Réseaux de neurones artificiels

Amin Fehri[1]

09/01/2017

[1] Centre de Morphologie Mathématique
Mines ParisTech

Plan

Introduction

Objectif : régression ou classification

Classification

- Imagerie médicale : malade ou pas ?
- Reconnaissance de caractères ?

Régression

- Conduite automatique : position optimale de la roue
- Kinect : où sont les membres ?

Cela est aussi valable pour des sorties structurées.

Objectif : régression ou classification

Classification

- Imagerie médicale : malade ou pas ?
- Reconnaissance de caractères ?

Régression

- Conduite automatique : position optimale de la roue
- Kinect : où sont les membres ?

Cela est aussi valable pour des sorties structurées.

Reconnaissance d’objets

Reconnaissance d’objets

Réseaux de neurones profonds

Classifieur linéaire

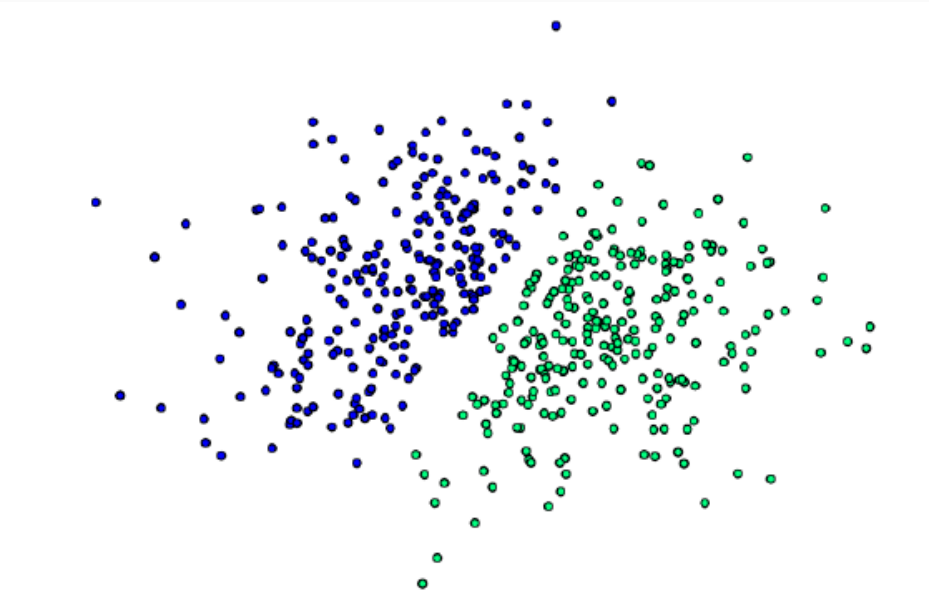
- Base de données : paires $(X^{(i)}, Y^{(i)})$, $i = 1, ..., N$.
- $X^{(i)} \in \mathbb{R}^n$, $Y^{(i)} \in \{-1, 1\}$
- Objectif : trouver w et b tels que $signe(w^T X^{(i)} + b) = Y^{(i)}$.

1

1. Figures : N. Leroux [2015]

Classifieur linéaire

- Base de données : paires $(X^{(i)}, Y^{(i)})$, $i = 1, ..., N$.
- $X^{(i)} \in \mathbb{R}^n$, $Y^{(i)} \in \{-1, 1\}$
- Objectif : trouver w et b tels que $signe(w^T X^{(i)} + b) = Y^{(i)}$.



1

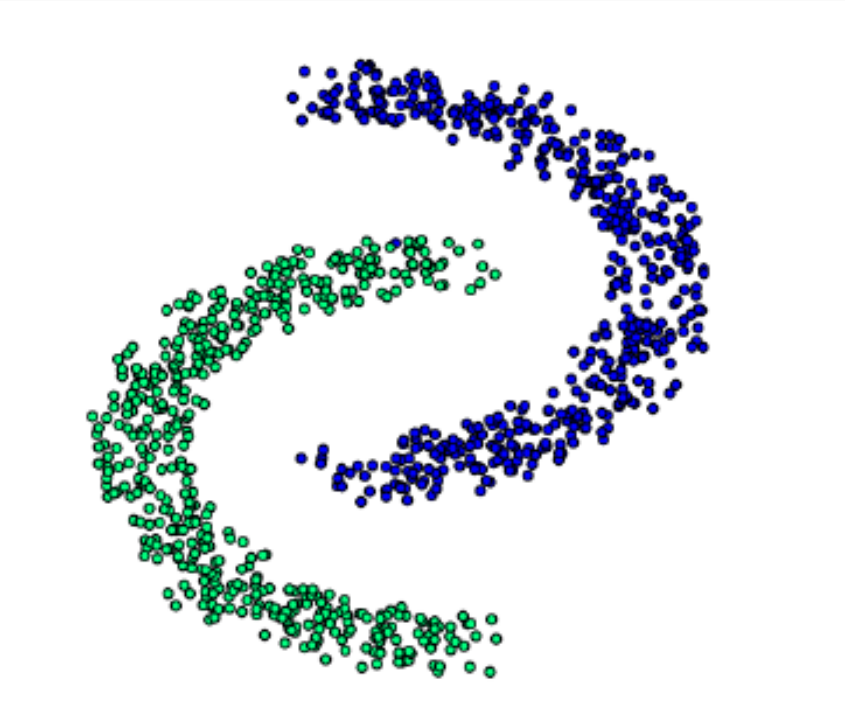
1. Figures : N. Leroux [2015]

Algorithme du perceptron (Rosenblatt, 1957)

- $w_0 = 0, b_0 = 0$
- $\hat{Y}^{(i)} = signe(w^T X^{(i)} + b)$
- $w_{t+1} \leftarrow w_t + \sum_i (Y^{(i)} - \hat{Y}^{(i)}) X^{(i)}$
- $b_{t+1} \leftarrow b_t + \sum_i (Y^{(i)} - \hat{Y}^{(i)})$

Demo linéairement séparable.

Certaines données ne sont pas linéairement séparables



L'algorithme du perceptron ne converge pas pour des données non linéairement séparables.

Non convergence de l'algorithme du perceptron

Perceptron non-linéairement séparable

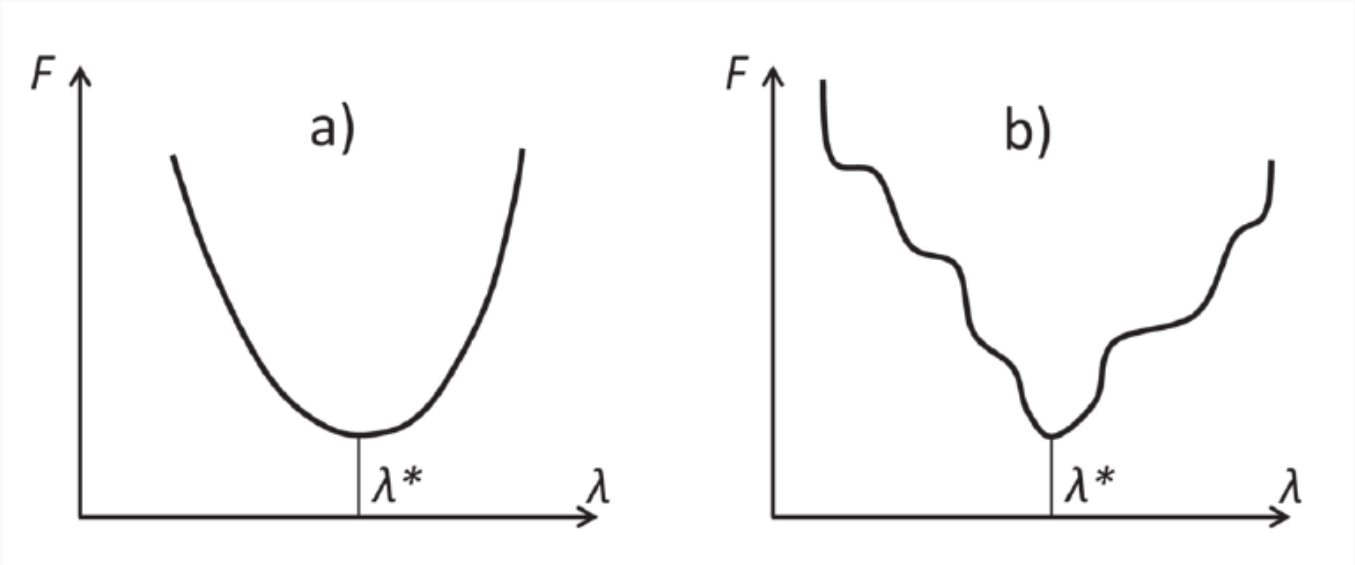
- On a besoin d'un algorithme qui marche à la fois sur des données séparables et non séparables.

Erreur de classification

Un bon classifieur minimise :

$$\begin{aligned}\ell(w) &= \sum_i \ell_i \\ &= \sum_i \ell(\hat{Y}^{(i)}, Y^{(i)}) \\ &= \sum_i \mathbb{1}_{signe(w^T X^{(i)} + b) \neq Y_i}\end{aligned}$$

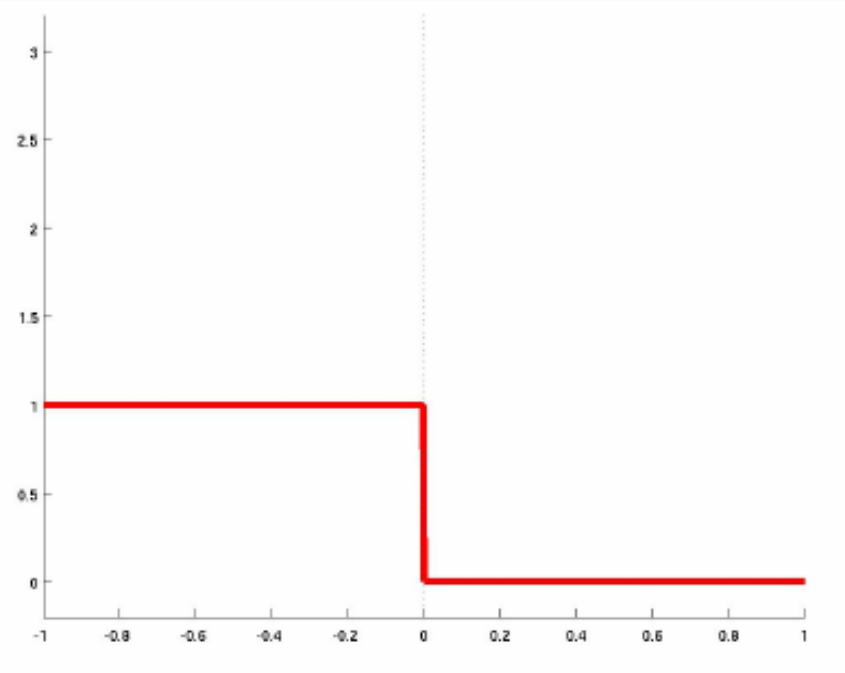
Importance de la convexité



a) Fonction convexe, b) Fonction non convexe mais adaptée à un problème d'optimisation

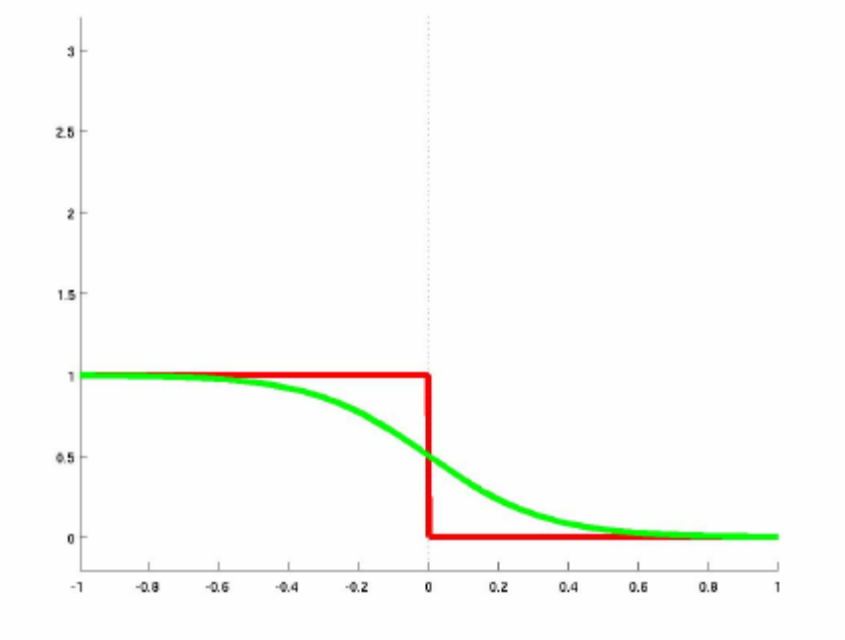
Fonction de coût

- L'erreur de classification n'est pas lisse.
- La sigmoïde est lisse mais pas convexe.
- La perte logistique est une borne supérieure convexe.
- La hinge loss ressemble beaucoup à la logistique.



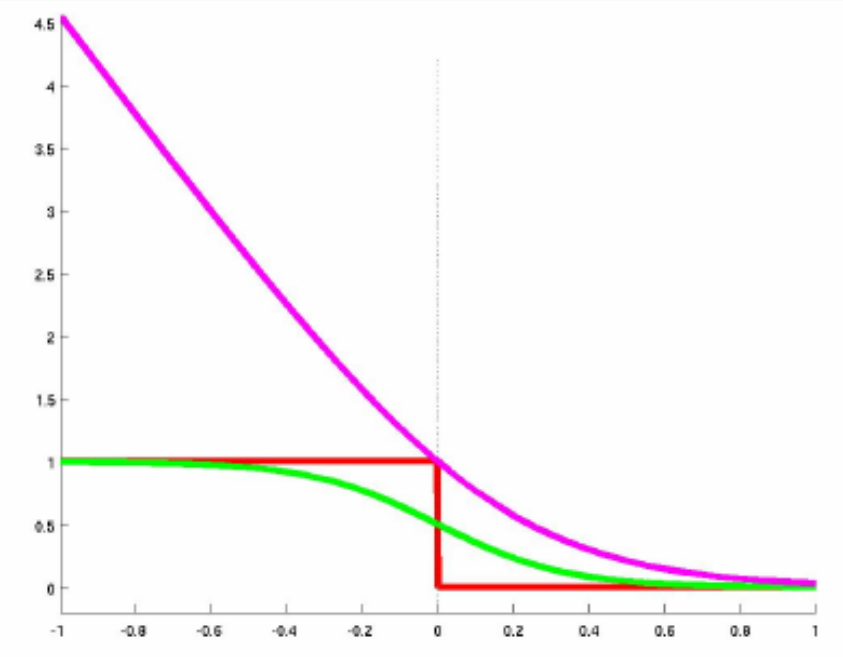
Fonction de coût

- L'erreur de classification n'est pas lisse.
- La sigmoïde est lisse mais pas convexe.
- La perte logistique est une borne supérieure convexe.
- La hinge loss ressemble beaucoup à la logistique.



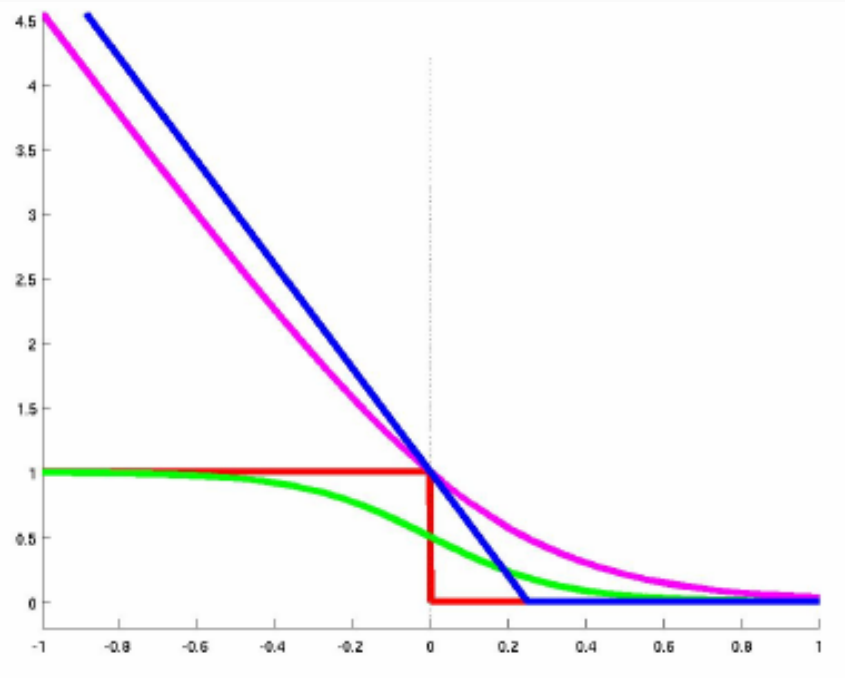
Fonction de coût

- L'erreur de classification n'est pas lisse.
- La sigmoïde est lisse mais pas convexe.
- La perte logistique est une borne supérieure convexe.
- La hinge loss ressemble beaucoup à la logistique.



Fonction de coût

- L'erreur de classification n'est pas lisse.
- La sigmoïde est lisse mais pas convexe.
- La perte logistique est une borne supérieure convexe.
- La hinge loss ressemble beaucoup à la logistique.



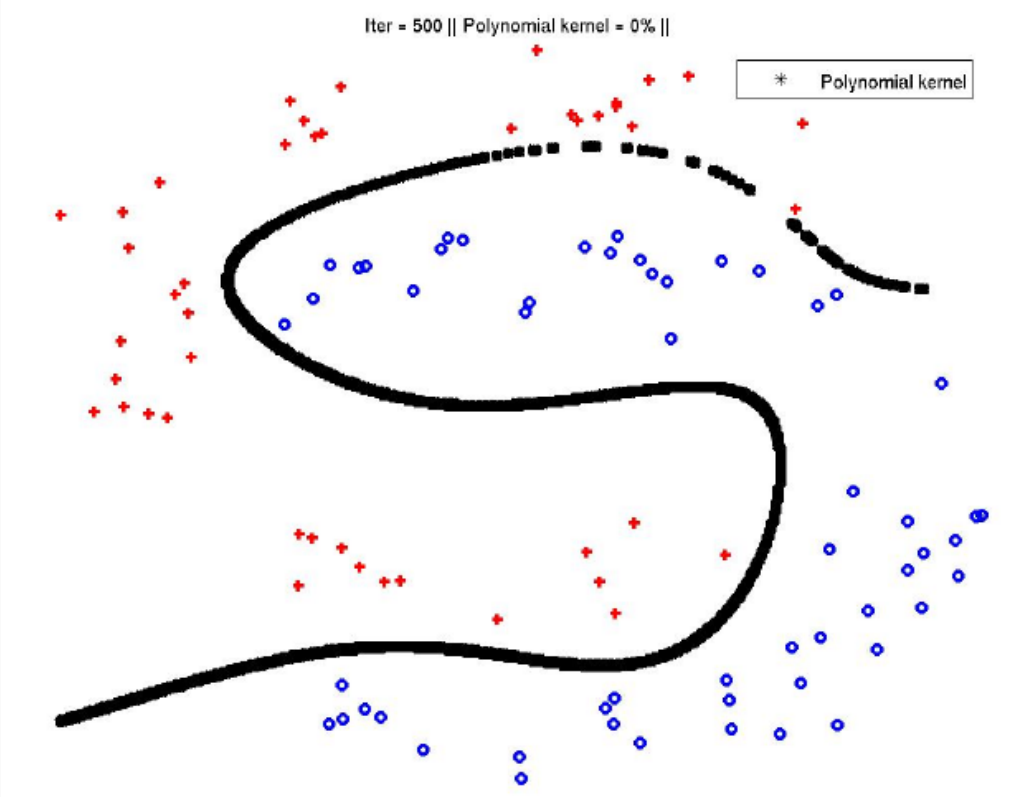
Résoudre des problèmes séparables ET non séparables

- *Fonction logistique non-linéairement séparable*
- *Fonction logistique linéairement séparable*

Classification non linéaire

Polynôme non-linéairement séparable

Classification non linéaire



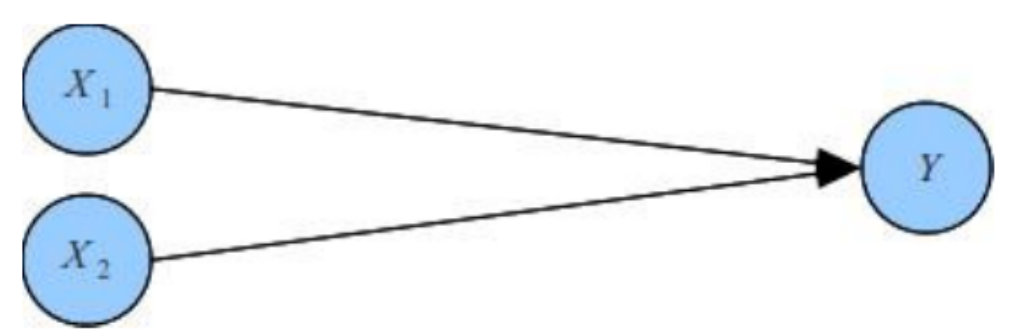
- Features : $X_1, X_2 \rightarrow$ classifieur linéaire
- Features : $X_1, X_2, X_1X_2, X_1^2, ... \rightarrow$ classifieur non-linéaire

Choisir les features

- Pour que ça fonctionne, on crée de nouveaux features :
- $X_1^i X_2^j$ pour $(i, j) \in [0, 4]$ ($p = 18$)
- Est-ce que ça fonctionnerait avec moins de features ?
- Test avec $X_1^i X_2^j$ pour $(i, j) \in [0, 3]$

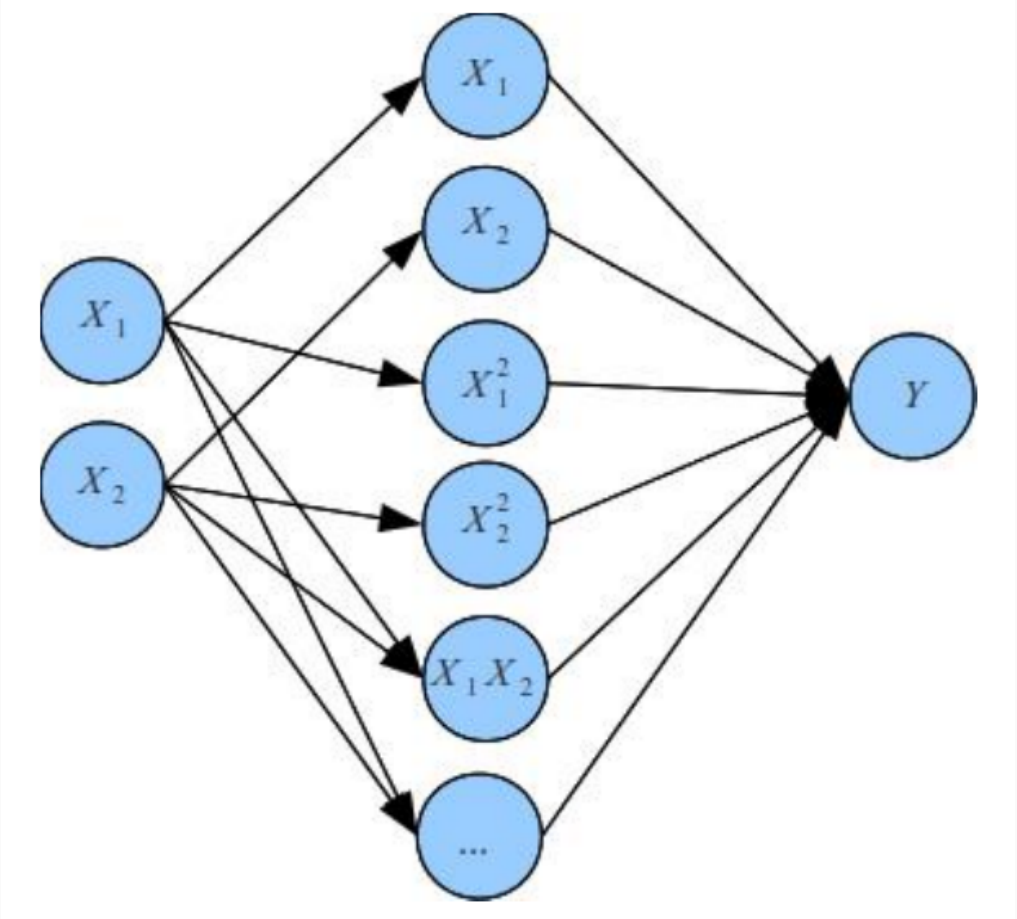
Polynôme non-linéairement séparable de degré 2

Une vue graphique des classifieurs



$$f(X) = w_1 X_1 + w_2 X_2 + b$$

Une vue graphique des classifieurs



$$f(X) = w_1X_1 + w_2X_2 + w_3X_1^2 + w_4X_2^2 + w_5X_1X_2 + \ldots$$

Features non-linéaires

- Un classifieur linéaire appliqué à des transformations non-linéaires est non-linéaire
- Un classifieur non-linéaire repose sur des features non-linéaires
- Lesquels choisir ?

- $H_j = X_1^{p_j} X_2^{q_j}$
- SVM : $H_j = K(X, X^{(j)})$ avec K fonction noyau
- Est-ce que les features doivent être prédéfinis ?
- Un réseau de neurones va apprendre les H_j

Features non-linéaires

- Un classifieur linéaire appliqué à des transformations non-linéaires est non-linéaire
- Un classifieur non-linéaire repose sur des features non-linéaires
- Lesquels choisir ?

- $H_j = X_1^{p_j} X_2^{q_j}$
- SVM : $H_j = K(X, X^{(j)})$ avec K fonction noyau
- Est-ce que les features doivent être prédéfinis ?
- Un réseau de neurones va apprendre les H_j

Features non-linéaires

- Un classifieur linéaire appliqué à des transformations non-linéaires est non-linéaire
- Un classifieur non-linéaire repose sur des features non-linéaires
- Lesquels choisir ?

- $H_j = X_1^{p_j} X_2^{q_j}$
- SVM : $H_j = K(X, X^{(j)})$ avec K fonction noyau
- Est-ce que les features doivent être prédéfinis ?
- Un réseau de neurones va apprendre les H_j

Features non-linéaires

- Un classifieur linéaire appliqué à des transformations non-linéaires est non-linéaire
- Un classifieur non-linéaire repose sur des features non-linéaires
- Lesquels choisir ?

- $H_j = X_1^{p_j} X_2^{q_j}$
- SVM : $H_j = K(X, X^{(j)})$ avec K fonction noyau
- Est-ce que les features doivent être prédéfinis ?
- Un réseau de neurones va apprendre les H_j

Features non-linéaires

- Un classifieur linéaire appliqué à des transformations non-linéaires est non-linéaire
- Un classifieur non-linéaire repose sur des features non-linéaires
- Lesquels choisir ?

- $H_j = X_1^{p_j} X_2^{q_j}$
- SVM : $H_j = K(X, X^{(j)})$ avec K fonction noyau
- Est-ce que les features doivent être prédéfinis ?
- Un réseau de neurones va apprendre les H_j

Algorithme pour réseau de neurones

- Choisir un exemple x
- Le transformer en $\hat{x} = Vx$ avec une matrice V
- Appliquer une transformation non-linéaire g à tous les éléments de $\hat{x}$
- Calculer $w^T \hat{x} + b$
- Calculer $w^T \hat{x} + b = w^T Vx + b = \hat{w}x + b$
- Calculer $w^T \hat{x} + b = w^T g(Vx) + b \neq \hat{w}x + b$

Algorithme pour réseau de neurones

- Choisir un exemple x
- Le transformer en $\hat{x} = Vx$ avec une matrice V
- Appliquer une transformation non-linéaire g à tous les éléments de $\hat{x}$
- Calculer $w^T \hat{x} + b$
- Calculer $w^T \hat{x} + b = w^T Vx + b = \hat{w}x + b$
- Calculer $w^T \hat{x} + b = w^T g(Vx) + b \neq \hat{w}x + b$

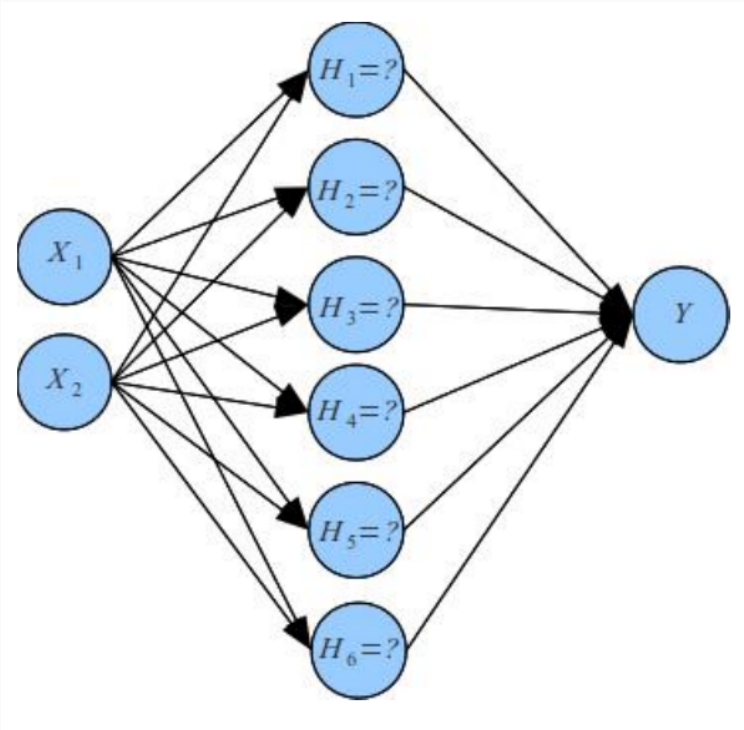
Algorithme pour réseau de neurones

- Choisir un exemple x
- Le transformer en $\hat{x} = Vx$ avec une matrice V
- Appliquer une transformation non-linéaire g à tous les éléments de $\hat{x}$
- Calculer $w^T \hat{x} + b$
- Calculer $w^T \hat{x} + b = w^T Vx + b = \hat{w}x + b$
- Calculer $w^T \hat{x} + b = w^T g(Vx) + b \neq \hat{w}x + b$

Algorithme pour réseau de neurones

- Choisir un exemple x
- Le transformer en $\hat{x} = Vx$ avec une matrice V
- Appliquer une transformation non-linéaire g à tous les éléments de $\hat{x}$
- Calculer $w^T \hat{x} + b$
- Calculer $w^T \hat{x} + b = w^T Vx + b = \hat{w}x + b$
- Calculer $w^T \hat{x} + b = w^T g(Vx) + b \neq \hat{w}x + b$

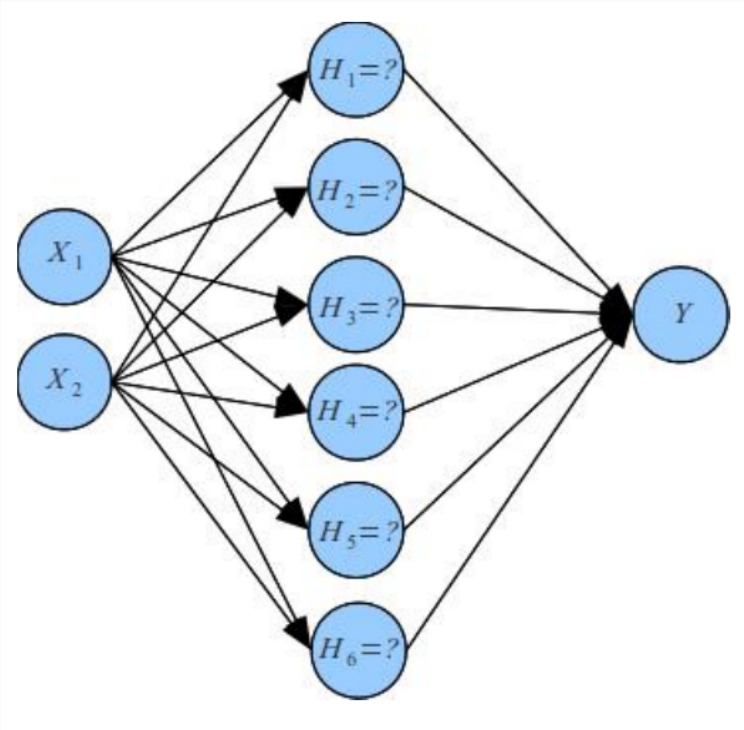
Réseaux de neurones



- Généralement, on utilise $H_j = g(v_j^T X)$

- H_j : Unité cachée
- v_j : Poids d'entrée
- g : Fonction de transfert

Réseaux de neurones

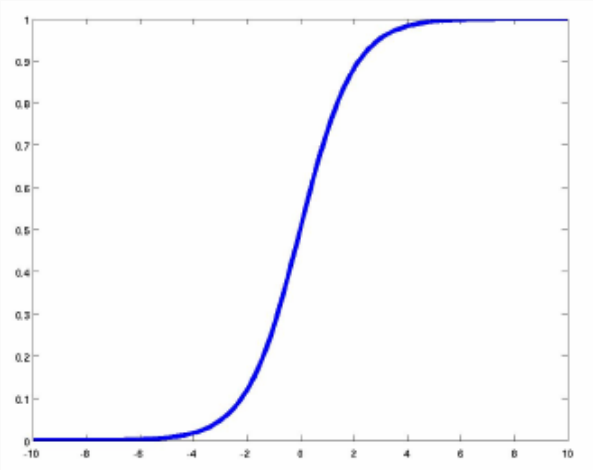


- Généralement, on utilise $H_j = g(v_j^T X)$
- H_j : Unité cachée
- v_j : Poids d'entrée
- g : Fonction de transfert

Fonctions de transfert

$$f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^T X)$$

- g est la *fonction de transfert*.
- g était auparavant une sigmoïde ou tanh

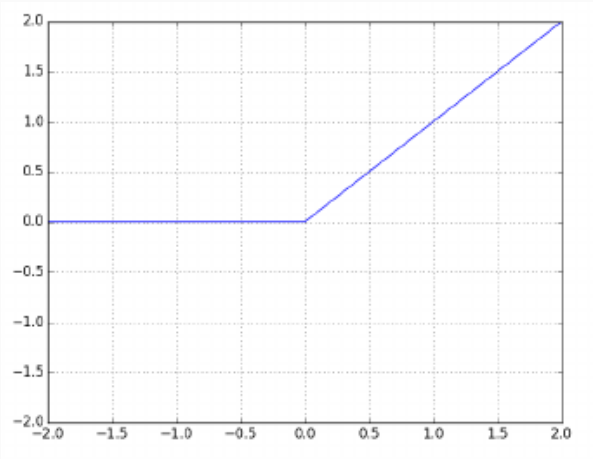


- Si g est une sigmoïde, chaque unité cachée est un *soft classifier*.

Fonctions de transfert

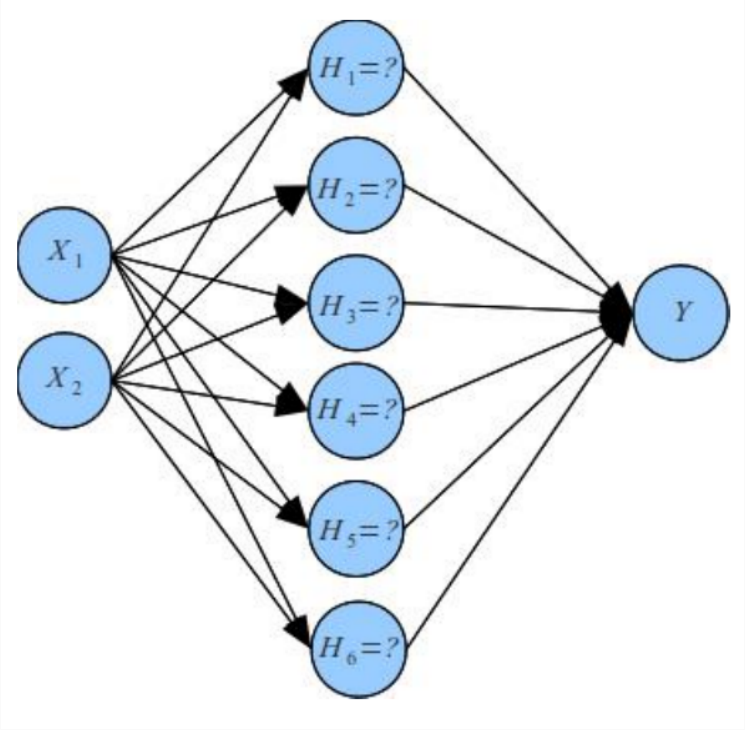
$$f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^T X)$$

- g est la *fonction de transfert*.
- g est désormais souvent la *partie positive*.



- On appelle cela une *rectified linear unit* (ReLU).

Réseaux de neurones



$$f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^T X)$$

Exemple sur le problème non-séparable

Réseau de neurones non-linéairement séparable 3

Entraîner un réseau de neurones

$$f(X) = \sum_j w_j g(v_j^T X)$$

$$f(X) = \sum_j W g(VX)$$

- Base de données : paires $(X^{(i)}, Y^{(i)})$, $i = 1, ..., N$
- Objectif : trouver V et W pour minimiser

$$\sum_i \ell(f(X^{(i)}), Y^{(i)}) = \sum_i \ell(Wg(VX^{(i)}), Y^{(i)})$$

- On peut le faire avec une descente de gradient.

Entraîner un réseau de neurones

$$f(X) = \sum_j w_j g(v_j^T X)$$

$$f(X) = \sum_j W g(VX)$$

- Base de données : paires $(X^{(i)}, Y^{(i)})$, $i = 1, ..., N$
- Objectif : trouver V et W pour minimiser

$$\sum_i \ell(f(X^{(i)}), Y^{(i)}) = \sum_i \ell(Wg(VX^{(i)}), Y^{(i)})$$

- On peut le faire avec une descente de gradient.

Entraîner un réseau de neurones

$$f(X) = \sum_j w_j g(v_j^T X)$$

$$f(X) = \sum_j W g(VX)$$

- Base de données : paires $(X^{(i)}, Y^{(i)})$, $i = 1, ..., N$
- Objectif : trouver V et W pour minimiser

$$\sum_i \ell(f(X^{(i)}), Y^{(i)}) = \sum_i \ell(Wg(VX^{(i)}), Y^{(i)})$$

- On peut le faire avec une descente de gradient.

Rétropropagation du gradient - trouver W

- Objectif : trouver V et W pour minimiser

$$\sum_i \ell(Wg(VX^{(i)}), Y^{(i)})$$

- On doit calculer le gradient de $\ell_i(W, V) = \ell(Wg(VX^{(i)}), Y^{(i)})$
- Dérivation de fonctions composées :

$$\frac{\partial \ell_i(W,V)}{\partial W} = \frac{\partial \ell_i(W,V)}{\partial f(X)} \frac{\partial f(X)}{\partial W}$$
$$= \frac{\partial \ell_i(W,V)}{\partial f(X)} g(VX)^T$$

Rétropropagation du gradient - trouver W

- Objectif : trouver V et W pour minimiser

$$\sum_i \ell(Wg(VX^{(i)}), Y^{(i)})$$

- On doit calculer le gradient de $\ell_i(W, V) = \ell(Wg(VX^{(i)}), Y^{(i)})$
- Dérivation de fonctions composées :

$$\frac{\partial \ell_i(W,V)}{\partial W} = \frac{\partial \ell_i(W,V)}{\partial f(X)} \frac{\partial f(X)}{\partial W}$$
$$= \frac{\partial \ell_i(W,V)}{\partial f(X)} g(VX)^T$$

Rétropropagation du gradient - trouver W

- Objectif : trouver V et W pour minimiser
$$\sum_i \ell(Wg(VX^{(i)}), Y^{(i)})$$
- On doit calculer le gradient de
$\ell_i(W, V) = \ell(Wg(VX^{(i)}), Y^{(i)})$
- Dérivation de fonctions composées :
$$\begin{aligned}\frac{\partial \ell_i(W,V)}{\partial W} &= \frac{\partial \ell_i(W,V)}{\partial f(X)} \frac{\partial f(X)}{\partial W} \\ &= \frac{\partial \ell_i(W,V)}{\partial f(X)} g(VX)^T\end{aligned}$$

Rétropropagation du gradient - trouver V

- On réécrit $Wg(VX) = WH$ avec $H = g(VX)$.
- Dérivation de fonctions composées :

$$\begin{aligned}\frac{\partial \ell_i(W,V)}{\partial V} &= \frac{\partial \ell_i(W,V)}{\partial f(X)} \frac{\partial f(X)}{\partial V} \\ &= \frac{\partial \ell_i(W,V)}{\partial f(X)} \frac{\partial f(X)}{\partial H} \frac{\partial H}{\partial V} \\ &= W \frac{\partial \ell_i(W,V)}{\partial f(X)} g'(VX) X^T\end{aligned}$$

Rétropropagation du gradient - trouver V

- On réécrit $Wg(VX) = WH$ avec $H = g(VX)$.
- Dérivation de fonctions composées :

$$\begin{aligned}\frac{\partial \ell_i(W,V)}{\partial V} &= \frac{\partial \ell_i(W,V)}{\partial f(X)} \frac{\partial f(X)}{\partial V} \\ &= \frac{\partial \ell_i(W,V)}{\partial f(X)} \frac{\partial f(X)}{\partial H} \frac{\partial H}{\partial V} \\ &= W \frac{\partial \ell_i(W,V)}{\partial f(X)} g'(VX) X^T\end{aligned}$$

Trouver une bonne transformation de x

- $H_j(x) = g(v_j^T x)$
- Est-ce que ce sont les bons H_j ? Que sait-on sur cela ?
- Théoriquement, un nombre fini de H_j est suffisant (théorème d'approximation universelle).
- Cela ne veut pas dire qu'il faut l'utiliser $\rightarrow$ le nombre de H_j nécessaire peut être excessivement grand.

Trouver une bonne transformation de x

- $H_j(x) = g(v_j^T x)$
- Est-ce que ce sont les bons H_j ? Que sait-on sur cela ?
- Théoriquement, un nombre fini de H_j est suffisant (théorème d'approximation universelle).
- Cela ne veut pas dire qu'il faut l'utiliser $\rightarrow$ le nombre de H_j nécessaire peut être excessivement grand.

Trouver une bonne transformation de x

- $H_j(x) = g(v_j^T x)$
- Est-ce que ce sont les bons H_j ? Que sait-on sur cela ?
- Théoriquement, un nombre fini de H_j est suffisant (théorème d'approximation universelle).
- Cela ne veut pas dire qu'il faut l'utiliser $\rightarrow$ le nombre de H_j nécessaire peut être excessivement grand.

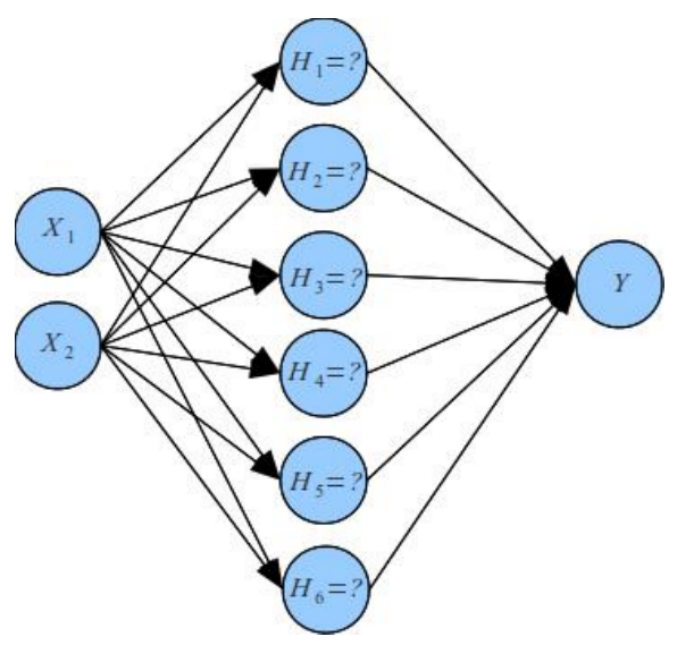
Approfondir le réseau

- $H_j(x) = g(v_j^T x)$
- $\hat{x}_j = g(v_j^T x)$
- On peut aussi transformer $\hat{x}$

Approfondir le réseau

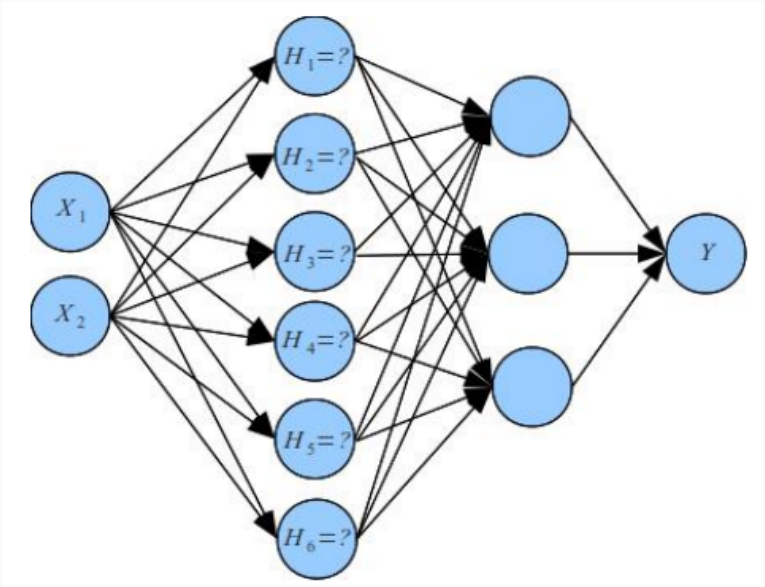
- $H_j(x) = g(v_j^T x)$
- $\hat{x}_j = g(v_j^T x)$
- On peut aussi transformer $\hat{x}$

De 2 à 3 couches



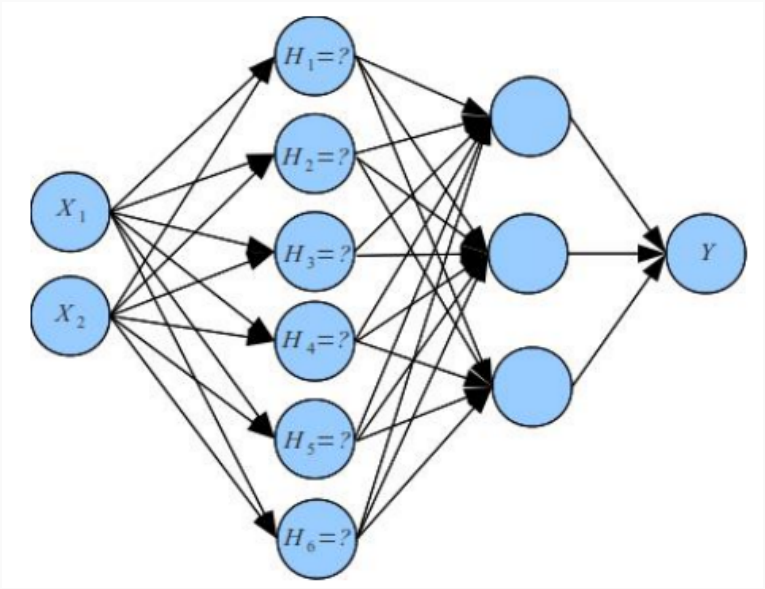
- On peut avoir autant de couches qu'on veut.
- Mais cela rend le problème d'optimisation plus dur.

De 2 à 3 couches



- On peut avoir autant de couches qu'on veut.
- Mais cela rend le problème d'optimisation plus dur.

De 2 à 3 couches



- On peut avoir autant de couches qu'on veut.
- Mais cela rend le problème d'optimisation plus dur.

Optimisation et régularisation

Plan

Nécessité d'un apprentissage rapide

- Les réseaux de neurones ont besoin de beaucoup d'exemples (quelques millions ou plus).
- On doit pouvoir les utiliser rapidement.

Descente de gradient : méthode batch

$$L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)})$$

$$\theta_{t+1} \to \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$$

- Pour calculer tous les paramètres, on a besoin de parcourir toutes les données.
- Cela peut être très coûteux.
- Que se passe-t'il si on a un nombre de données infini ?

Descente de gradient : méthode batch

$$L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)})$$

$$\theta_{t+1} \to \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$$

- Pour calculer tous les paramètres, on a besoin de parcourir toutes les données.
- Cela peut être très coûteux.
- Que se passe-t'il si on a un nombre de données infini ?

Descente de gradient : méthode batch

$$L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)})$$

$$\theta_{t+1} \to \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$$

- Pour calculer tous les paramètres, on a besoin de parcourir toutes les données.
- Cela peut être très coûteux.
- Que se passe-t'il si on a un nombre de données infini ?

Que faire alors ?

1. Supprimer des données ?
2. Utiliser des méthodes d'approximation.
   - Mise à jour des poids = moyenne des mises à jour pour tous les points.
   - Est-ce que ces mises à jour sont vraiment différentes ?
   - Si non, comment peut-on apprendre plus rapidement ?

Que faire alors ?

1. Supprimer des données ?
2. Utiliser des méthodes d'approximation.
    - Mise à jour des poids = moyenne des mises à jour pour tous les points.
    - Est-ce que ces mises à jour sont vraiment différentes ?
    - Si non, comment peut-on apprendre plus rapidement ?

Descente de gradient stochastique

- $L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)})$
- $\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$
- $\theta_{t+1} \rightarrow \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}$
- Que perd-t'on lorsque on met à jour les paramètres pour satisfaire à un seul exemple ?

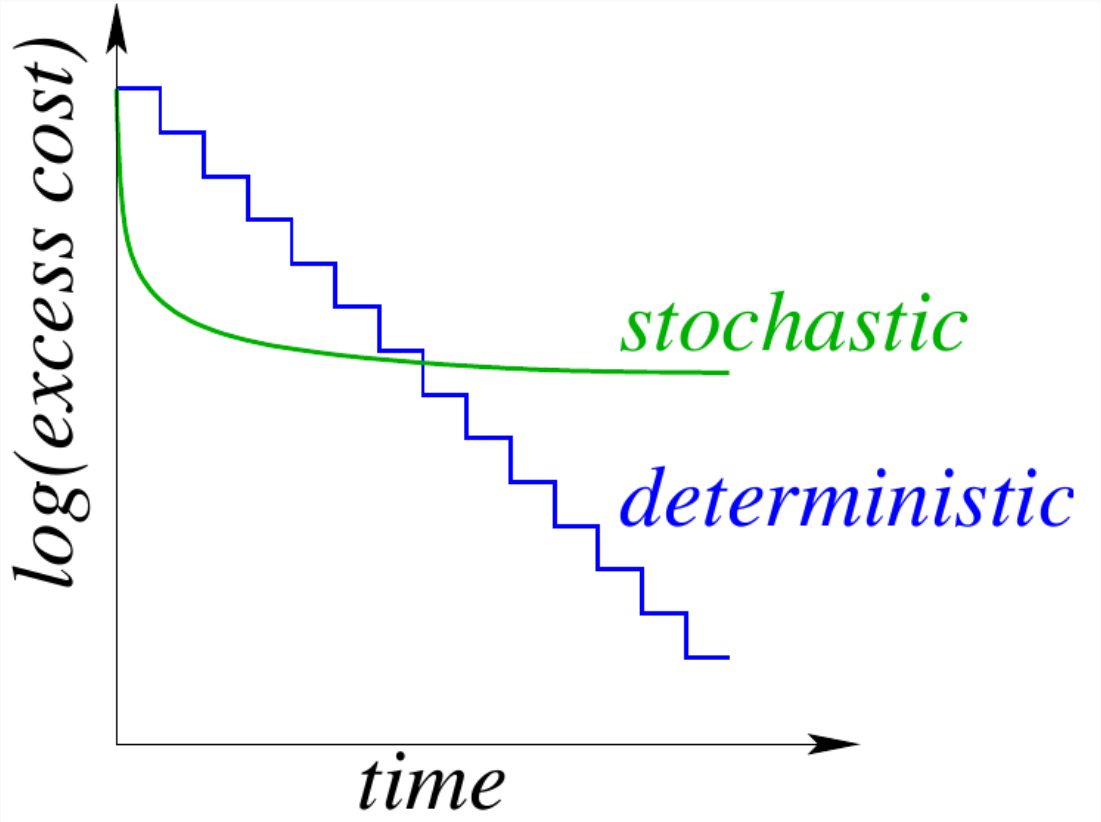
Descente de gradient stochastique

- $L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)})$
- $\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$
- $\theta_{t+1} \rightarrow \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}$
- Que perd-t'on lorsque on met à jour les paramètres pour satisfaire à un seul exemple ?

Descente de gradient stochastique

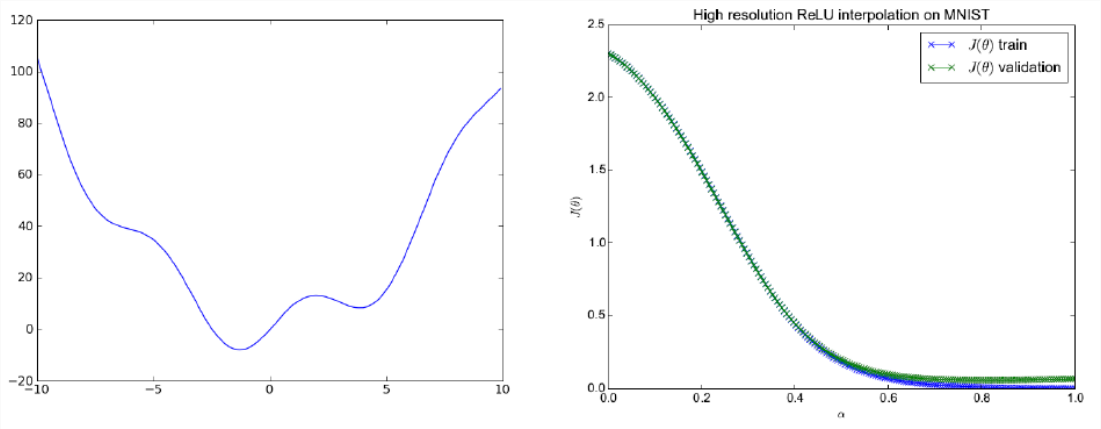
- $L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)})$
- $\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$
- $\theta_{t+1} \rightarrow \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}$
- Que perd-t'on lorsque on met à jour les paramètres pour satisfaire à un seul exemple ?

Batch vs stochastique



- Pour les problèmes non-convexes, le stochastique marche le mieux.

Comprendre la fonction de perte des réseaux profonds



- Des études récentes disent que la plupart des minima locaux sont proches du minimum global.

Plan

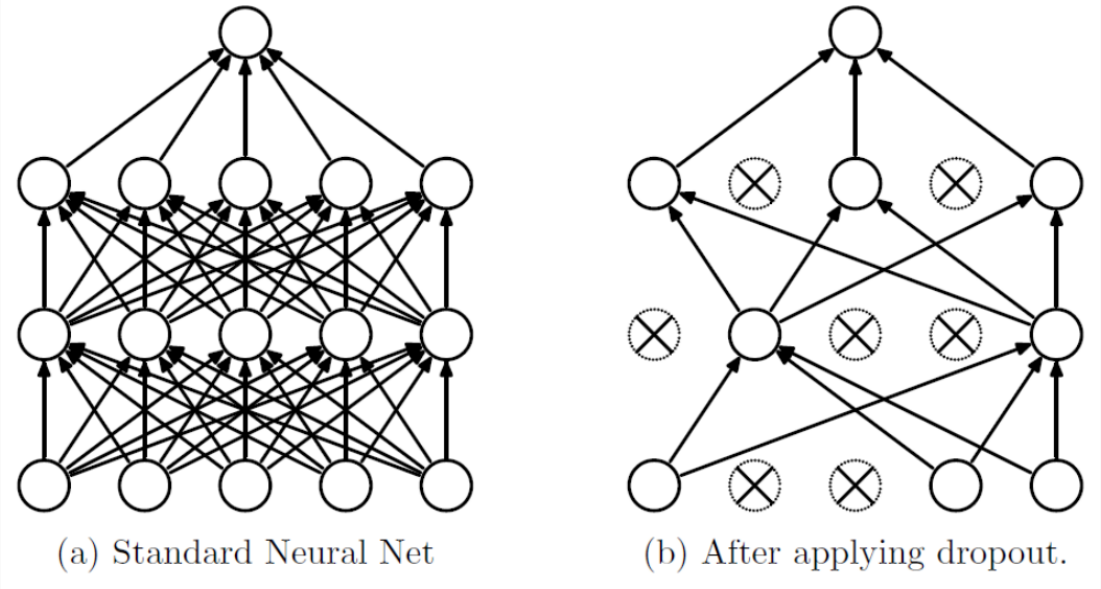
Régulariser les réseaux profonds

- Les réseaux profonds ont de nombreux paramètres.
- Comment éviter le surrapprentissage ?
  - En régularisant les paramètres ?
  - En limitant le nombre d'unités
  - Dropout

Dropout

- Le surrapprentissage intervient quand chaque unité devient trop spécialisée.
- L'idée est d'empêcher chaque unité de trop s'appuyer sur les autres.
- Comment ?

Dropout - illustration



Dropout : A Simple Way to Prevent Neural Networks from Overfitting

Dropout - conclusion

- En supprimant les unités au hasard, on force les autres à être moins spécialisées.
- Au moment du test, on utilise toutes les unités.
- Parfois présenté comme une méthode ensembliste.

Autres techniques utilisées

- Local response normalization
- Data augmentation
- Batch gradient normalization

Conclusion

Réseaux de neurones - Résumé

- Un classifieur linéaire dans un espace de features adapté peut modéliser des frontières non-linéaires.
- Trouver un bon espace de représentation (features space) est essentiel.
- On peut définir cet espace à la main.
- On peut apprendre cet espace en utilisant moins de features.
- Apprendre cet espace est potentiellement compliqué (non-convexité).

Conclusions

- Les réseaux de neurones sont des modèles non-linéaires complexes.
- Les optimiser est difficile.
- Il s'agit autant de théorie que d'ingénierie.
- Lorsqu'ils sont correctement paramétrés, ils peuvent donner de très bons résultats.

Bibliographie

- Deep Learning book http ://www-labs.iro.umontreal.ca/bengioy/DLbook/
- Dropout : A Simple Way to Prevent Neural Networks from Overfitting, by Srivistave et al.
- Hugo Larochelle MOOC

  https ://www.youtube.com/playlist ?list=PL6Xpj9I5qXYEcOhn7TqghAJ6NAPrNmUBH
- A Neural Network Playground http ://playground.tensorflow.org/