Apprentissage profond
ou deep learning (pour les anglophones)

Mathieu Lefort

13 mai 2016

Un (très) rapide aperçu de l'apprentissage automatique

La question

$\mathbf{y} = \mathbf{h}(\mathbf{x})$ avec $\mathbf{x}$ et $\mathbf{y} \in \mathcal{Y}$

Remarques :

- régression : $\mathcal{Y}$ continu, typiquement $\mathbb{R}^n$
- classification : $\mathcal{Y}$ discret
- si on veut faire du non supervisé on peut prendre $\mathbf{x}$ comme $\mathbf{y}$

Un (très) rapide aperçu de l'apprentissage automatique

La question

$\mathbf{y} = \mathbf{h}(\mathbf{x})$ avec $\mathbf{x}$ et $\mathbf{y} \in \mathcal{Y}$

Remarques :

- régression : $\mathcal{Y}$ continu, typiquement $\mathbb{R}^n$
- classification : $\mathcal{Y}$ discret
- si on veut faire du non supervisé on peut prendre $\mathbf{x}$ comme $\mathbf{y}$

La solution (classique)

$\mathbf{h}(\mathbf{x}) = \sum_{i=1}^{n} (\mathbf{a_i^T}.\mathbf{x} + \mathbf{b_i})\phi(\mathbf{x}, \theta_\mathbf{i})$ avec ϕ une fonction non linéaire paramétrée par $\theta_\mathbf{i}$,
$\mathbf{b_i} \in \mathbb{R}^{card(\mathcal{Y})}$ et $\mathbf{a_i} \in \mathbb{R}^{card(\mathcal{Y}) \times card(\mathcal{X})}$

Un (très) rapide aperçu de l'apprentissage automatique

La question

$\mathbf{y} = \mathbf{h}(\mathbf{x})$ avec $\mathbf{x}$ et $\mathbf{y} \in \mathcal{Y}$

Remarques :

- régression : $\mathcal{Y}$ continu, typiquement $\mathbb{R}^n$
- classification : $\mathcal{Y}$ discret
- si on veut faire du non supervisé on peut prendre $\mathbf{x}$ comme $\mathbf{y}$

La solution (classique)

$\mathbf{h}(\mathbf{x}) = \sum_{i=1}^{n} (\mathbf{a_i^T}.\mathbf{x} + \mathbf{b_i})\phi(\mathbf{x}, \theta_\mathbf{i})$ avec ϕ une fonction non linéaire paramétrée par $\theta_\mathbf{i}$,
$\mathbf{b_i} \in \mathbb{R}^{card(\mathcal{Y})}$ et $\mathbf{a_i} \in \mathbb{R}^{card(\mathcal{Y}) \times card(\mathcal{X})}$

Ce qui diffère entre les modèles

- La façon de représenter/considérer la tâche
- Le nombre de composantes
- Les non linéarités choisies
- Les paramètres qui sont appris
- Les méthodes d'apprentissage

Un (très) rapide aperçu de l'apprentissage automatique

La question

$\mathbf{y} = \mathbf{h}(\mathbf{x})$ avec $\mathbf{x}$ et $\mathbf{y} \in \mathcal{Y}$

Remarques :

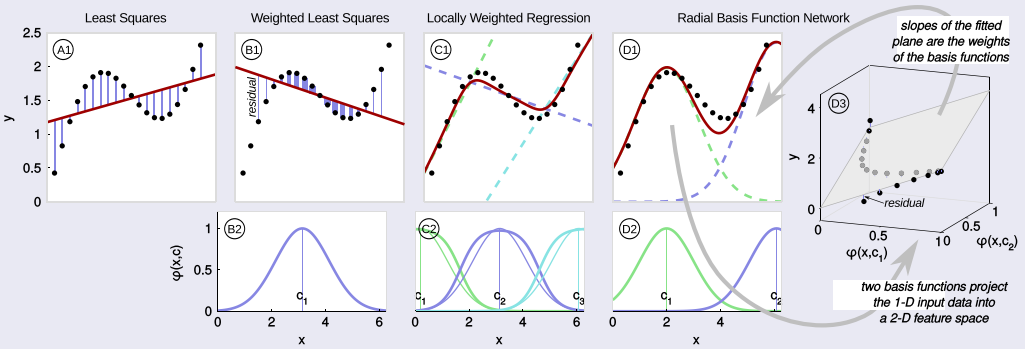
- régression : $\mathcal{Y}$ continu, typiquement $\mathbb{R}^n$
- classification : $\mathcal{Y}$ discret
- si on veut faire du non supervisé on peut prendre $\mathbf{x}$ comme $\mathbf{y}$

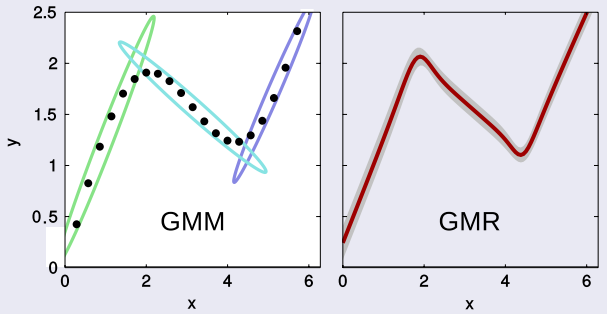
La solution (classique)

$\mathbf{h}(\mathbf{x}) = \sum_{i=1}^{n} (\mathbf{a_i^T}.\mathbf{x} + \mathbf{b_i})\phi(\mathbf{x}, \theta_\mathbf{i})$ avec ϕ une fonction non linéaire paramétrée par $\theta_\mathbf{i}$,

$\mathbf{b_i} \in \mathbb{R}^{card(\mathcal{Y})}$ et $\mathbf{a_i} \in \mathbb{R}^{card(\mathcal{Y}) \times card(\mathcal{X})}$

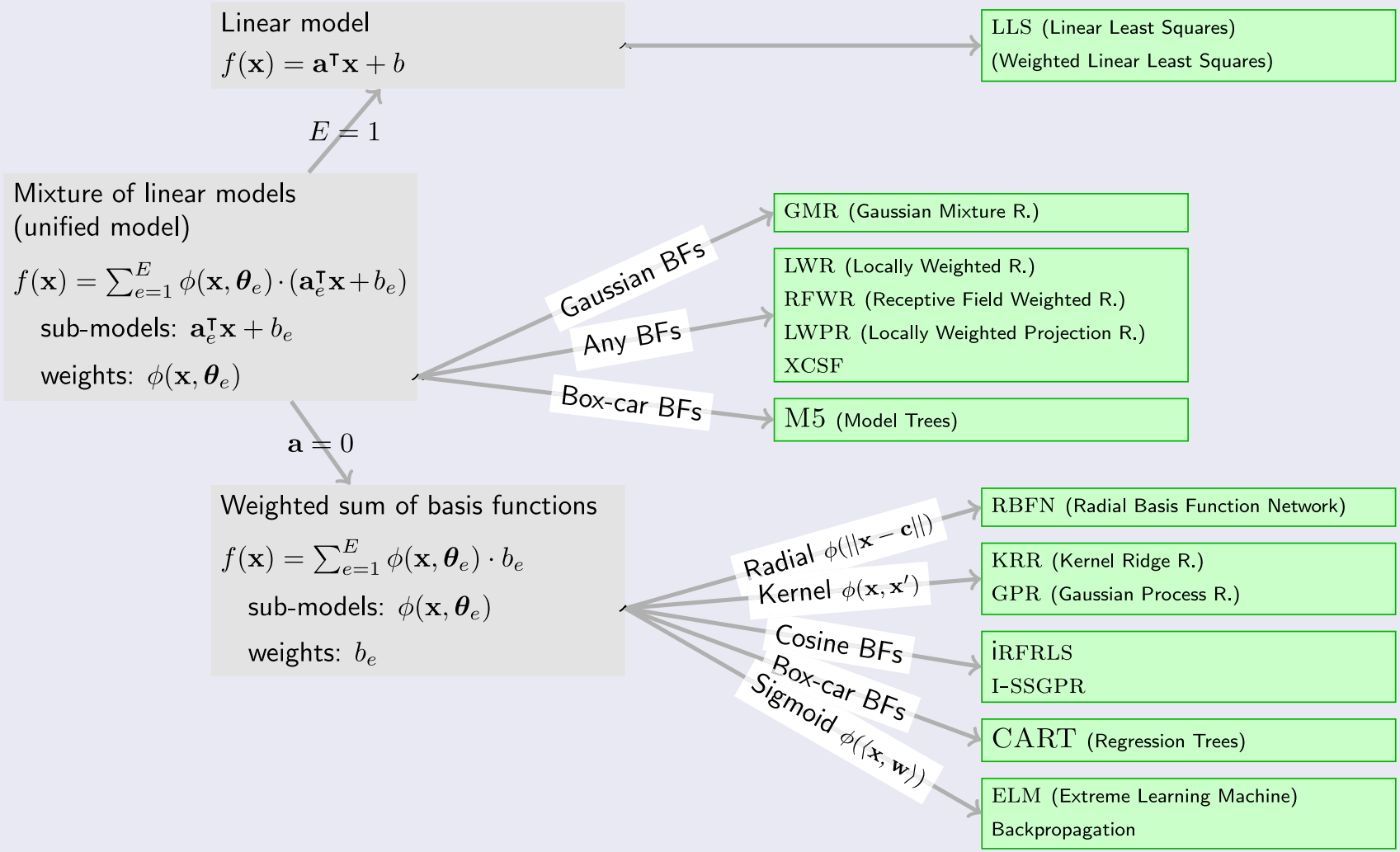
Exemples





Un (très) rapide aperçu de l'apprentissage automatique

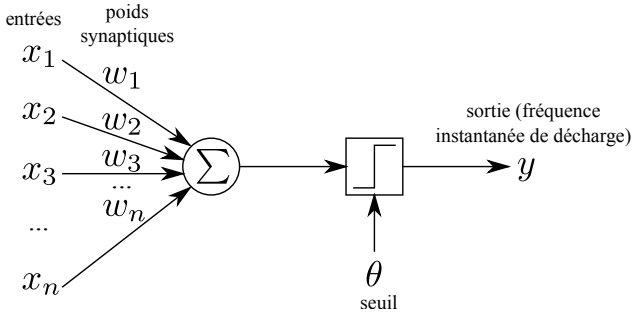
Récapitulatif



Neurone artificiel - McCulloch et Pitts (1943)

L'inspiration biologique :

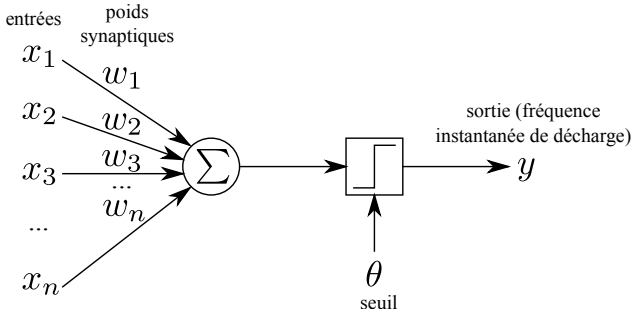
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=1}^{n} w_i x_i - \theta > 0 \\ 0 & \text{sinon} \end{cases}$



Neurone artificiel - McCulloch et Pitts (1943)

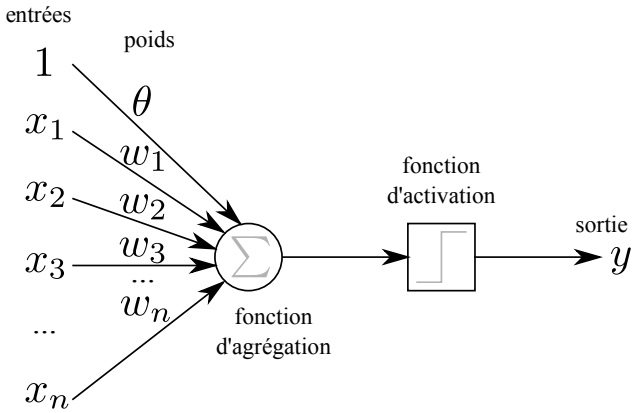
L'inspiration biologique :

$$\underline{\textbf{Sortie du neurone}} : y = \begin{cases} 1 & \text{si } \sum_{i=1}^{n} w_i x_i - \theta > 0 \\ 0 & \text{sinon} \end{cases}$$



Le neurone en apprentissage automatique :

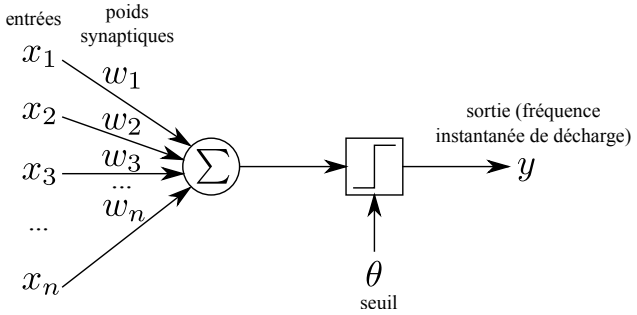
$$\underline{\textbf{Sortie du neurone}} : y = \begin{cases} 1 & \text{si } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$$



Neurone artificiel - McCulloch et Pitts (1943)

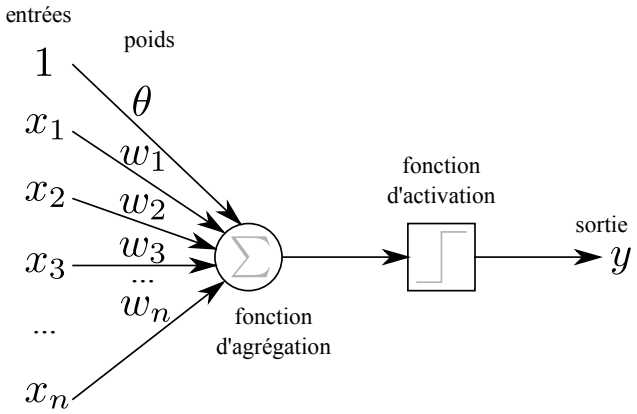
L'inspiration biologique :

$$\underline{\textbf{Sortie du neurone}} : y = \begin{cases} 1 & \text{si } \sum_{i=1}^{n} w_i x_i - \theta > 0 \\ 0 & \text{sinon} \end{cases}$$



Le neurone en apprentissage automatique :

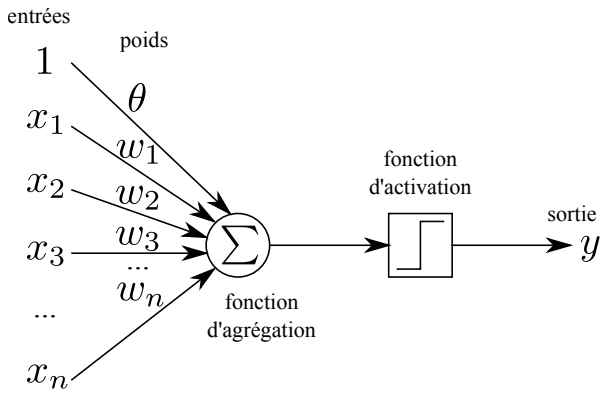
$$\underline{\textbf{Sortie du neurone}} : y = \begin{cases} 1 & \text{si } \mathbf{w}^T\mathbf{x} > 0 \\ 0 & \text{sinon} \end{cases}$$



Perceptron - Rosenblatt (1957)

Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

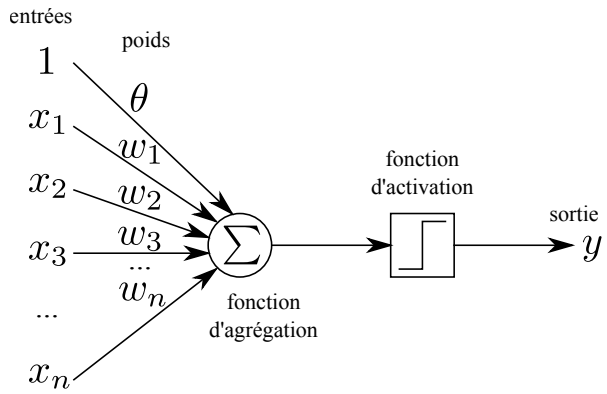
Apprentissage : $\Delta \mathbf{w} = \eta \mathbf{x}(t - y)$



Perceptron - Rosenblatt (1957)

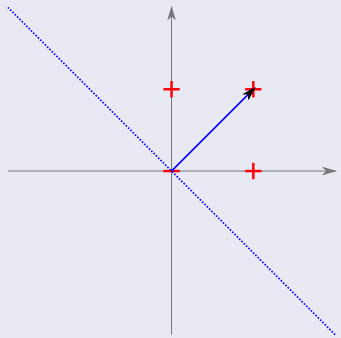
Sortie du neurone : $y = \begin{cases} 1 & \text{si } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{sinon} \end{cases}$

Apprentissage : $\Delta \mathbf{w} = \eta \mathbf{x}(t - y)$



Un exemple : le OU

Entrée **x**			Poids **w**			Sortie	Sortie voulue
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	0	0	1	1	0	0

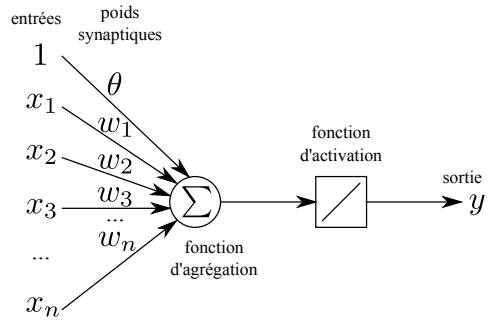


Propriétés

- Classifieur linéaire (découpage de l'espace avec un hyperplan)
- Poids = vecteur normal à l'hyperplan, Seuil (biais) = ordonnée à l'origine
- **Convergence uniquement si les données sont linéairement séparables** et si η est (infinitésimalement) petit

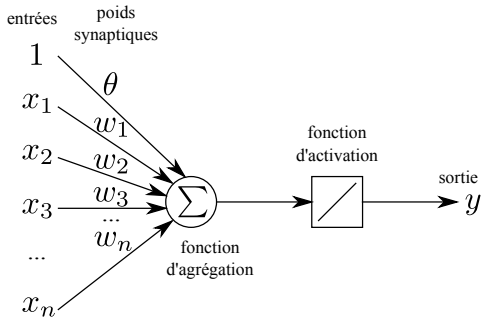
Perceptron - Descente de gradient stochastique

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

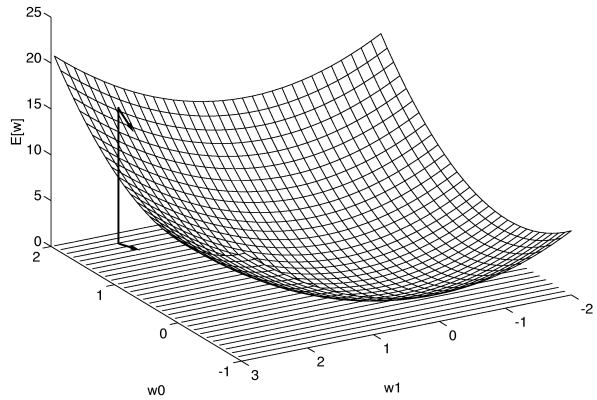


Perceptron - Descente de gradient stochastique

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

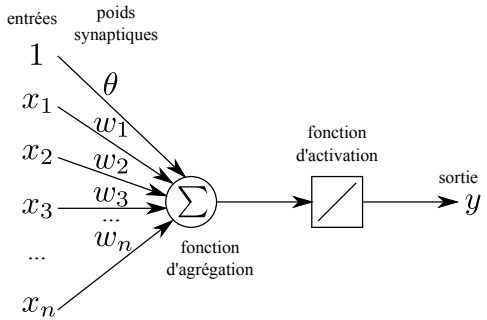


Apprentissage : On veut minimiser l'erreur quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (t - y)^2$

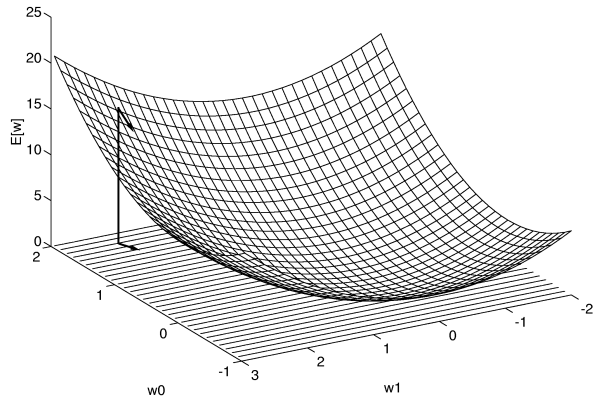


Perceptron - Descente de gradient stochastique

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



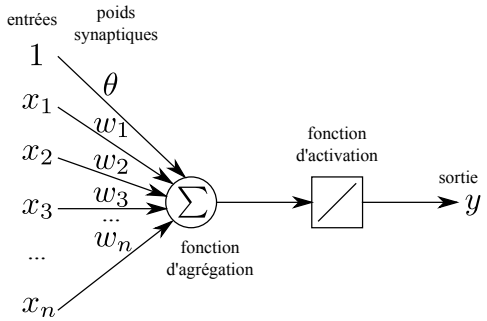
Apprentissage : On veut minimiser l'erreur quadratique $E = \frac{1}{2}\sum_{\mathbf{x}}(t-y)^2$



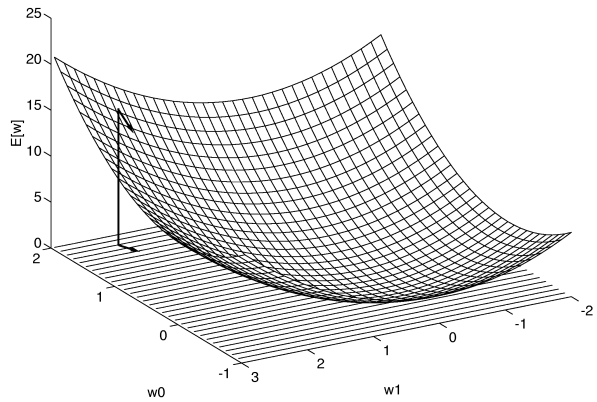
$$\begin{aligned}
\Delta w_i &= -\eta \frac{\partial E}{\partial w_i} \\
&= -\frac{\eta}{2}\sum_{\mathbf{x}} \frac{\partial (t-y)^2}{\partial w_i} \\
&= -\frac{\eta}{2}\sum_{\mathbf{x}} -2\frac{\partial y}{\partial w_i}(t-y) \\
&= \sum_{\mathbf{x}} \eta x_i (t-y)
\end{aligned}$$

Perceptron - Descente de gradient stochastique

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



Apprentissage : On veut minimiser l'erreur quadratique $E = \frac{1}{2} \sum_{\mathbf{x}} (t - y)^2$ par descente de gradient stochastique : $\Delta \mathbf{w} = \eta \mathbf{x}(t - y)$



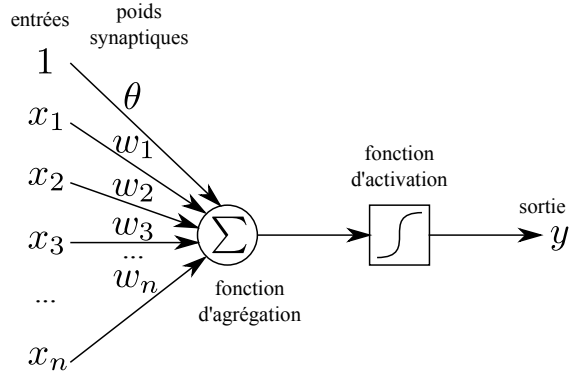
Propriétés

- 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*

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

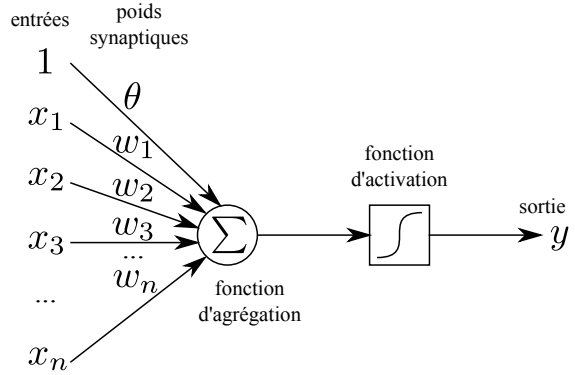
Sortie du neurone : $s = \sum_{i=0}^{n} w_i x_i$

$$y = \frac{1}{1 + e^{-s}}$$



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

$$y = \frac{1}{1+e^{-s}}$$

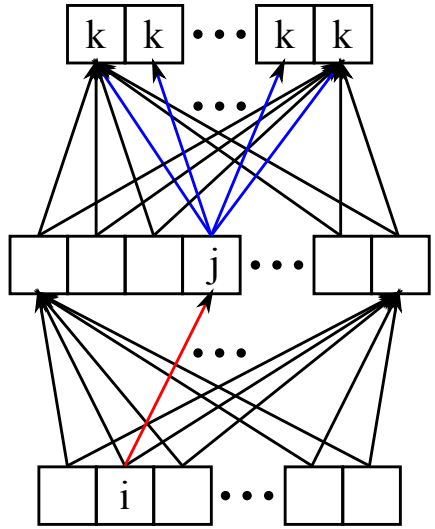


Apprentissage : On veut minimiser l'erreur quadratique $E = \frac{1}{2}\sum_{\mathbf{x}}(t-y)^2$ par descente de gradient stochastique : $\Delta \mathbf{w} = \eta \mathbf{x}(t-y)y(1-y)$

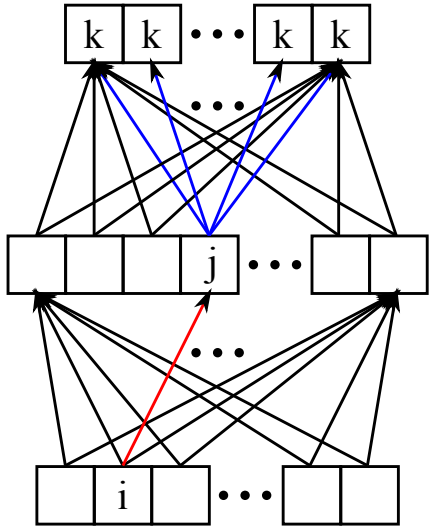
$$\begin{aligned}
\Delta w_i &= -\eta \frac{\partial E}{\partial w_i} \\
&= -\eta \frac{\partial E}{\partial s}\frac{\partial s}{\partial w_i} \\
&= \eta \sum_{\mathbf{x}} \frac{\partial y}{\partial s}(t-y)\frac{\partial s}{\partial w_i} \\
&= \sum_{\mathbf{x}} \eta y(1-y)(t-y)x_i
\end{aligned}$$

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



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

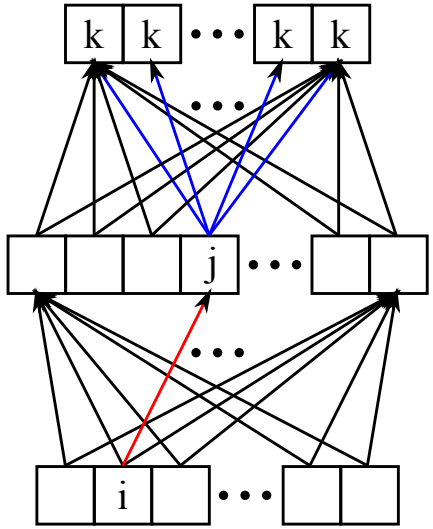


Réseau : connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie

Apprentissage : On veut minimiser l'erreur quadratique $E = \frac{1}{2}\sum_{\mathbf{x}}(t-y)^2$ par descente de gradient stochastique, on suppose connu les $\delta_k = -\frac{\partial E}{\partial s_k}$ de la couche supérieure

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

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

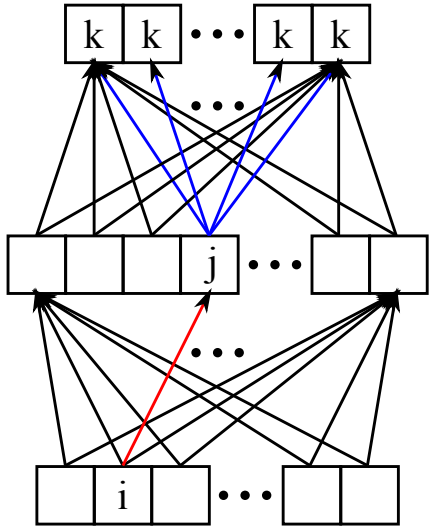


Réseau : connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie

Apprentissage : On veut minimiser l'erreur quadratique $E = \frac{1}{2}\sum_{\mathbf{x}}(t-y)^2$ par descente de gradient stochastique, on suppose connu les $\delta_k = -\frac{\partial E}{\partial s_k}$ de la couche supérieure

$$
\begin{aligned}
-\frac{\partial E}{\partial s_j} &= -\sum_k \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial y_j}\frac{\partial y_j}{\partial s_j} \\
&= -\sum_k -\delta_k w_{jk} y_j(1-y_j) \\
&= y_j(1-y_j)\sum_k \delta_k w_{jk} \\
\Delta w_{ij} &= -\eta\frac{\partial E}{\partial w_{ij}} \\
&= -\eta\frac{\partial E}{\partial s_j}\frac{\partial s_j}{\partial w_{ij}} \\
&= \eta\delta_j x_i
\end{aligned}
$$

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

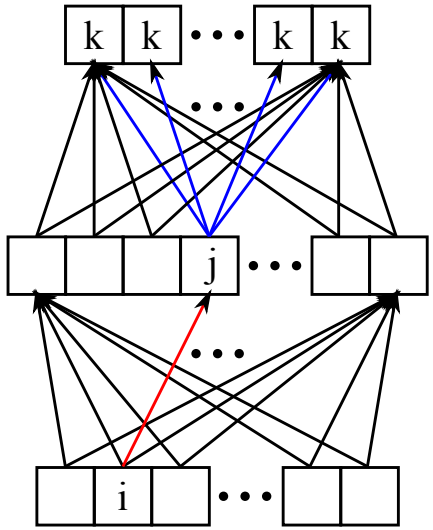


Réseau : connectivité complète entre la couche d'entrée, la(les) couche(s) cachée(s) et la couche de sortie

Apprentissage : Pour chaque entrée reçue :

1. Calculer la sortie **y** du réseau par propagation (couche par couche) de l'activité
2. Calculer l'erreur de la couche de sortie : $\delta_k = y_k(1 - y_k)(t_k - y_k)$
3. Rétropopager l'erreur à travers chaque couche j du réseau $\delta_j = y_j(1 - y_j) \sum_k \delta_k w_{jk}$
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 :

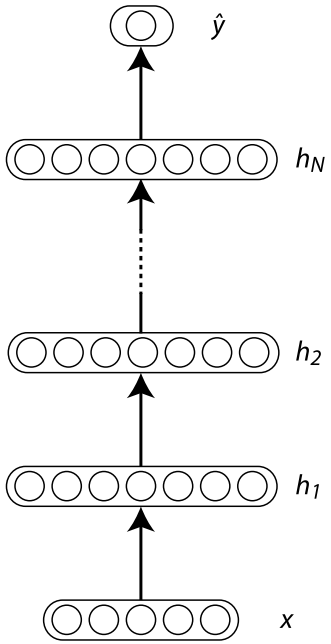
1. Calculer la sortie **y** du réseau par propagation (couche par couche) de l'activité
2. Calculer l'erreur de la couche de sortie : $\delta_k = y_k(1 - y_k)(t_k - y_k)$
3. Rétropopager l'erreur à travers chaque couche j du réseau $\delta_j = y_j(1 - y_j) \sum_k \delta_k w_{jk}$
4. Modifier chaque poids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

Propriétés

- Approximateur universel (avec fonction d'activation sigmoïde ou gaussienne)
- Convergence potentiellement peu efficace (non convexe et gradient evanescent)

Le bestiaire de l'apprentissage profond

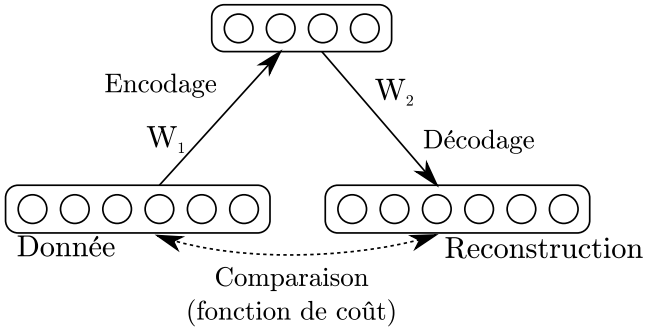
- *Perceptron / Multi-layer perceptron*

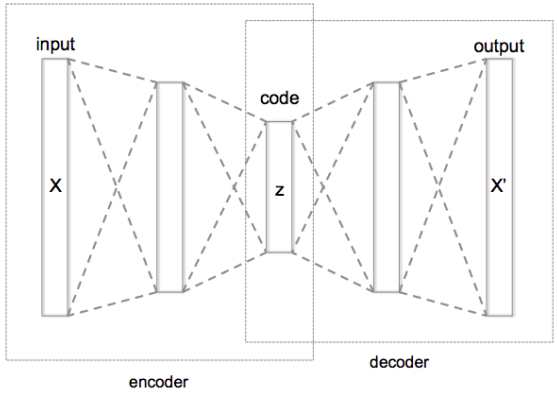


- *Auto-encoder / Stacked auto-encoder*
- *Restricted Boltzmann machine / Deep belief network*
- *Convolutional neural network*
- *Recurrent neural network*
- *Long Short Term Memory*

Le bestiaire de l'apprentissage profond

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

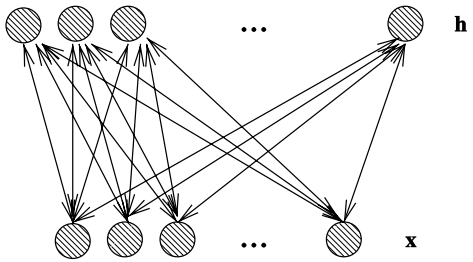




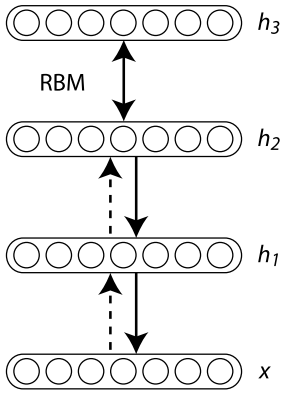
- *Restricted Boltzmann machine / Deep belief network*
- *Convolutional neural network*
- *Recurrent neural network*
- *Long Short Term Memory*

Le bestiaire de l'apprentissage profond

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



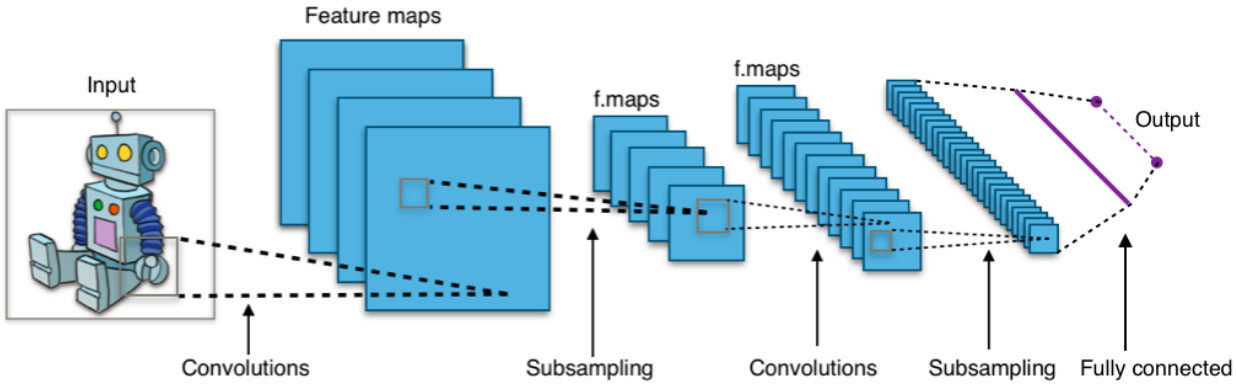
$$P(\mathbf{h}^{\ell-1}, \mathbf{h}^{\ell}) \propto e^{\mathbf{b}'\mathbf{h}^{\ell-1} + \mathbf{c}'\mathbf{h}^{\ell} + \mathbf{h}^{\ell\prime} W \mathbf{h}^{\ell-1}}$$



- *Convolutional neural network*
- *Recurrent neural network*
- *Long Short Term Memory*

Le bestiaire de l'apprentissage profond

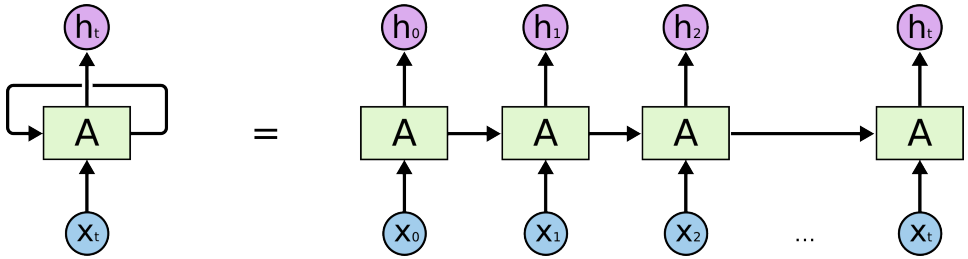
- *Perceptron / Multi-layer perceptron*
- *Auto-encoder / Stacked auto-encoder*
- *Restricted Boltzmann machine / Deep belief network*
- *Convolutional neural network*



- *Recurrent neural network*
- *Long Short Term Memory*

Le bestiaire de l'apprentissage profond

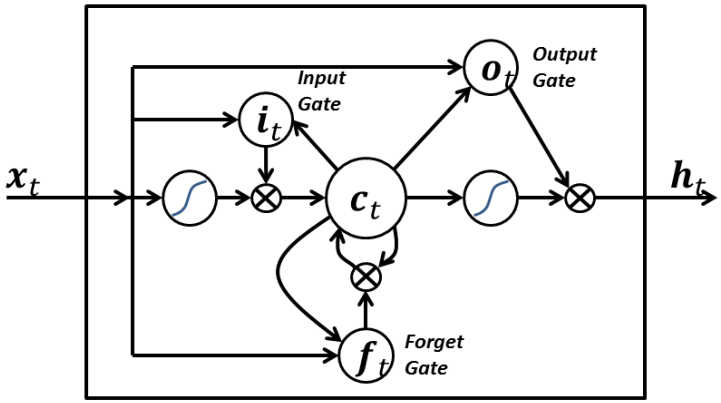
- *Perceptron / Multi-layer perceptron*
- *Auto-encoder / Stacked auto-encoder*
- *Restricted Boltzmann machine / Deep belief network*
- *Convolutional neural network*
- *Recurrent neural network*



- *Long Short Term Memory*

Le bestiaire de l'apprentissage profond

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

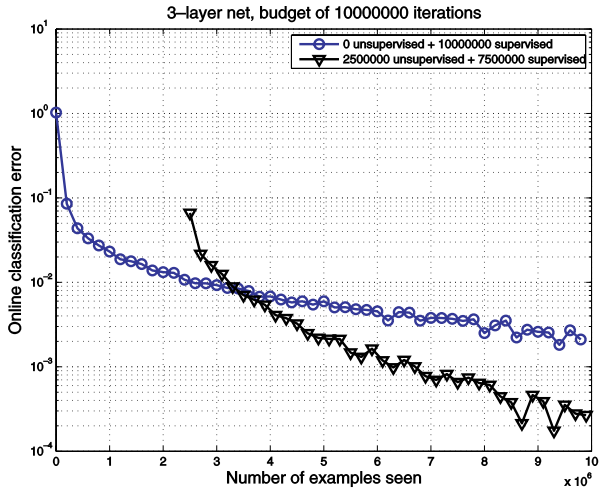


Régularisation et paramétrage

- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur
- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)
- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)
- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)
- Architecture : une ou plusieurs couches de taille limitée
- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)
- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées, ...

Régularisation et paramétrage

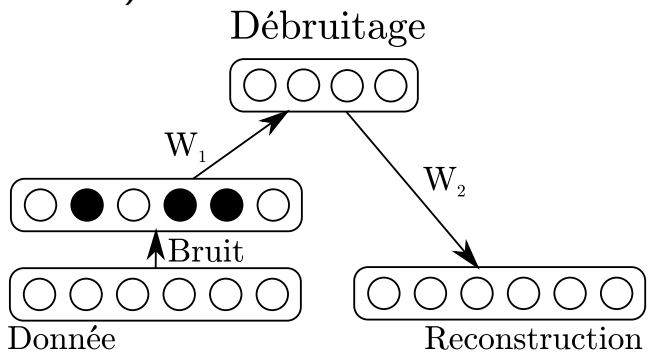
- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur



- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)
- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)
- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)
- Architecture : une ou plusieurs couches de taille limitée
- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)
- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées

Régularisation et paramétrage

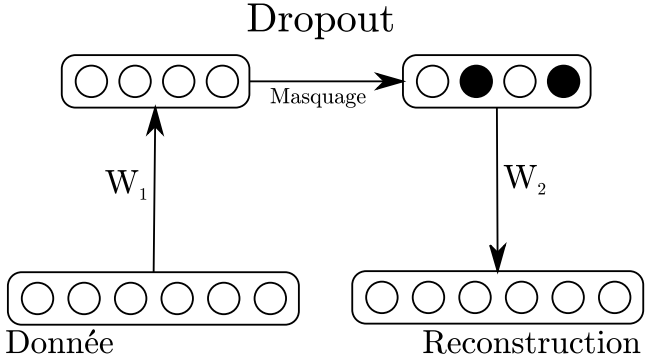
- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur
- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)



- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)
- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)
- Architecture : une ou plusieurs couches de taille limitée
- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)
- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées, ...

Régularisation et paramétrage

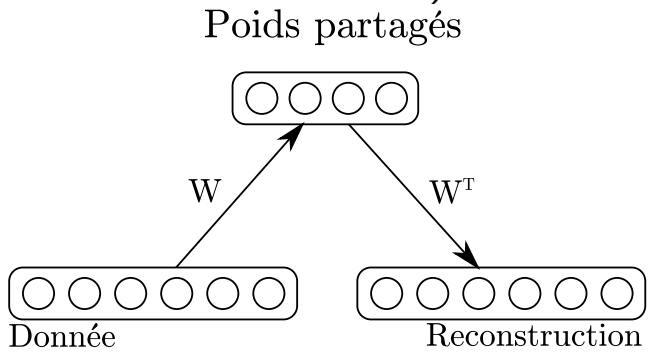
- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur
- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)
- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)



- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)
- Architecture : une ou plusieurs couches de taille limitée
- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)
- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées, ...

Régularisation et paramétrage

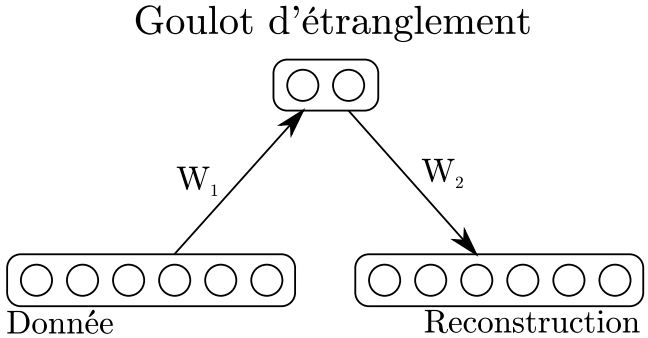
- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur
- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)
- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)
- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)



- Architecture : une ou plusieurs couches de taille limitée
- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)
- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées, ...

Régularisation et paramétrage

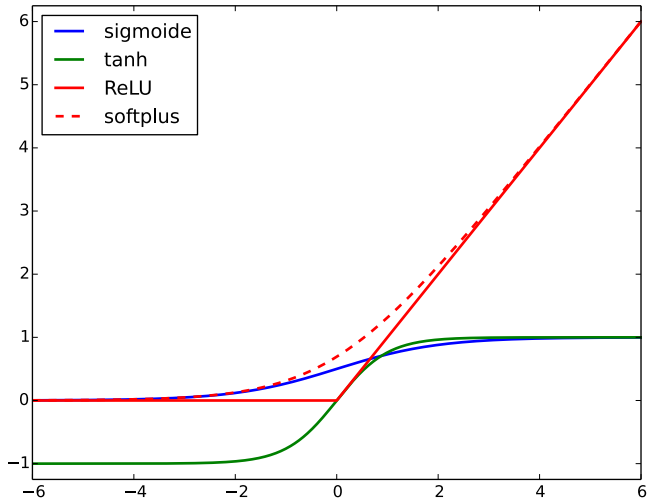
- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur
- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)
- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)
- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)
- Architecture : une ou plusieurs couches de taille limitée



- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)
- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées, ...

Régularisation et paramétrage

- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur
- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)
- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)
- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)
- Architecture : une ou plusieurs couches de taille limitée
- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)



- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées, ...

Régularisation et paramétrage

- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur
- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)
- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)
- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)
- Architecture : une ou plusieurs couches de taille limitée
- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)
- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées, ...

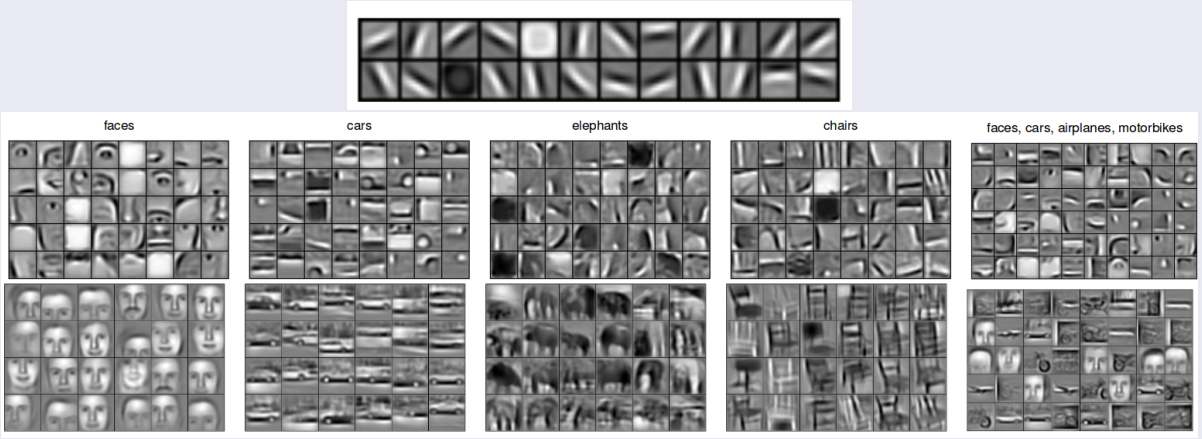
Régularisation et paramétrage

- Fonction de coût : rajout d'un terme dépendant de la norme des poids (généralement norme L_1 ou L_2) ou à l'activation des couches (codage épars)
- *Pre-training* : (pré) apprentissage séparé de chaque couche en tant qu'auto-encodeur
- Débruitage : mise à zéro d'une partie des composantes de l'entrée (généralement utilisé pour les auto-encodeurs)
- *Drop out* : mise à zéro d'une partie des composantes de la représentation (généralement utilisé pour les auto-encodeurs)
- Partage des poids : *Convolution Neural Network* (combiné à du *pooling*) ou transposée des poids (dans les auto-encodeurs)
- Architecture : une ou plusieurs couches de taille limitée
- Fonction d'activation : sigmoïde, tangeante hyperbolique ou ReLU (*Rectified Linear Unit*)
- Calcul du gradient : momentum, gradient adaptatif, mini batch, ...
- Pré-traitement : sélection de dimensions, données centrées normées, ...

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)

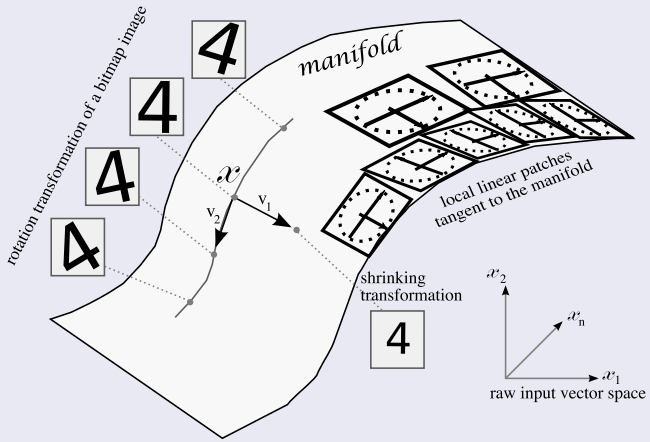


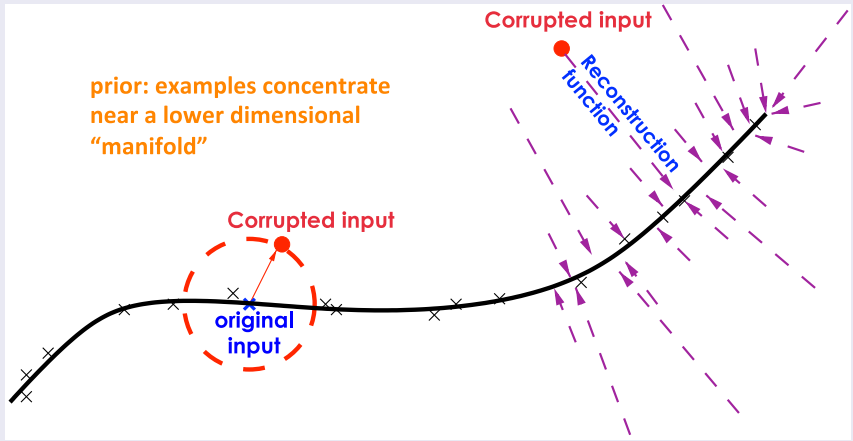
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)





- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Pourquoi ça marche ?

- Beaucoup de données, de GPUs, d'ingénieurs et de chercheurs ... oui mais pas que
- Approximateur universel ... oui mais ce n'est pas le seul
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appréhende mieux les données/tâches

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Pourquoi ça marche ?

- Beaucoup de données, de GPUs, d'ingénieurs et de chercheurs ... oui mais pas que
- Approximateur universel ... oui mais ce n'est pas le seul
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appréhende mieux les données/tâches

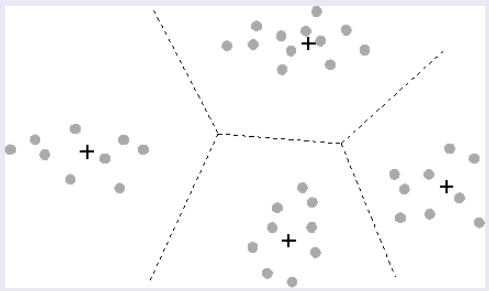
Bilan

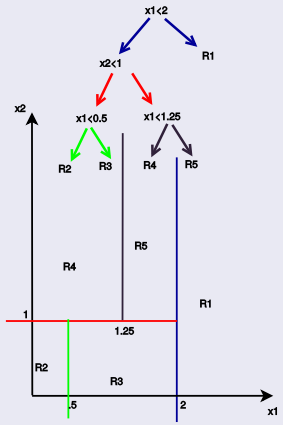
Qu'est ce que ça fait ?

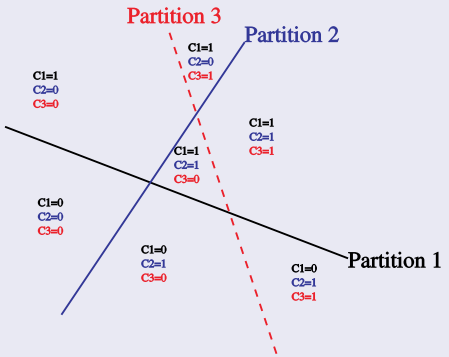
- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

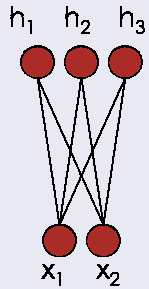
Pourquoi ça marche ?

- Beaucoup de données, de GPUs, d'ingénieurs et de chercheurs ... oui mais pas que
- Approximateur universel ... oui mais ce n'est pas le seul
- *Features* (vs prototypes) et leur apprentissage









- Pouvoir (exponentiel) d'abstraction
- Appréhende mieux les données/tâches

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Pourquoi ça marche ?

- Beaucoup de données, de GPUs, d'ingénieurs et de chercheurs ... oui mais pas que
- Approximateur universel ... oui mais ce n'est pas le seul
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appréhende mieux les données/tâches

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Pourquoi ça marche ?

- Beaucoup de données, de GPUs, d'ingénieurs et de chercheurs ... oui mais pas que
- Approximateur universel ... oui mais ce n'est pas le seul
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appréhende mieux les données/tâches

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Pourquoi ça marche ?

- Beaucoup de données, de GPUs, d'ingénieurs et de chercheurs ... oui mais pas que
- Approximateur universel ... oui mais ce n'est pas le seul
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appréhende mieux les données/tâches

Tout ce dont je ne vous ai pas parlé

Transfer learning, *Multi-tasks learning*, *Deep reinforcement learning*, *Curriculum learning*, *Memory network* (Facebook), *Neural Turing machine* (Deep Mind), ...

Bilan

Qu'est ce que ça fait ?

- Combinaison/Abstraction de représentations non linéaires (*features*)
- Apprentissage de variétés (et d'un champ de vecteur projectif)
- Désemmêlage des facteurs explicatifs
- Programmation fonctionnelle

Pourquoi ça marche ?

- Beaucoup de données, de GPUs, d'ingénieurs et de chercheurs ... oui mais pas que
- Approximateur universel ... oui mais ce n'est pas le seul
- *Features* (vs prototypes) et leur apprentissage
- Pouvoir (exponentiel) d'abstraction
- Appréhende mieux les données/tâches

Tout ce dont je ne vous ai pas parlé

Transfer learning, *Multi-tasks learning*, *Deep reinforcement learning*, *Curriculum learning*, *Memory network* (Facebook), *Neural Turing machine* (Deep Mind), ...

Les acteurs

Y. Le Cun (CNN), G. Hinton (DBN), Y. Bengio (MLP), J. Schmidhuber (LSTM)

Merci de votre attention.
Des questions ?

Démo