Machine Learning avancé
Optimisation numérique : application au perceptron

Maxime Jumelle

ESLSCA Big Data - MBA 2

2019 - 2020

Pourquoi ?

L'optimisation numérique est le domaine des mathématiques et de l'informatique qui a pour objectif de résoudre des **problèmes de minimisation (ou maximisation) de fonctions**.

C'est une clé de voûte du Machine Learning (mais pas uniquement) et une très large majorité de modèles ont recours à des **solveurs numériques**.

Sommaire

Problème d'optimisation

On se donne une fonction $f : \mathbb{R}^p \to \mathbb{R}$. On se fixe est contraine d'inégalité $g : \mathbb{R}^p \to \mathbb{R}^p$ et d'égalité $h : \mathbb{R}^p \to \mathbb{R}^p$. Un problème d'optimisation est un problème de minimisation de la forme suivante

$$\min_{\mathbf{x} \in E} f(\mathbf{x})$$

avec

$$E = \{\mathbf{x} \in \mathbb{R}^p : g(x) \leq 0, h(\mathbf{x}) = 0\}$$

Formellement, on cherche donc la (ou les) solutions $\mathbf{x}^*$ qui minimise la fonction f sous les conditions d'inégalités et d'égalités imposées.

Optimisation numérique

Bien souvent, il arrive que la fonction à minimiser f ou que les contraintes imposées rendent le problème impossible à résoudre de manière analytique.

Pour contourner ce problème, on a recours à des **solveurs numériques** : ce sont des algorithmes qui vont essayer d'approcher la solution optimale en respectant les contraintes imposées. Le défaut est que la solution obtenue **n'est pas forcément la solution unique analytique**. Malheureusement, c'est bien souvent la seule méthode dont on dispose.

Mais nous allons voir que dans la plupart des cas, la solution approchée est en réalité satisfaisante, et que son utilisation n'est pas si contraignante.

Descente de gradient

La descente de gradient est un algorithme d'optimisation numérique de **premier ordre** qui est à la base d'une grande majorité de solveurs utilisés dans la pratique. La descente de gradient par de la notion intuitive qui consiste à utiliser la **direction inverse du gradient** de la fonction à minimiser, de sorte à chercher la position où ce dernier s'annule.

Descente de gradient

Algorithm 1: Algorithme de descente du gradient à pas constant

Input: f différentiable, initialisation x_0, pas η
Output: Solution approchée de $\nabla f(x) = 0$

$x = x_0$
while $\|\nabla f(x)\| > \varepsilon$ **do**
    Mise à jour de la solution :

$$x = x - \eta \nabla f(x)$$

return x

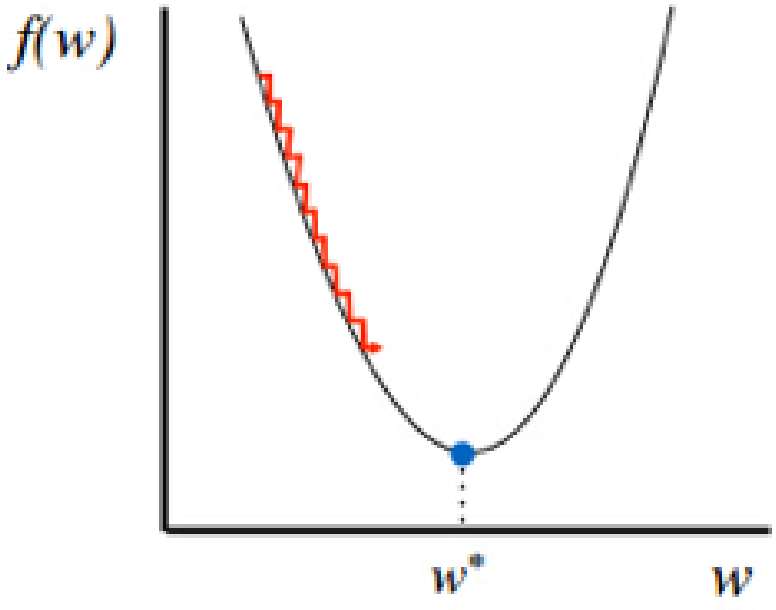
Choix du pas

Lorsque l'on détermine un pas fixe, on est face à un problème : ce dernier peut ne pas toujours s'adapter à la fonction rencontrée.

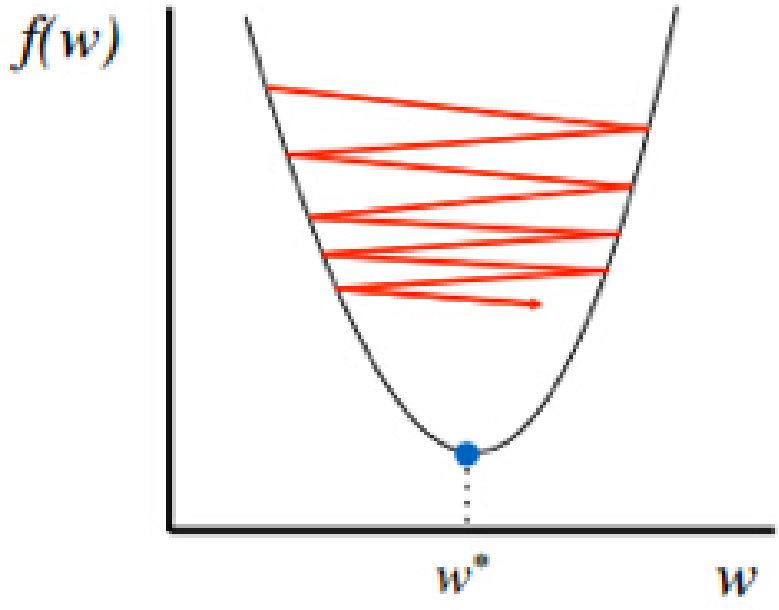
- Si le pas est trop grand, on peut ne jamais approcher la solution optimale ;
- à l'inverse, si le pas est trop petit, la convergence vers la solution optimale peut être trop longue.

Le cas optimal serait donc de **déterminer automatiquement** un pas fixe avant le lancement de l'algorithme.

Choix du pas

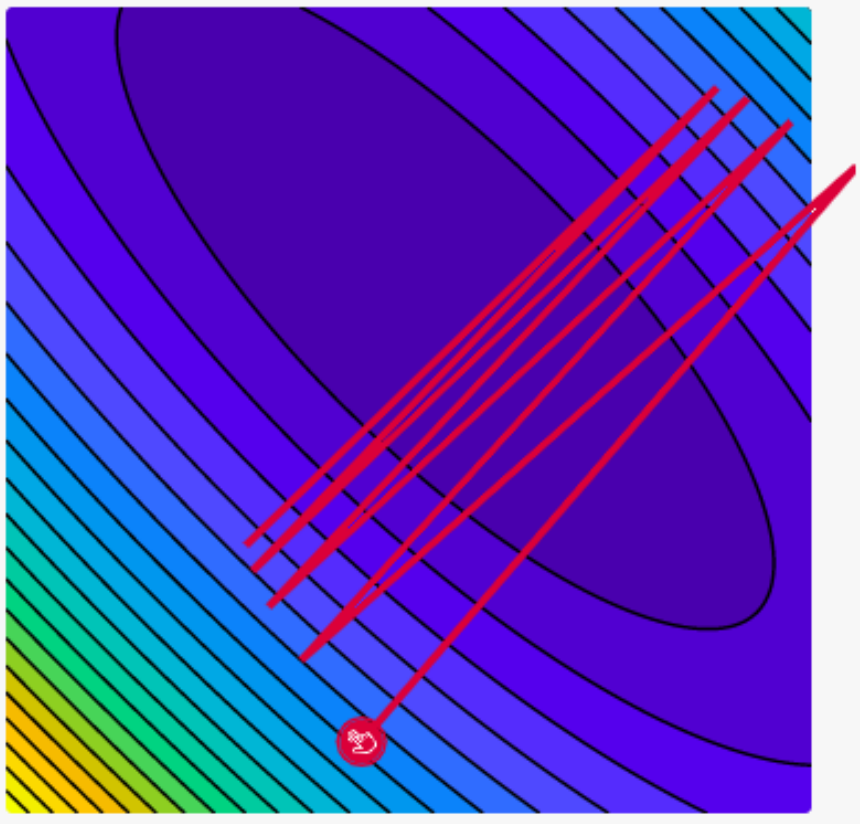


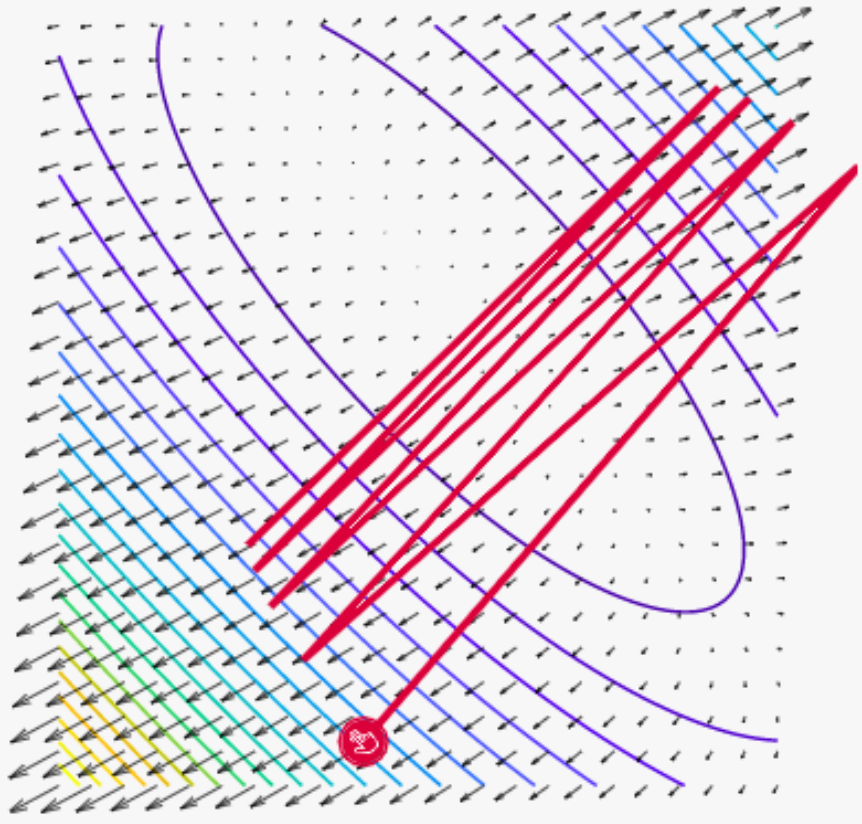
Too small: converge very slowly



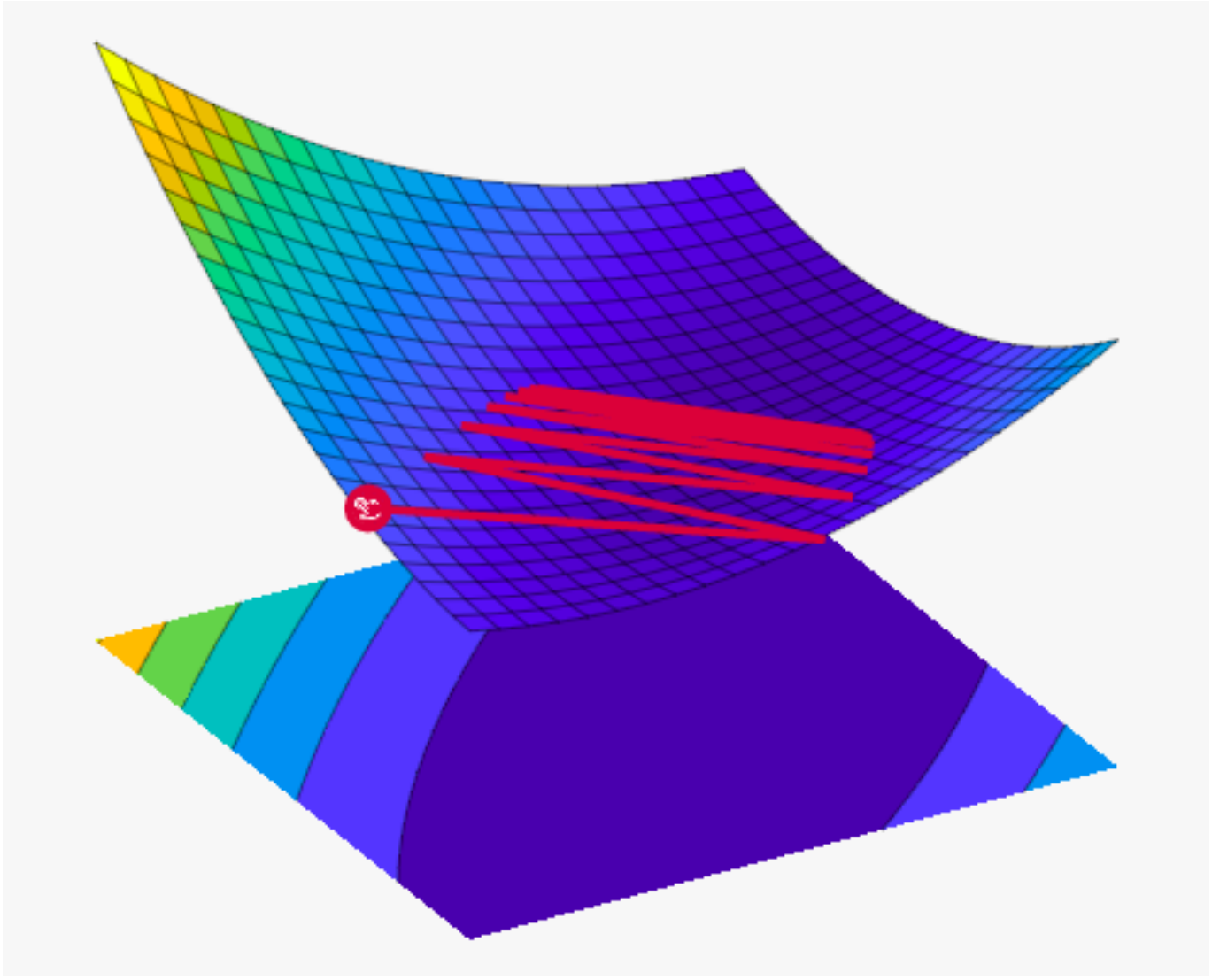
Too big: overshoot and even diverge

Choix du pas





Choix du pas



Descente de gradient

Algorithm 2: Algorithme de descente du gradient

Input: f différentiable, initialisation x_0
Output: Solution approchée de $\nabla f(x) = 0$
$x = x_0$
while $\|\nabla f(x)\| > \varepsilon$ **do**
    Calcul d'un pas optimal η
    Mise à jour de la solution :

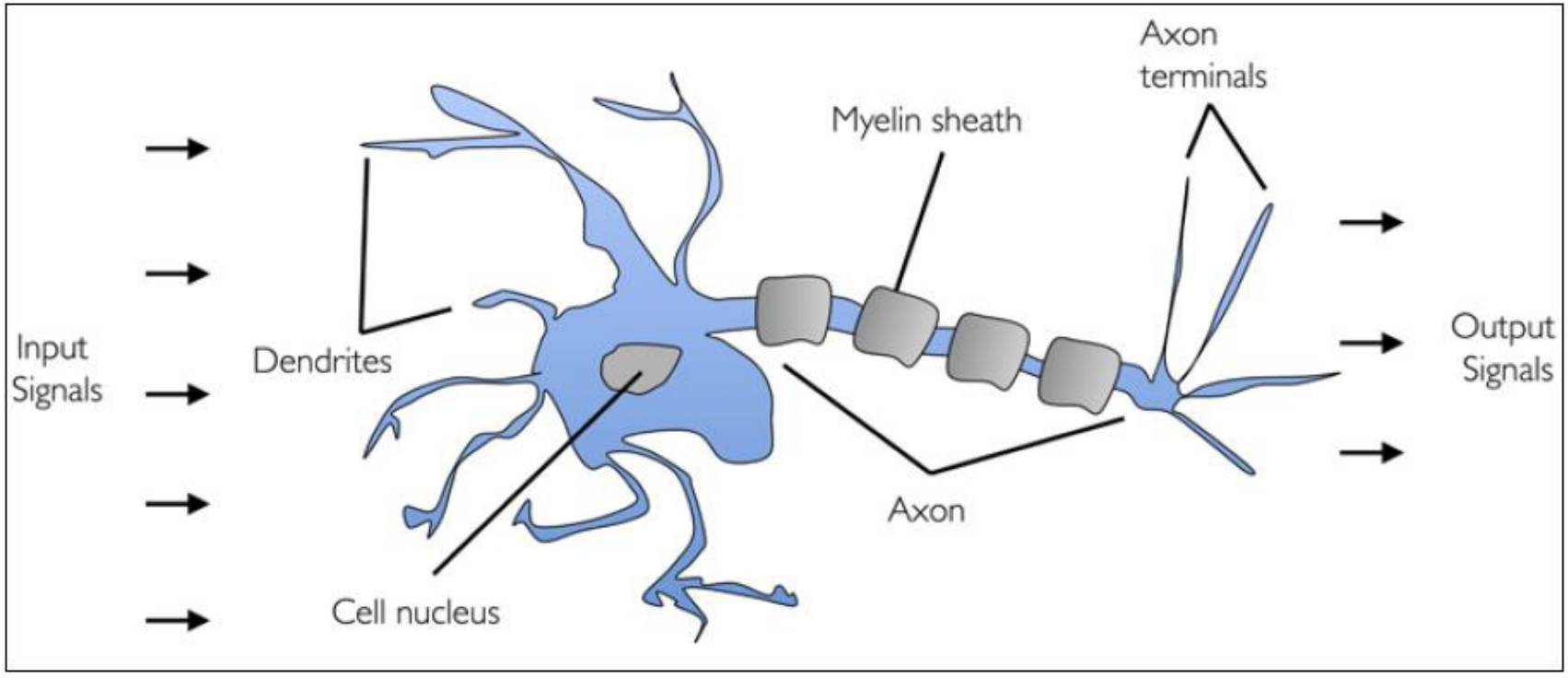
$$x = x - \eta \nabla f(x)$$

return x

Sommaire

Neurone biologique

Le perceptron est un modèle mathématique qui a pour objectif de représenter artificiellement le fonctionnement d'un neurone biologique, bien que ce dernier soit évidemment bien plus complexe que le formalisme mathématique associé.



Neurone artificiel

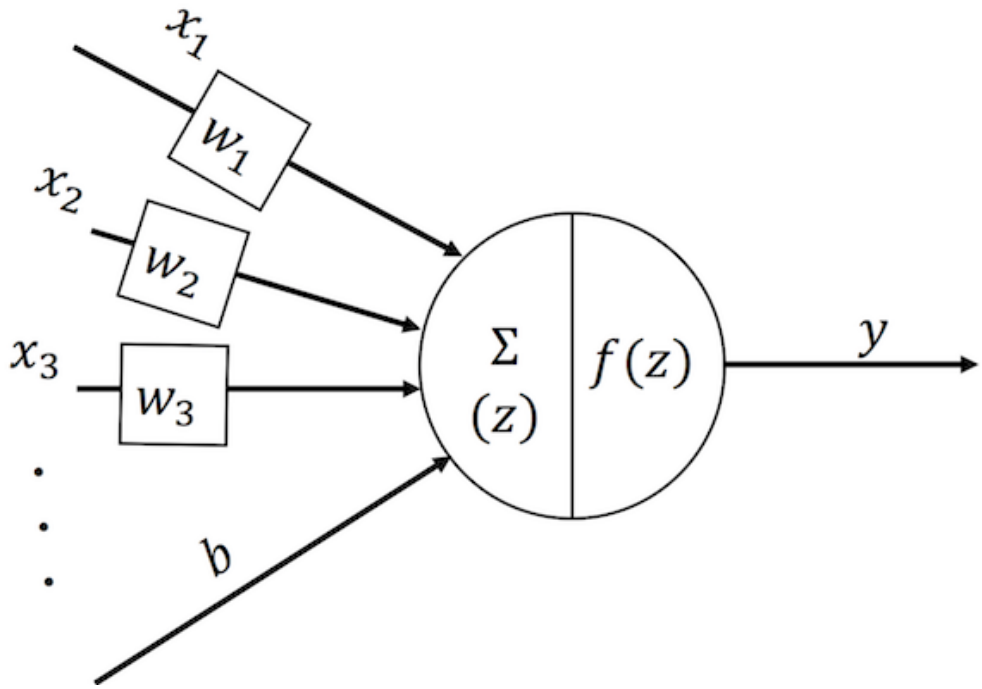
Dans les années 1950, Frank Rosenblatt essayait de comprendre et quantifier l'information que pouvait appréhender une mouche par ses yeux. Il se basait alors sur le modèle du neurone de McCulloch-Pitts, qui est aujourd'hui à la base des réseaux de neurones d'aujourd'hui.

FIGURE – Frank Rosenblatt

Premier modèle

Un premier modèle a été proposé avec les caractéristiques suivantes :

- On considère, en entrée, un vecteur de taille p $(x_1, \ldots, x_p)$.
- Chaque composante est convoluée par le vecteur des poids $(w_1, \ldots, w_p)$ en plus d'un biais initial w_0.
- On applique une fonction dite **d'activation** f à cette somme, ce qui nous donne le résultat.



Premier modèle

Le modèle s'écrit donc :

$$y = f\left(w_0 + \sum_{j=1}^{p} w_j x_j\right)$$

On cherche ainsi à **estimer les poids** $\mathbf{w} = (w_0, w_1, \ldots, w_p)$, la fonction d'activation f étant **fixée à l'avance** et les composantes de x connues. Plusieurs questions nous viennent à l'esprit :

- Comment choisir une fonction d'activation f appropriée ?
- Comment peut-on optimiser les poids $\mathbf{w}$?

Fonctions d'activation

En théorie, nous pourrions placer n'importe quelle fonction d'activation f. Cependant, on souhaite qu'elle soit (en partie) différentiable, mais surtout qu'elle soit en adéquation avec la problématique considérée. Par exemple, si l'on souhaitait réaliser une régression linéaire, quelle serait le bon choix de la fonction d'activation f ?

Pour $f = \mathrm{Id}$, on obtient

$$y = w_0 + \sum_{j=1}^{p} w_j x_j$$

ce qui correspond au modèle de régression linéaire multiple (où les β_i sont ici des w_i).

Fonctions d'activation

Un autre exemple que nous avons vu auparavant est la régression logistique :

$$\log \frac{\mathbb{P}(Y=1|X=\mathbf{x})}{1-\mathbb{P}(Y=1|X=\mathbf{x})} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j = \beta^\top \mathbf{x}$$

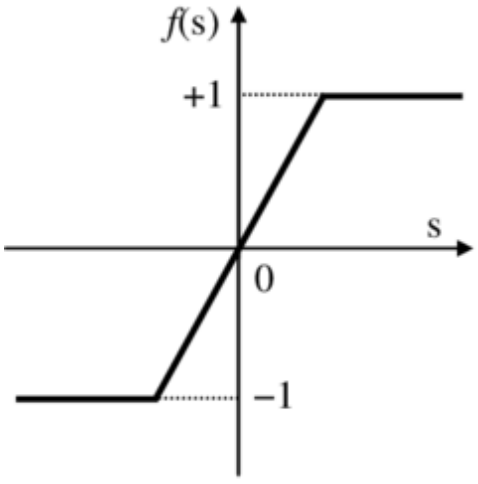
Si $y = \mathbb{P}(Y=1|X=\mathbf{x})$ et $\forall 0 \leq j \leq p, \beta_j = w_j$ on a

$$\log \frac{y}{1-y} = w_0 + \sum_{j=1}^{p} w_j x_j$$

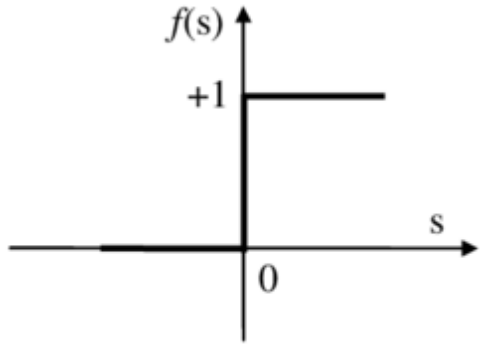
Exercice

Quelle est ici la fonction d'activation f ?

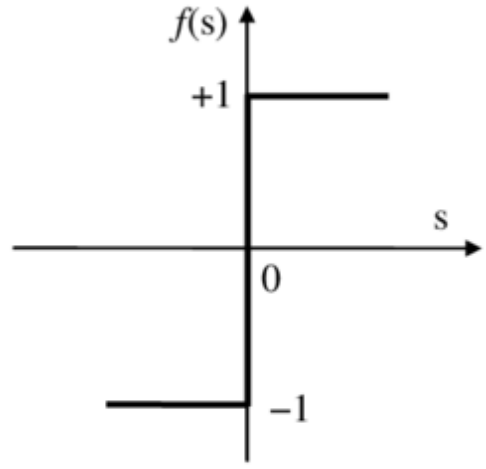
Quelques fonctions d'activation



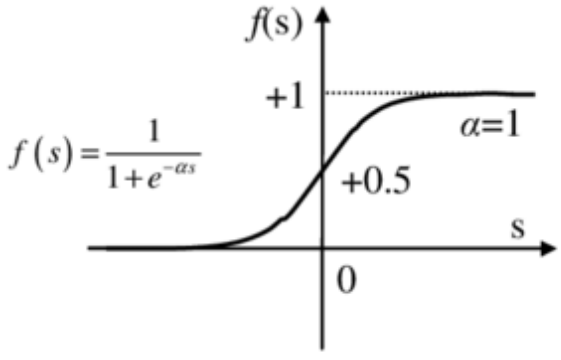
(a) fonction bipolaire linéaire



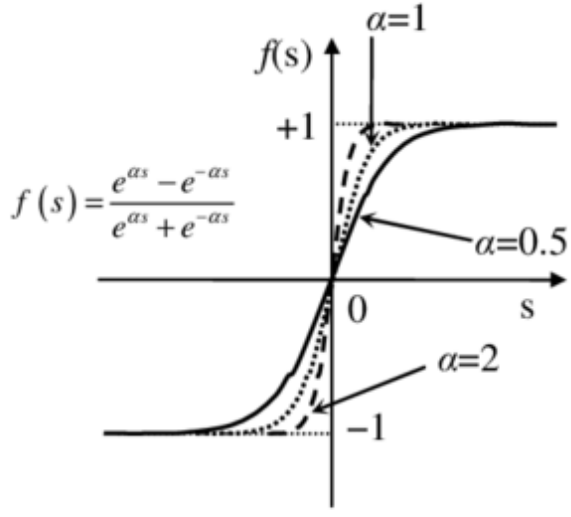
(b) fonction de seuil



(c) fonction de seuil bipolaire



(d) fonction sigmoïde



(e) fonction tangente hyperbolique

Poids optimaux

L'avantage du perceptron, c'est qu'il est naturellement adapté pour des **problèmes de classification et de régression**, là où il était nécessaire de relâcher certaines hypothèses sur le modèle linéaire pour l'étendre à des problèmes plus généraux.

Mais cette flexibilité a pour conséquence de ne pas fournir de solution optimale (si elle existe) ! En effet, étant donné la solution optimale $\mathbf{w}^*$ dépends de la fonction d'activation f, il faut calculer une solution pour chaque fonction d'activation. C'est alors que les méthodes d'optimisation numérique nous viennent en aide.

Fonction de perte

Pour trouver les poids optimaux $\mathbf{w}^*$, il nous faut donc une mesure de "coût" $\mathcal{L}$, appelée **fonction de perte**, qui nous indique si nous nous rapprochons des labels observés, ou non, et donc que le modèle est fidèle pour retranscrire la réalité.

$$\min_{\mathbf{w}} \mathcal{L}(Y, f(X))$$

Nous avons déjà vu l'idée qui se cache derrière cette fonction de perte lors de la régression linéaire : nous avions mesuré l'écart entre les labels observés Y et les labels prédits $\hat{Y}$, et on mesurait l'erreur par la norme suivante :

$$\|Y - f(X)\|_2^2 = \mathcal{L}(Y, f(X))$$

On utilisait donc une fonction de perte particulières, appelée la **perte quadratique**.

Fonction de perte

Tout comme pour les fonctions d'activation, le choix d'une fonction de perte n'est pas prise au hasard. Pour une régression, on choisit souvent une fonction de perte quadratique, mais pour une classification, plusieurs choix s'offrent à nous, et une particulièrement utilisée pour les réseaux de neurones est **l'entropie croisée**.

$$\mathcal{L}(y, f(x)) = -\{y \log f(x) + (1-y) \log(1-f(x))\}$$

Optimisation

Un algorithme d'optimisation que l'on pourrait utiliser ici est la descente de gradient. En effet, si f est différentiable, alors $\mathcal{L}$ l'est aussi et le gradient associé est

$$\nabla_{\mathbf{w}} \mathcal{L}(y, f(x, \mathbf{w})) = -\left\{ y \frac{\nabla_{\mathbf{w}} f(x, \mathbf{w})}{f(x, \mathbf{w})} - (1 - y) \frac{\nabla_{\mathbf{w}} f(x, \mathbf{w})}{1 - f(x, \mathbf{w})} \right\}$$

Optimisation

Exercice

Déterminer un algorithme de gradient pour le perceptron muni d'une fonction d'activation sigmoïde. Pour cela, on injectera directement l'expression analytique de f dans la définition de la perte $\mathcal{L}$ pour ensuite calculer le gradient selon $\mathbf{w}$.

Optimisation

Dans le cas d'une perte entropie croisée avec une fonction sigmoïde, le gradient de la fonction de perte se calcule par

$$\nabla_{\mathbf{w}} \mathcal{L}(y, f(x)) = x^\top \left(1 - y - \frac{1}{1 + \exp(x^\top \mathbf{w})}\right)$$

On peut déterminer un algorithme de descente de gradient avec l'étape de mise à jour suivante :

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i^\top \left(1 - y_i - \frac{1}{1 + \exp(\mathbf{x}_i^\top \mathbf{w}_t)}\right)$$

Selon vous, quel est le problème de cette méthode lorsque le nombre d'individus n est élevé ?

Sommaire

Optimisation offline

En optimisation "classique" (dite *offline*), nous avons accès à tous les gradients pour chaque observation x. On cherche donc à optimiser la fonction f à l'aide de tous les gradients $\nabla f(x)$.

Malheureusement, l'évaluation des gradients peut être très coûteuse, et lorsqu'il y a beaucoup d'observations (n élevé), le coût computationel pour chaque itération ne permet plus d'obtenir une solution approchée en un temps réalise.

Optimisation online

L'optimisation stochastique (dite *online*) n'a pas accès à tous les gradients de chacune des observations. Dans ce contexte, on n'évalue, à chaque itération, qu'un sous-ensemble de gradients parmi les observations : il y a souvent un **tirage aléatoire** pour sélectionner les gradients candidats. On ne cherche plus à optimiser f mais son espérance :

$$\min_{x \in \mathbb{R}} \mathbb{E}[f(x)]$$

En optimisation stochastique, il existe un gradient oracle aléatoire $\tilde{\nabla} f(x)$ dont

$$\mathbb{E}[\tilde{\nabla} f(x)] = \nabla f(x)$$

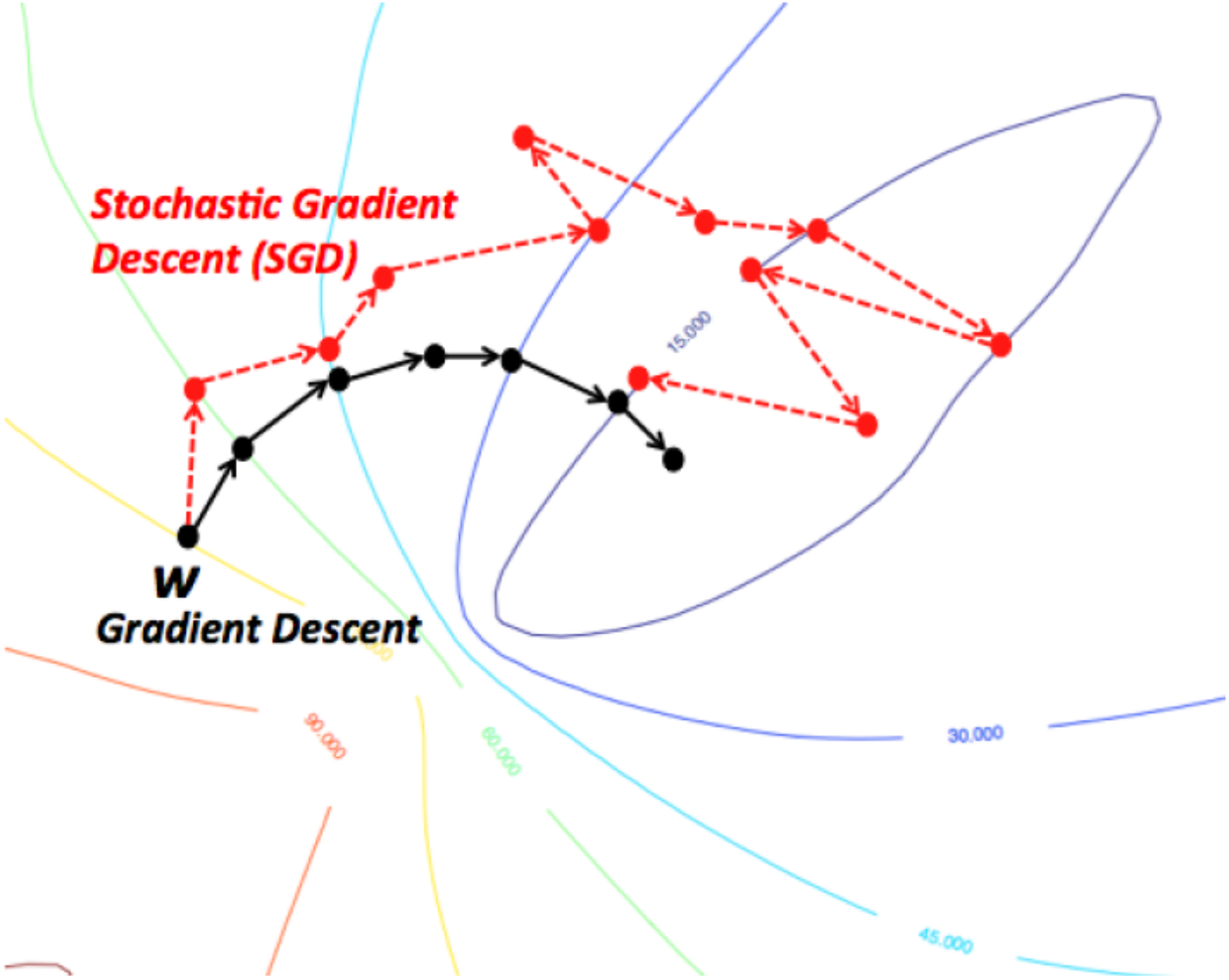
Descente de gradient stochastique (SGD)

Contrairement à la descente de gradient *offline*, l'algorithme de gradient stochastique ne sélectionne **qu'une seule observation** x_t à la t-ième itération. Ainsi, plutôt que de calculer la somme des gradients de chaque observation, l'algorithme SGD n'en calcule qu'un seul.

$$x_{t+1} = x_t - \eta_t \nabla f(x_t)$$

avec η_t le pas de la t-ième itération.

Descente de gradient stochastique (SGD)



Intérêts du gradient stochastique

A priori, la gradient stochastique semble moins efficace que le gradient *offline*, puisqu'il se peut que l'on descende dans la "mauvaise direction" sur certaines itérations. Il y a en réalité plusieurs avantages :

- Bien que plus d'itérations soient nécessaires pour aboutir à une solution convaincante pour le SGD, le temps de calcul d'une seule itération est beaucoup plus rapide que dans le cas *offline*.
- En pratique, on aboutit plus rapidement à la solution optimale avec le SGD que dans le cas *offline* : **les erreurs commises à certaines itérations sont compensées par les temps de calculs beaucoup plus faibles**.

Ceci explique la grande popularité du SGD dans les algorithmes populaires du Machine et Deep Learning.

Descente de gradient stochastique (SGD)

Algorithm 3: Algorithme de descente du gradient stochastique

Input: f différentiable, initialisation x_0
Output: Solution approchée de $\mathbb{E}[\nabla f(x)] = 0$

$x = x_0$
$t = 0$
$g_0 = 2\varepsilon$
while $\|g_t| > \varepsilon$ **do**
    Calcul d'un pas optimal η_t
    Sélection aléatoire d'une observation x_t
    Calcul du gradient $g_t \stackrel{\Delta}{=} \nabla f(x_t)$
    Mise à jour de la solution :

$$x_{t+1} = x_t - \eta_t g_t$$

    $t = t + 1$
return x

Perceptron

Le principe est donc très similaire au gradient *offline*, à la différence où une seule observation est utilisée. Ainsi, pour une observation aléatoire x_t tirée lors de la i-ième itération, l'étape de mise à jour est

$$\mathbf{w}_{t+1} = \mathbf{w}_t - x_t^\top \left(1 - y_t - \frac{1}{1 + \exp(x_t^\top \mathbf{w}_t)}\right)$$

où (x_t, y_t) est la paire tirée aléatoirement de l'ensemble d'entraînement (X, Y).