Perceptron

Introduction à l'apprentissage automatique
Master Sciences Cognitives
Aix Marseille Université

Alexis Nasr

Motivations

- Le **perceptron** est un classifieur linéaire simple qui a joué un rôle important dans l'histoire de l'apprentissage automatique.
- Il permet **d'automatiser** la procédure manuelle que l'on avait adoptée pour trouver un classifieur linéaire.
- Il permet d'illustrer simplement **différentes** manières d'apprendre un classifieur linéaire.
- Il est à la base des **réseaux de neurones** que l'on verra dans les cours suivants.

Objectifs

- Introduire la notion de méthode d'apprentissage **itérative**, qui modifie les paramètres du modèle en fonction de la qualité de ses prédiction sur chaque exemple d'apprentissage.
- Introduire la représentation des paramètres et des features sous la forme de **vecteurs** et effectuer le **produit scalaire** des deux.
- Appliquer l'algorithme de **descente du gradient** à la **fonction d'erreur**.

Plan

Limites du classifieur manuel

- Les deux points A et B permettant de tracer la frontière de décision sont déterminés manuellement à partir de la représentation graphique des données dans le plan.
- Plus viable quand le nombre de features devient important (vecteur $\mathbf{x}$ de dimension n).
- On souhaite que l'ordinateur détermine automatiquement l'équation de la droite séparatrice.

Le perceptron : idée générale

- Méthode itérative :
  - On choisit une droite D au hasard
  - Si D permet une classification parfaite, alors on a fini.
  - Sinon, tant que D fait des erreurs de classification
    - On évalue la qualité de la séparation que D permet de réaliser.
    - On met à jour les paramètres de D pour améliorer la séparation.
- Plusieurs méthodes existent pour réaliser la mise à jour.
- On en verra deux :
  - La règle du perceptron
  - La règle delta

Règle du perceptron

Exemple

i	x_1	x_2	y	i	x_1	x_2	y
1	4.79	0.02	-1	11	2.78	1.90	$+1$
2	4.98	0.10	-1	12	2.51	2.03	$+1$
3	5.24	0.12	-1	13	2.63	1.60	$+1$
4	5.73	0.00	-1	14	2.75	2.08	$+1$
5	4.72	0.02	-1	15	2.54	1.92	$+1$
6	5.39	0.00	-1	16	2.65	1.55	$+1$
7	5.19	0.08	-1	17	2.35	1.75	$+1$
8	5.17	0.00	-1	18	2.70	1.77	$+1$
9	5.06	0.02	-1	19	2.66	1.73	$+1$
10	5.40	0.02	-1	20	2.35	1.47	$+1$

- Fréquences des lettres u (x_1) et w (x_2) dans des document en anglais et en français.
- $y = +1$ pour les documents en anglais.
- $y = -1$ pour les documents en français.

Le perceptron : idée générale

- Modèle linéaire :

$$w_1 x_1 + w_2 x_2$$

- On souhaite déterminer les valeurs de w_1 et w_2 de sorte que :

$$\forall i, signe(w_1 x_{1,i} + w_2 x_{2,i}) = y_i$$

- Rappel

$$signe(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ -1 & \text{sinon} \end{cases}$$

- Notations :
  $x_{1,i}$ est la première composante (x_1) de l'exemple i.

Algorithme du perceptron

- On donne une valeur quelconque à w_1 et w_2.
- On considère les 20 exemples les uns après les autres.
- Pour chaque exemple i, on calcule :

$$\hat{y}_i = signe(w_1 x_{1,i} + w_2 x_{2,i})$$

- Trois cas peuvent se présenter :
  1. $\hat{y}_i = y_i$: l'exemple i est **bien classé**, on ne **modifie pas** w_1 et w_2
  2. $\hat{y}_i = 1$ et $y_i = -1$: on **diminue** les valeurs de w_1 et w_2, de sorte à **diminuer** la valeur de l'expression $w_1 x_{1,i} + w_2 x_{2,i}$
  3. $\hat{y}_i = -1$ et $y_i = 1$: on **augmente** les valeurs de w_1 et w_2, de sorte à **augmenter** la valeur de l'expression $w_1 x_{1,i} + w_2 x_{2,i}$

Règle de mise à jour des poids

- Pour chaque exemple i, la règle de mise à jour du perceptron s'écrit de la manière suivante :

$$w_1 \leftarrow w_1 + \eta(y_i - \hat{y}_i) \times x_{1,i}$$
$$w_2 \leftarrow w_2 + \eta(y_i - \hat{y}_i) \times x_{2,i}$$

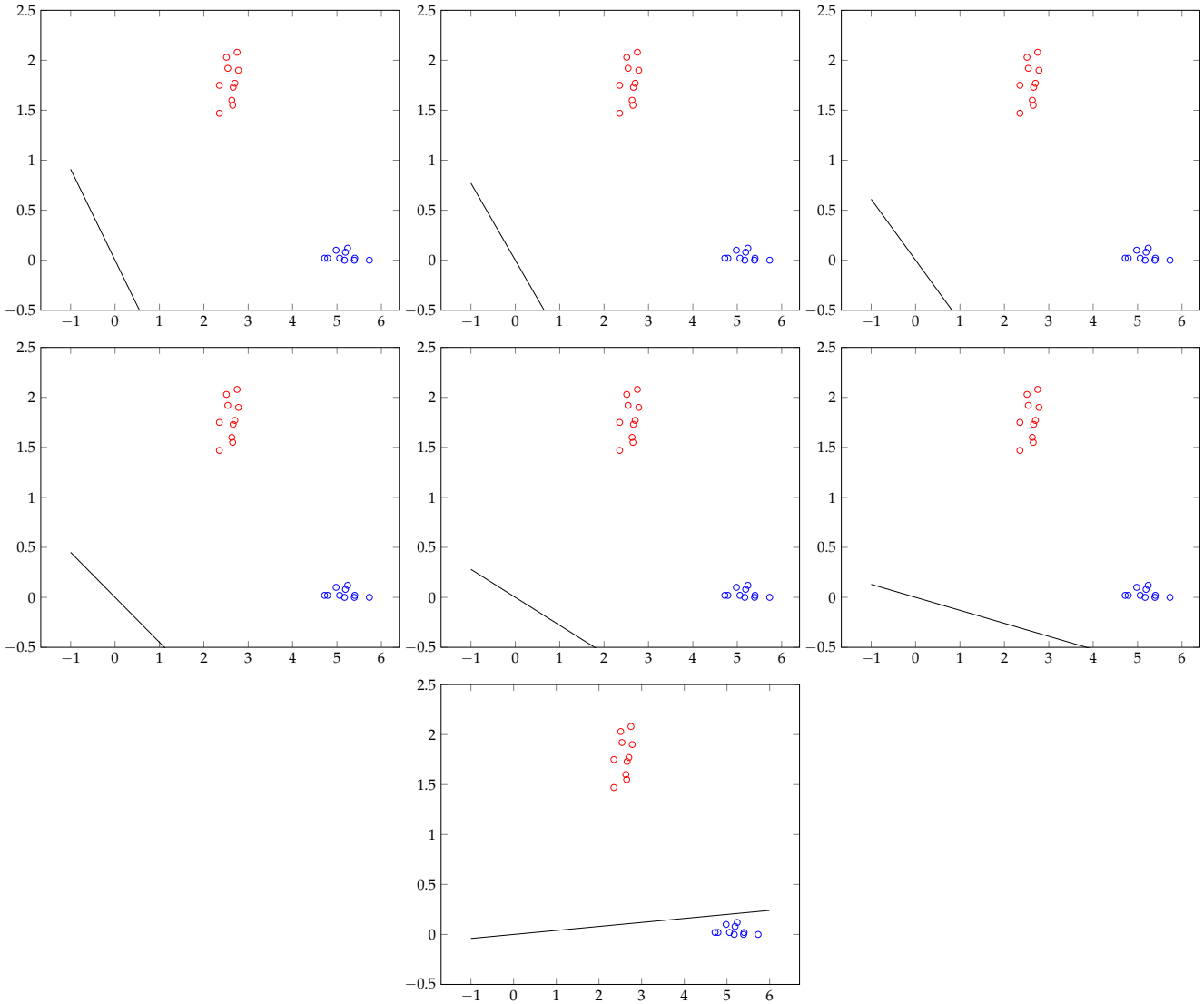
- η est appelé **pas d'apprentissage** (learning rate), il permet de régler l'amplitude de la mise à jour.
- Trois cas :

$\hat{y}_i = y_i$	$\hat{y}_i = 1$ et $y_i = -1$	$\hat{y}_i = -1$ et $y_i = 1$
$w_1 \leftarrow w_1$	$w_1 \leftarrow w_1 - 2 \times \eta \times x_{1,i}$	$w_1 \leftarrow w_1 + 2 \times \eta \times x_{1,i}$
$w_2 \leftarrow w_2$	$w_2 \leftarrow w_2 - 2 \times \eta \times x_{2,i}$	$w_2 \leftarrow w_2 + 2 \times \eta \times x_{2,i}$

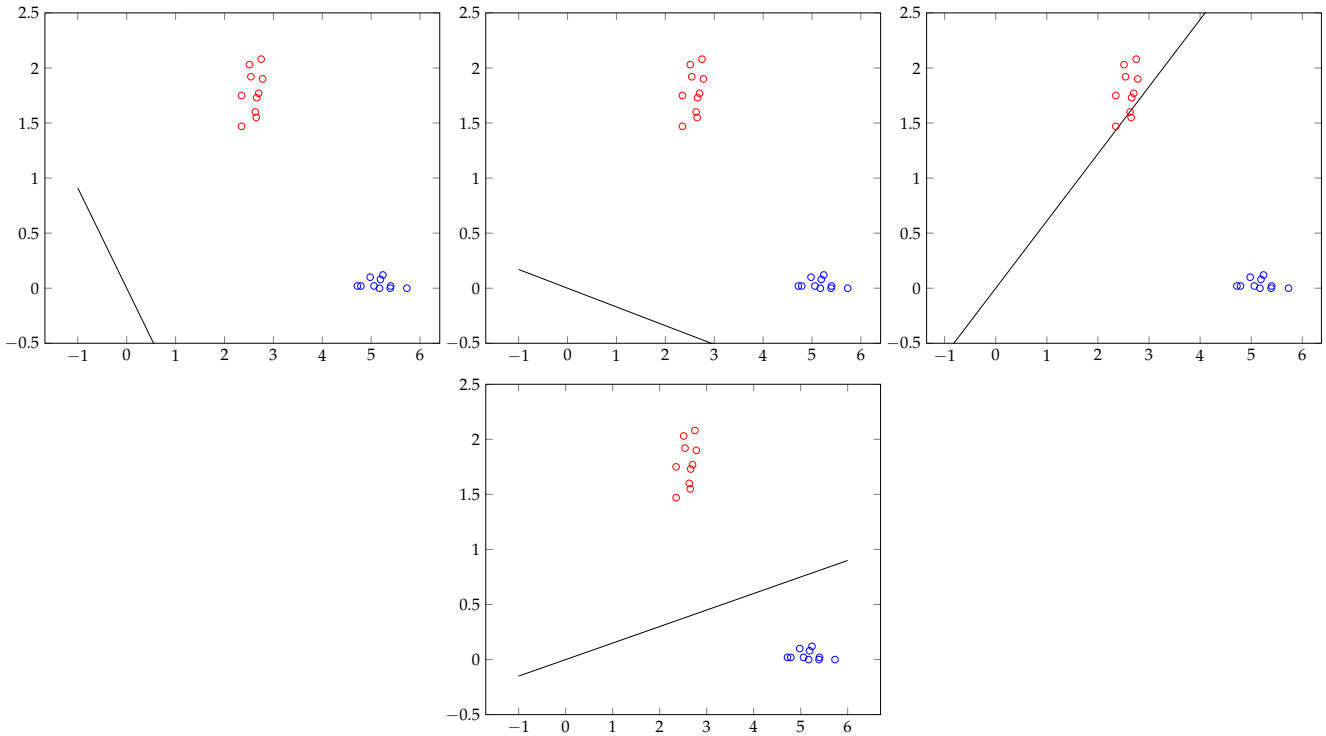
Exemple $\eta = 0.1$

i	y_i	$\hat{y}_i$	$x_{1,i}$	$x_{2,i}$	w_1	w_2	$-\frac{w_1}{w_2}$
0					5.92	6.48	-0.91
1	-1	1	4.79	0.02	4.96	6.48	-0.77
2	1	1	2.78	1.9	4.96	6.48	-0.77
3	-1	1	4.98	0.1	3.97	6.46	-0.61
4	1	1	2.51	2.03	3.97	6.46	-0.61
5	-1	1	5.24	0.12	2.92	6.43	-0.45
6	1	1	2.63	1.6	2.92	6.43	-0.45
7	-1	1	5.73	0.0	1.77	6.43	-0.28
8	1	1	2.75	2.08	1.77	6.43	-0.28
9	-1	1	4.72	0.02	0.83	6.43	-0.13
10	1	1	2.54	1.92	0.83	6.43	-0.13
11	-1	1	5.39	0.0	-0.25	6.43	0.04

Exemple $\eta = 0.1$



Exemple $\eta = 0.5$



Introduction d'un biais

- Les droites de séparations susceptibles d'être trouvées par l'algorithme du perceptron sont de la forme : $y = -\frac{w_1}{w_2}x$
- Ces droites passent toutes par le **point origine** $(0, 0)$.
- Si les données ne sont pas séparables par une droite passant par l'origine, l'algorithme ne pourra pas déterminer une droite de séparation des données.
- Pour remédier à ce problème, on ajoute le paramètre, w_0, qui a la particularité de ne pas être associé à une variable d'entrée.
- Le modèle devient :

$$signe(w_0 + w_1 x_1 + w_2 x_2)$$

- Le paramètre w_0 est mis à jour par le peceptron presque de la même manière que les autres paramètres w_1 et w_2.

$$w_0 \leftarrow w_0 + \eta(y_i - \hat{y}_i) \times 1$$

Généralisation à n dimensions

- Le perceptron tel que présenté jusque là possédait une limite importante, celle de ne prendre en entrée que deux features, dans notre cas la fréquence des lettres u et w.
- On avait imposé cette contrainte afin de pouvoir visualiser les données dans le plan et la surface de décision sous la forme d'une droite.
- L'algorithme du perceptron peut être **généralisé** à un nombre quelconque de features.

Généralisation à n dimensions

- Pour $n = 2$, les exemples à classifier peuvent être représentés comme des points dans le **plan** et la surface de séparation est une **droite**, c'est ce que nous avons fait jusque là.
- Pour $n = 3$ les exemples sont des points dans l'espace **tri-dimensionnel** et la surface de séparation est un **plan**.
- A partir de $n = 4$, il n'existe pas de moyen simple de visualiser les données ni les surfaces de séparation.

Représentation vectorielle

- Il est commode d'organiser les features sous la forme d'un vecteur de dimension n, noté $\mathbf{x}$:

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$$

- Les poids sont eux aussi organisés sous la forme d'un vecteur de même dimension, noté $\mathbf{w}$:

$$\mathbf{w} = (w_1, w_2, \ldots, w_n)^T$$

- La multiplication des poids par les features est vu comme un **produit scalaire** :

$$\mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{n} w_i x_i$$

Prise en compte du biais

- on peut ajouter une dimension à chacun des deux vecteurs **x** et **w** pour prendre en compte le biais :

$$\mathbf{x} = (1, x_1, x_2, \ldots, x_n)^T$$

$$\mathbf{w} = (w_0, w_1, w_2, \ldots, w_n)^T$$

- Le produit scalaire permet bien de réaliser le calcul souhaité :

$$\mathbf{w} \cdot \mathbf{x} = \sum_{i=0}^{n} w_i x_i = w_0 + \sum_{i=1}^{n} w_i x_i$$

- Le perceptron s'écrit :

$$signe(\mathbf{w} \cdot \mathbf{x})$$

Implémentation du perceptron en python

```
def train(lx, ly, eta, maxIteration):
    dim = len(lx[0])
    w = [random.uniform(1,10)] * dim
    iteration = 0
    exemplesMalClasses = 1
    while exemplesMalClasses != 0 and iteration < maxIteration :
        exemplesMalClasses = 0
        for i in range(len(lx)):
            x = lx[i]
            reference = ly[i]
            prediction = signe(produitScalaire(x,w))
            if prediction != reference :
                exemplesMalClasses += 1
                if reference > 0 :
                  addVec(w, multVec(x, eta))
                else:
                  subVec(w, multVec(x, eta))
   return w
```

Convergence

- On peut prouver que cette méthode **converge** en un nombre fini d'étapes pour une valeur faible de η et si les données sont **linéairement séparables**.
- Si les données ne sont pas linéairement séparables, la convergence n'est **pas assurée**.

Règle Delta

Règle delta

- La règle de mise à jour du perceptron permet de trouver une solution au problème de classification lorsque les données sont **linéairement séparable**.
- En revanche, il peut ne **pas converger** lorsque les données ne sont pas linéairement séparables.
- Autre manière d'estimer les coefficients du modèle : la **règle delta**, qui permet de converger vers la meilleure solution possible lorsque les données ne sont pas linéairement séparables.
- Cette méthode repose sur une idée générale : la **minimisation de fonction d'erreur**

Illustration sur le perceptron sans seuil

- Rappel : le perceptron est composé de deux étapes
  1. calcul du produit scalaire $\mathbf{w} \cdot \mathbf{x}$
  2. le produit scalaire est fourni en entrée à la fonction $signe()$, pour donner :

$$\hat{y} = signe(\mathbf{w} \cdot \mathbf{x})$$

- Le **perceptron sans seuil** correspond à la première étape :

$$\hat{y} = \mathbf{w} \cdot \mathbf{x}$$

- On verra à la fin du cours la raison pour laquelle la fonction $signe()$ n'a pas été prise en compte.

Fonction d'erreur

- On utilise l'erreur quadratique :

$$E(\mathbf{w}; \mathcal{D}) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

- Pour chaque valeur possible de $\mathbf{w}$, correspond une valeur de $E(\mathbf{w}; \mathcal{D})$.
- On souhaite trouver la valeur $\mathbf{w}^*$ qui minimise la fonction $E(\mathbf{w}; \mathcal{D})$:

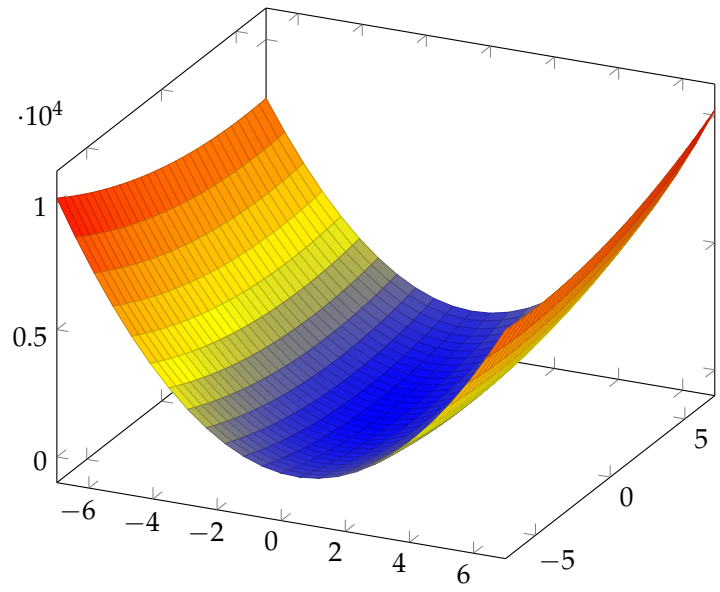
$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^2} E(\mathbf{w}; \mathcal{D})$$

Visualisation de la fonction d'erreur

- Il est éclairant de tracer la fonction d'erreur dans le cas simple où le vecteur d'entrée **x** est de dimension 2.
- Dans ce cas, le perceptron prend une forme particulièrement simple : $\hat{y} = w_1 x_1 + w_2 x_2$
- La fonction d'erreur s'écrit alors :

$$\begin{aligned} E(w_1, w_2; \mathcal{D}) &= \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \\ &= \frac{1}{2} \sum_{i=1}^{N} (w_1 x_{1,i} + w_2 x_{2,i} - y_i)^2 \end{aligned}$$

Visualisation de la fonction d'erreur I



Fonction d'erreur pour l'exemple des fréquences des lettres u et w en anglais et en français :

$$E(w_1, w_2; \mathcal{D}) = 167.59w_1^2 + 16.04w_2^2 + 24.08w_1w_2 + 12w_1 - 8.7w_2 + 10$$

Descente du gradient I

- Si l'on est capable de calculer le gradient de la fonction $E(\mathbf{w}; \mathcal{D})$, alors, on peut trouver $\mathbf{w}^*$ en effectuant la **descente du gradient**.
- La règle de **mise à jour** est la suivante :

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w})$$

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

- Pour la mettre en œuvre, il faut calculer les **dérivées partielles** $\frac{\partial E}{\partial w_i}$

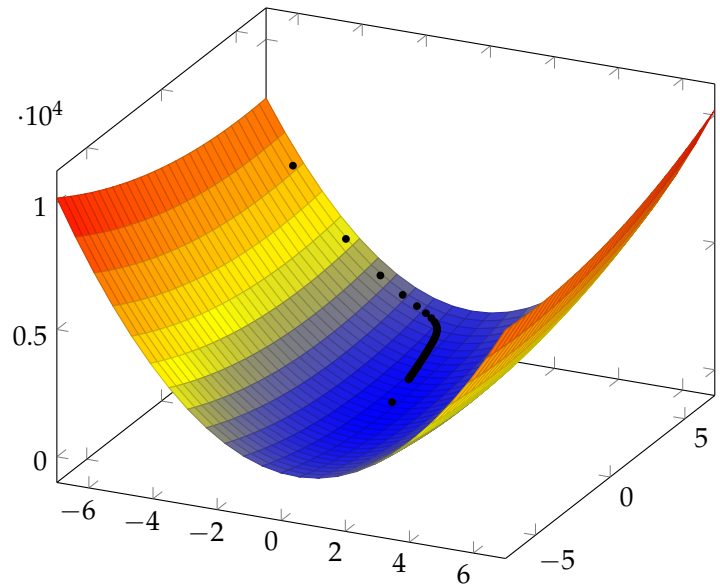
Calcul des dérivées partielles

$$
\begin{aligned}
\frac{\partial E}{\partial w_1} &= \frac{\partial}{\partial w_1}\frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{N}\frac{\partial}{\partial w_1}(\hat{y}_i - y_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{N}2(\hat{y}_i - y_i)\frac{\partial}{\partial w_1}(\hat{y}_i - y_i) \\
&= \sum_{i=1}^{N}(\hat{y}_i - y_i)\frac{\partial}{\partial w_1}(w_1 x_{1,i} + w_2 x_{2,i} - y_i) \\
&= \sum_{i=1}^{N}(\hat{y}_i - y_i)x_{1,i}
\end{aligned}
$$

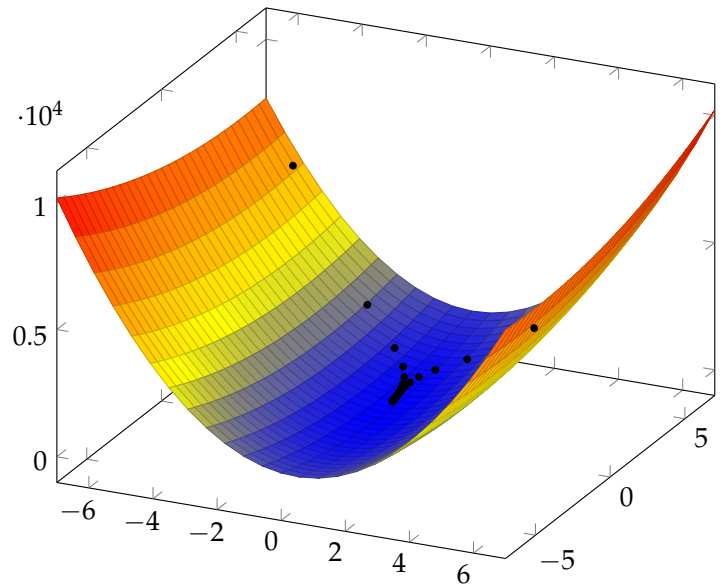
On a donc :

$$
\nabla E(w_1, w_2; \mathcal{D}) = (\sum_{i=1}^{N}(\hat{y}_i - y_i)x_{1,i}, \sum_{i=1}^{N}(\hat{y}_i - y_i)x_{2,i})^T
$$

Descente du gradient de la fonction d'erreur $\eta = 0.001$



Descente du gradient de la fonction d'erreur $\eta = 0.005$



Dérivabilité et convexité de la fonction d'erreur

- Pour calculer le gradient de la fonction d'erreur $E(\mathbf{w}; \mathcal{D})$, il faut qu'elle soit soit **dérivable**.
- C'est la raison pour laquelle nous avons utilisé un perceptron sans seuil dans notre exemple, la fonction $signe()$ n'étant pas dérivable.
- Dans le cas d'un classifieur linéaire et lorsque la fonction d'erreur est la somme des carrés des différences, on aboutit à une fonction **convexe** et la descente du gradient permet de trouver une solution optimale.
- Cela n'est pas garanti pour d'autres fonctions d'erreurs.
- La solution $\mathbf{w}^*$ peut être un **minimum local**.

Sources

- Tom Mitchell, *Machine Learning*, McGraw Hill, 1997.