste, *i.e.* (1,0,0)
  - les enregistrements des mariés seront codés par 1 pour marié, et 0 pour le reste, *i.e.* (0,1,0)
  - les enregistrements des divorcés seront codés par 1 pour divorcé, et 0 pour le reste, *i.e.* (0,0,1)
  - les enregistrements pour la valeur *unknown* peuvent alors être codés par 0 pour chacune des 3 valeurs, *i.e.* (0,0,0)
- En général, les attributs catégoriels avec k valeurs peuvent être traduites en $k-1$ attributs "indicateurs".

Un seul nœud de sortie / plusieurs nœuds de sortie

- Les nœuds de sorties des réseaux neuronaux renvoient toujours une valeur entre 0 et 1.
- De nombreux problèmes de classification ont un résultat dichotomique, avec seulement deux valeurs possibles.
  e.g., "Méningite, oui ou non"
- Pour ces problèmes, on peut utiliser un seul nœud de sortie, avec une valeur de seuil fixé *a priori* qui sépare les classes.
  Par exemple, si la décision est "oui" lorsque la sortie ≥ 0.3*," une sortie valant* 0.4 *permettrait de classer l'enregistrement comme étant "oui".*
- Un seul nœud de sortie peut être suffisant lorsque les classes sont clairement ordonées (vis-à-vis de la sortie). *E.g.*, supposons qu'on veuille classer les degrés d'une maladie de patients. On peut dire :
  - Si $0.00 \leq$ *sortie* < 0.33, classer dans "faible"
  - Si $0.33 \leq$ *sortie* < 0.66, classer dans "sévère"
  - Si $0.66 \leq$ *sortie* < 1.00, classer dans "grave"

- Si on a des catégories non ordonnées pour l'attribut cible, on crée un nœud de sortie pour chacune des catégories.
  E.g., si on veut prédire l'état matrimonial, le réseau pourrait avoir 4 nœuds de sortie (dans la couche de sortie), un pour :
  - célibataire, marié, divorcé, et unknown.

  $\Rightarrow$ Le nœud de sortie avec la plus grande valeur est alors choisi pour classer l'enregistrement particulier traité.

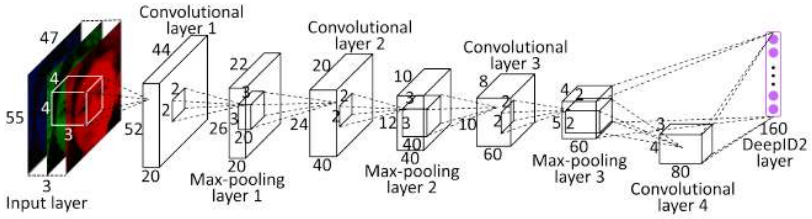
Deep Learning

- Un **théorème d'approximation universelle montre** qu'un perceptron à une couche cachée (avec un nombre de neurones grand mais fini) est suffisant pour résoudre tous les problèmes classiques d'apprentissage.
- En pratique, le grand nombre de paramètres nécessaires induit des problèmes d'estimation et de stabilité. Ce résultat est donc essentiellement théorique.
- Les **réseaux de neurones profonds** partent du principe qu'il est plus efficace de multiplier le nombre de couches plutôt que le nombre de neurones.
- Ainsi, plusieurs réseaux de neurones profonds ont été développés récemment en **empilant** un nombre généralement important de **couches de neurones aux propriétés spécifiques**.
- Trois principaux types de réseaux profonds (1/3)

Deep Learning

- Un **théorème d'approximation universelle montre** qu'un perceptron à une couche cachée (avec un nombre de neurones grand mais fini) est suffisant pour résoudre tous les problèmes classiques d'apprentissage.
- En pratique, le grand nombre de paramètres nécessaires induit des problèmes d'estimation et de stabilité. Ce résultat est donc essentiellement théorique.
- Les **réseaux de neurones profonds** partent du principe qu'il est plus efficace de multiplier le nombre de couches plutôt que le nombre de neurones.
- Ainsi, plusieurs réseaux de neurones profonds ont été développés récemment en **empilant** un nombre généralement important de **couches de neurones aux propriétés spécifiques**.
- Trois principaux types de réseaux profonds (1/3)

ConvNet : *convolutional* neural networks, pour l'analyse d'image



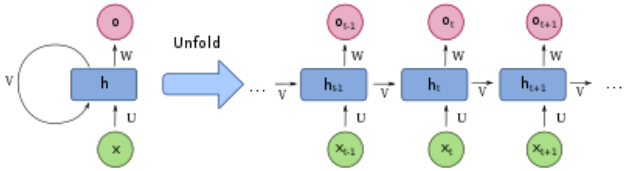
CÔTE D'AZUR

Deep Learning

AI for bio
J-P Comet
Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learing
Evaluation
Data Mining
Project

- Un **théorème d'approximation universelle montre** qu'un perceptron à une couche cachée (avec un nombre de neurones grand mais fini) est suffisant pour résoudre tous les problèmes classiques d'apprentissage.
- En pratique, le grand nombre de paramètres nécessaires induit des problèmes d'estimation et de stabilité. Ce résultat est donc essentiellement théorique.
- Les **réseaux de neurones profonds** partent du principe qu'il est plus efficace de multiplier le nombre de couches plutôt que le nombre de neurones.
- Ainsi, plusieurs réseaux de neurones profonds ont été développés récemment en **empilant** un nombre généralement important de **couches de neurones aux propriétés spécifiques**.
- Trois principaux types de réseaux profonds (1/3)

LSTM : *long-short term memory*, pour le traitement du signal ou analyse du langage naturel



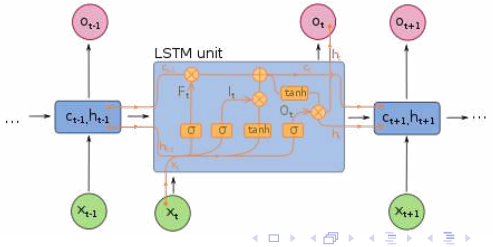
CÔTE D'AZUR

Deep Learning

AI for bio
J-P Comet
Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learing
Evaluation
Data Mining
Project

- Un **théorème d'approximation universelle montre** qu'un perceptron à une couche cachée (avec un nombre de neurones grand mais fini) est suffisant pour résoudre tous les problèmes classiques d'apprentissage.
- En pratique, le grand nombre de paramètres nécessaires induit des problèmes d'estimation et de stabilité. Ce résultat est donc essentiellement théorique.
- Les **réseaux de neurones profonds** partent du principe qu'il est plus efficace de multiplier le nombre de couches plutôt que le nombre de neurones.
- Ainsi, plusieurs réseaux de neurones profonds ont été développés récemment en **empilant** un nombre généralement important de **couches de neurones aux propriétés spécifiques**.
- Trois principaux types de réseaux profonds (1/3)

LSTM : *long-short term memory*, pour le traitement du signal ou analyse du langage naturel



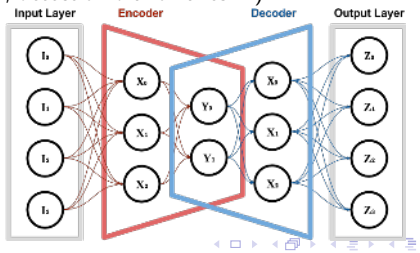
CÔTE D'AZUR

Deep Learning

AI for bio
J-P Comet
Introduction
Trees
Clustering
kNN
Bayes
SVM
NN
Introduction
Le perceptron
Descente de gradient
Perceptron
Multi-couche
Soft Threshold
Deep Learing
Evaluation
Data Mining
Project

- Un **théorème d'approximation universelle montre** qu'un perceptron à une couche cachée (avec un nombre de neurones grand mais fini) est suffisant pour résoudre tous les problèmes classiques d'apprentissage.
- En pratique, le grand nombre de paramètres nécessaires induit des problèmes d'estimation et de stabilité. Ce résultat est donc essentiellement théorique.
- Les **réseaux de neurones profonds** partent du principe qu'il est plus efficace de multiplier le nombre de couches plutôt que le nombre de neurones.
- Ainsi, plusieurs réseaux de neurones profonds ont été développés récemment en **empilant** un nombre généralement important de **couches de neurones aux propriétés spécifiques**.
- Trois principaux types de réseaux profonds (1/3)

AutoEncoder decoder(diabolo), pour l'apprentissage nonsupervisé (débruitage d'images ou signaux, détection d'anomalies...)

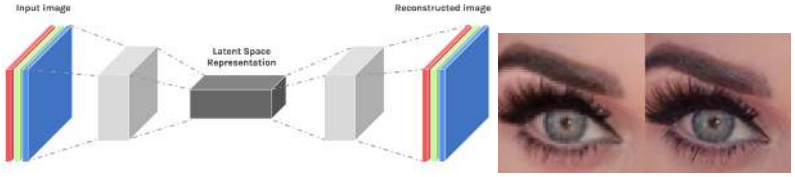


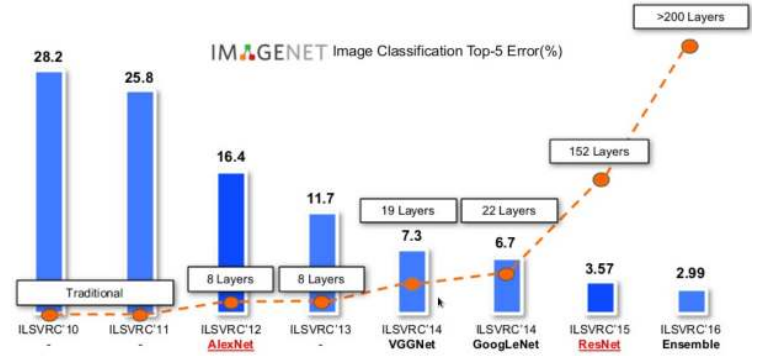
Les auto-encodeurs

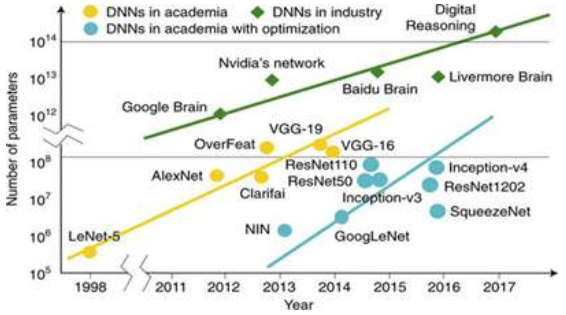
- l'**auto-encodeur** est un réseau de neurones (NN), et un algorithme d'apprentissage non-supervisé (apprentissage de caractéristiques).
  - Il applique la rétro-propagation, en voulant prédire (target) l'entrée à partir de l'entrée.
  - Il essaie de prédire x à partir de x, sans aucun besoin d'étiquette.

$$x(n) = Q^{-1}Qx(n)$$

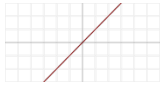
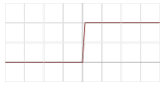
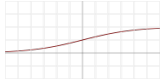
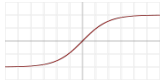
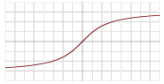
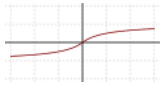
  - Il essaie d'apprendre une approximation de la fonction **"identity"**.
  - Mais on rajoute des **contraintes** sur le réseau, comme par exemple **un nombre limité** d'unités dans la couche cachée.
    cette couche représente l'entrée
    cette couche est celle qui est comprise entre **Encodeur & Décodeur**.



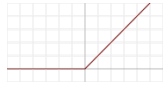
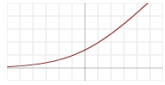
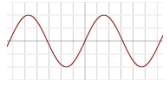
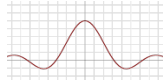
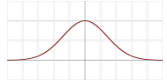




Autres fonctions d'activation (1)

Identité/Rampe		$f(x) = x$
Marche/Heaviside		$f(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases}$
Logistique (ou sigmoïde)		$f(x) = \frac{1}{1+e^{-x}}$
Tang. Hyperb. (Tanh)		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
arctan ou tan^{-1}		$f(x) = \tan^{-1}(x)$
Signe doux		$f(x) = \frac{x}{1+\lvert x \rvert}$

Autres fonctions d'activation (2)

Unité de Rectification Linéaire (ReLU)		$f(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ x & \text{si} \quad x \geq 0 \end{cases}$
Unité de Rectif. Linéaire Douce (SoftPlus)		$f(x) = \log_e(1+e^x)$
Sinusoïde		$f(x) = sin(x)$
Sinus cardinal (Sinc)		$f(x) = \begin{cases} 1 & \text{si} \quad x = 0 \\ \frac{\sin(x)}{x} & \text{si} \quad x \neq 0 \end{cases}$
Fonction Gaussienne		$f(x) = e^{-x^2}$