Jérémie Cabessa Laboratoire DAVID, UVSQ

Soit $f: \mathbb{R} \longrightarrow \mathbb{R}$ une fonction. On dit que f est différentiable au point $x_0 \in \mathbb{R}$ si la limite suivante existe:

$$\frac{df}{dx}(x_0) := \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Supposons f différentiable en tous points. Alors la limite

$$f'(x_0) := \frac{df}{dx}(x_0)$$

est la dérivée de f en x_0 .

La fonction

$$f': \mathbb{R} \longrightarrow \mathbb{R}$$
$$x \longmapsto f'(x)$$

est la dérivée de f.



▶ Soit $f: \mathbb{R} \longrightarrow \mathbb{R}$ une fonction. On dit que f est différentiable au point $x_0 \in \mathbb{R}$ si la limite suivante existe:

$$\frac{df}{dx}(x_0) := \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Supposons f différentiable en tous points. Alors la limite

$$f'(x_0) := \frac{df}{dx}(x_0)$$

est la dérivée de f en x_0 .

La fonction

$$f': \mathbb{R} \longrightarrow \mathbb{R}$$
 $x \longmapsto f'(x)$

est la *dérivée* de f.



Soit $f: \mathbb{R} \longrightarrow \mathbb{R}$ une fonction. On dit que f est différentiable au point $x_0 \in \mathbb{R}$ si la limite suivante existe:

$$\frac{df}{dx}(x_0) := \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Supposons f différentiable en tous points. Alors la limite

$$f'(x_0) := \frac{df}{dx}(x_0)$$

est la dérivée de f en x_0 .

La fonction

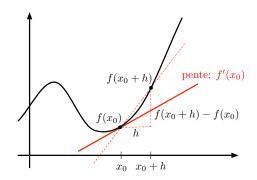
$$f': \mathbb{R} \longrightarrow \mathbb{R}$$

 $x \longmapsto f'(x)$

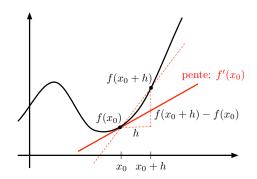
est la *dérivée* de f.



- $f'(x_0)$ est la pente de la tangeante au graphe de f au point $(x_0, f(x_0))$.
- $ightharpoonup f'(x_0)$ est le taux d'accroissement de f en x_0 .



- $f'(x_0)$ est la pente de la tangeante au graphe de f au point $(x_0, f(x_0))$.
- ▶ $f'(x_0)$ est le taux d'accroissement de f en x_0 .



Soient $f: \mathbb{R}^N \longrightarrow \mathbb{R}$ une fonction différentiable (définition un peu différente). Le gradient de f au point $\boldsymbol{x} = (x_1, \dots, x_N)$ est le vecteur défini par

$$abla f(oldsymbol{x}) := egin{pmatrix} rac{\partial f}{\partial x_1}(oldsymbol{x}) \\ dots \\ rac{\partial f}{\partial x_N}(oldsymbol{x}) \end{pmatrix}$$

La fonction

$$\nabla f: \mathbb{R}^N \longrightarrow \mathbb{R}^N$$
$$\boldsymbol{x} \longmapsto \nabla f(\boldsymbol{x})$$

est le *gradient* de *f*

Soient $f: \mathbb{R}^N \longrightarrow \mathbb{R}$ une fonction différentiable (définition un peu différente). Le gradient de f au point $\boldsymbol{x} = (x_1, \dots, x_N)$ est le vecteur défini par

$$\nabla f(\boldsymbol{x}) := \begin{pmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{x}) \\ \vdots \\ \frac{\partial f}{\partial x_N}(\boldsymbol{x}) \end{pmatrix}$$

► La fonction

$$abla f: \mathbb{R}^N \longrightarrow \mathbb{R}^N \\
oldsymbol{x} \longmapsto
abla f(oldsymbol{x})$$

est le gradient de f.

- ightharpoonup
 abla f(x) est un vecteur de dimension N ("dans le sol").
- Chaque dimension $\frac{\partial f}{\partial x_i}(x)$ de $\nabla f(x)$ représente le taux d'accroissement de f en x dans la direction e_i .

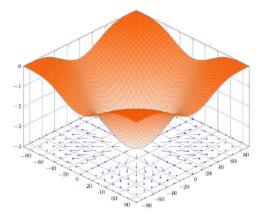
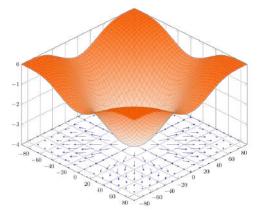


Figure taken from [Wikipedia contributors, 2022a]

- ightharpoonup
 abla f(x) est un vecteur de dimension N ("dans le sol").
- ▶ Chaque dimension $\frac{\partial f}{\partial x_i}(x)$ de $\nabla f(x)$ représente le taux d'accroissement de f en x dans la direction e_i .



- ightharpoonup
 abla f(x) représente la direction de la plus grande pente ascendante de f au point x.
- ightharpoonup
 abla f(x) is the direction of steepest ascent at x.
- De manière équivalente, $-\nabla f(x)$ représente la direction de la plus grande pente descendante de f au point x.
- $ightharpoonup -\nabla f(x)$ is the direction of steepest descent at x

- ightharpoonup
 abla f(x) représente la direction de la plus grande pente ascendante de f au point x.
- ▶ $\nabla f(x)$ is the direction of steepest ascent at x.
- De manière équivalente, $-\nabla f(x)$ représente la direction de la plus grande pente descendante de f au point x.
- $ightharpoonup -\nabla f(x)$ is the direction of steepest descent at x

- ightharpoonup
 abla f(x) représente la direction de la plus grande pente ascendante de f au point x.
- ightharpoonup
 abla f(x) is the direction of steepest ascent at x.
- ▶ De manière équivalente, $-\nabla f(x)$ représente la direction de la plus grande pente descendante de f au point x.
- $ightharpoonup -\nabla f(x)$ is the direction of steepest descent at x

- ightharpoonup
 abla f(x) représente la direction de la plus grande pente ascendante de f au point x.
- ightharpoonup
 abla f(x) is the direction of steepest ascent at x.
- ▶ De manière équivalente, $-\nabla f(x)$ représente la direction de la plus grande pente descendante de f au point x.
- ▶ $-\nabla f(x)$ is the direction of steepest descent at x.

La dérivée directionnelle de f au point $\boldsymbol{x}=(x_1,\dots,x_N)$ selon la direction $\boldsymbol{v}=(v_1,\dots,v_N)$ est défini par

$$\nabla_{\boldsymbol{v}} f(\boldsymbol{x}) := \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{v}) - f(\boldsymbol{x})}{h \|\boldsymbol{v}\|}.$$

Intuitivement, $\nabla_{\boldsymbol{v}} f(\boldsymbol{x})$ représente le taux d'accroissement de f en \boldsymbol{x} dans la direction \boldsymbol{v} .

▶ Dire que le gradient est la direction de la plus grande pente ascendante signifie formellement que:

le gradient est le vecteur qui maximise la dérivée directionnelle.

THEOREM

Soit $f: \mathbb{R}^N \longrightarrow \mathbb{R}$ une fonction différentiable. Pour tout $x \in \mathbb{R}^N$, on a:

$$\nabla f(\boldsymbol{x}) \in \arg\max_{\boldsymbol{v} \in \mathbb{R}^N} \nabla_{\boldsymbol{v}} f(\boldsymbol{x}).$$

Preuve: On peut montrer que

$$abla_{oldsymbol{v}}f(oldsymbol{x}) =
abla f(oldsymbol{x}) \cdot rac{oldsymbol{v}}{\|oldsymbol{v}\|} = \|
abla f(oldsymbol{x})\| \cdot rac{\|oldsymbol{v}\|}{\|oldsymbol{v}\|} \cdot \cos heta = \|
abla f(oldsymbol{x})\| \cdot \cos(heta)$$

où
$$\theta := \theta \left(\nabla f(\boldsymbol{x}), \boldsymbol{v} \right).$$

Ainsi, $\nabla_{\boldsymbol{v}} f(\boldsymbol{x})$ est maximal lorsque $\cos(\theta) = 1$, i.e., lorsque \boldsymbol{v} est parallèle à $\nabla f(\boldsymbol{x})$. Donc en particulier,

$$\nabla f(x) \in \arg\max_{v \in \mathbb{R}^N} \nabla_v f(x)$$

Preuve: On peut montrer que

$$\nabla_{\boldsymbol{v}} f(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) \cdot \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} = \|\nabla f(\boldsymbol{x})\| \cdot \frac{\|\boldsymbol{v}\|}{\|\boldsymbol{v}\|} \cdot \cos \theta = \|\nabla f(\boldsymbol{x})\| \cdot \cos(\theta)$$

où
$$\theta := \theta \left(\nabla f(\boldsymbol{x}), \boldsymbol{v} \right)$$
.

Ainsi, $\nabla_{\boldsymbol{v}} f(\boldsymbol{x})$ est maximal lorsque $\cos(\theta) = 1$, i.e., lorsque \boldsymbol{v} est parallèle à $\nabla f(\boldsymbol{x})$. Donc en particulier,

$$\nabla f(\boldsymbol{x}) \in \arg\max_{\boldsymbol{v} \in \mathbb{R}^N} \nabla_{\boldsymbol{v}} f(\boldsymbol{x})$$

Preuve: On peut montrer que

$$\nabla_{\boldsymbol{v}} f(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) \cdot \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} = \|\nabla f(\boldsymbol{x})\| \cdot \frac{\|\boldsymbol{v}\|}{\|\boldsymbol{v}\|} \cdot \cos \theta = \|\nabla f(\boldsymbol{x})\| \cdot \cos(\theta)$$

où
$$\theta := \theta \left(\nabla f(\boldsymbol{x}), \boldsymbol{v} \right)$$
.

Ainsi, $\nabla_{\boldsymbol{v}} f(\boldsymbol{x})$ est maximal lorsque $\cos(\theta) = 1$, i.e., lorsque \boldsymbol{v} est parallèle à $\nabla f(\boldsymbol{x})$. Donc en particulier,

$$\nabla f(\boldsymbol{x}) \in \arg\max_{\boldsymbol{v} \in \mathbb{R}^N} \nabla_{\boldsymbol{v}} f(\boldsymbol{x})$$

Preuve: On peut montrer que

$$\nabla_{\boldsymbol{v}} f(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) \cdot \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} = \|\nabla f(\boldsymbol{x})\| \cdot \frac{\|\boldsymbol{v}\|}{\|\boldsymbol{v}\|} \cdot \cos \theta = \|\nabla f(\boldsymbol{x})\| \cdot \cos(\theta)$$

où
$$\theta := \theta \left(\nabla f(\boldsymbol{x}), \boldsymbol{v} \right)$$
.

Ainsi, $\nabla_{\pmb{v}} f(\pmb{x})$ est maximal lorsque $\cos(\theta)=1$, i.e., lorsque \pmb{v} est parallèle à $\nabla f(\pmb{x})$. Donc en particulier,

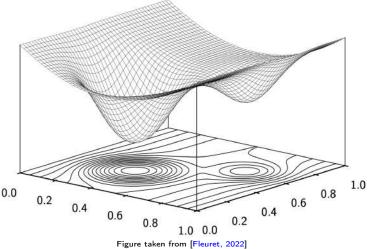
$$\nabla f(\boldsymbol{x}) \in \arg\max_{\boldsymbol{v} \in \mathbb{R}^N} \nabla_{\boldsymbol{v}} f(\boldsymbol{x}).$$

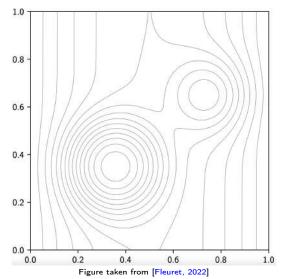
- ▶ Soit $f: \mathbb{R}^N \longrightarrow \mathbb{R}$ une fonction différentiable.
- ightharpoonup On cherche à minimiser f: minimum local, global.
- ▶ Descente de gradient: on part d'un point au hasard; puis pas à pas, on descend le long de la surface en direction de la plus grande pente $-\nabla f(x)$.
- Remarque: on ne descend pas vraiment le long de la surface, on bouge dans le sol...

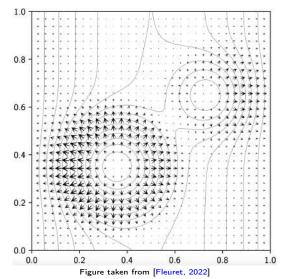
- ▶ Soit $f : \mathbb{R}^N \longrightarrow \mathbb{R}$ une fonction différentiable.
- ightharpoonup On cherche à minimiser f: minimum local, global.
- ▶ Descente de gradient: on part d'un point au hasard; puis pas à pas, on descend le long de la surface en direction de la plus grande pente $-\nabla f(x)$.
- Remarque: on ne descend pas vraiment le long de la surface, on bouge dans le sol...

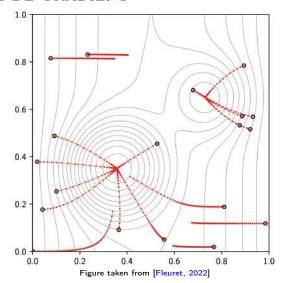
- ▶ Soit $f : \mathbb{R}^N \longrightarrow \mathbb{R}$ une fonction différentiable.
- \triangleright On cherche à minimiser f: minimum local, global.
- ▶ Descente de gradient: on part d'un point au hasard; puis pas à pas, on descend le long de la surface en direction de la plus grande pente $-\nabla f(x)$.
- ▶ Remarque: on ne descend pas vraiment le long de la surface, on bouge dans le sol...

- ▶ Soit $f : \mathbb{R}^N \longrightarrow \mathbb{R}$ une fonction différentiable.
- \triangleright On cherche à minimiser f: minimum local, global.
- ▶ Descente de gradient: on part d'un point au hasard; puis pas à pas, on descend le long de la surface en direction de la plus grande pente $-\nabla f(x)$.
- Remarque: on ne descend pas vraiment le long de la surface, on bouge dans le sol...









Algorithm 1: Gradient descent

Inputs: differentiable function $f: \mathbb{R}^N \longrightarrow \mathbb{R}$; initial point $\boldsymbol{x} \in \mathbb{R}^N$; learning rate $\lambda > 0$; tolerence $\epsilon > 0$.

```
while \|\nabla f(x)\| > \epsilon do \|x := x - \lambda \nabla f(x)\| end
```

L'algorithme donne lieu à une suite de points x_0, x_1, x_2 qui, espérons, converge vers un minimum local ou global.

Algorithm 1: Gradient descent

Inputs: differentiable function $f: \mathbb{R}^N \longrightarrow \mathbb{R}$; initial point $x \in \mathbb{R}^N$; learning rate $\lambda > 0$; tolerence $\epsilon > 0$.

end

return x

Algorithm 1: Gradient descent

Inputs: differentiable function $f: \mathbb{R}^N \longrightarrow \mathbb{R}$; initial point $\mathbf{x} \in \mathbb{R}^N$; learning rate $\lambda > 0$; tolerence $\epsilon > 0$.

return x

L'algorithme donne lieu à une suite de points x_0, x_1, x_2, \ldots qui, espérons, converge vers un minimum local ou global.

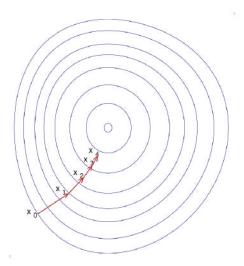


Figure taken from [Wikipedia contributors, 2022b]



- lacksquare Soit $S=\left\{(oldsymbol{x_i},oldsymbol{y_i})\in\mathbb{R}^{d_1} imes\mathbb{R}^{d_2}:i=1,\ldots,N
 ight\}$ un dataset.
- ▶ Soit $\hat{f}(\cdot; \Theta)$ un modèle qui dépend des paramètres Θ :

$$\hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2}$$
 $\boldsymbol{x}_i \longmapsto \hat{\boldsymbol{y}}_i := \hat{f}(\boldsymbol{x}_i; \boldsymbol{\Theta})$

- lacksquare Soit $S=\left\{(oldsymbol{x_i},oldsymbol{y_i})\in\mathbb{R}^{d_1} imes\mathbb{R}^{d_2}:i=1,\ldots,N
 ight\}$ un dataset.
- ▶ Soit $\hat{f}(\cdot; \Theta)$ un modèle qui dépend des paramètres Θ :

$$egin{array}{lll} \hat{f}(\cdot;oldsymbol{\Theta}):\mathbb{R}^{d_1} &\longrightarrow& \mathbb{R}^{d_2} \ oldsymbol{x_i} &\longmapsto& \hat{oldsymbol{y}_i}:=\hat{f}(oldsymbol{x_i};oldsymbol{\Theta}) \end{array}$$

Soit une fonction de coût (cost or loss function) qui mesure l'erreur entre la prédiction \hat{y}_i et la réalité y_i :

$$egin{array}{lll} \ell: \mathbb{R}^{d_2} imes \mathbb{R}^{d_2} & \longrightarrow & \mathbb{R} \ (\hat{m{y}}_{m{i}}, m{y}_{m{i}}) & \longmapsto & \ell\left(\hat{m{y}}_{m{i}}, m{y}_{m{i}}
ight) \end{array}$$

La fonction de coût peut être généralisée à un ensemble de prédictions et de réalités (e.g., moyenne des ℓ individuelles):

$$egin{array}{lll} \mathcal{L}: \mathbb{R}^{d_2} imes \cdots & \mathbb{R}^{d_2} & \longrightarrow & \mathbb{R} \ (\hat{m{y}}_1, \ldots, \hat{m{y}}_N, m{y}_i \ldots, m{y}_N) & \longmapsto & \mathcal{L}\left(\hat{m{y}}_1, \ldots, \hat{m{y}}_N, m{y}_i \ldots, m{y}_N
ight) \end{array}$$

Soit une fonction de coût (cost or loss function) qui mesure l'erreur entre la prédiction \hat{y}_i et la réalité y_i :

$$egin{array}{lll} \ell: \mathbb{R}^{d_2} imes \mathbb{R}^{d_2} & \longrightarrow & \mathbb{R} \ (\hat{m{y}}_{m{i}}, m{y}_{m{i}}) & \longmapsto & \ell\left(\hat{m{y}}_{m{i}}, m{y}_{m{i}}
ight) \end{array}$$

La fonction de coût peut être généralisée à un ensemble de prédictions et de réalités (e.g., moyenne des ℓ individuelles):

$$egin{array}{lll} \mathcal{L}: \mathbb{R}^{d_2} imes \cdots imes \mathbb{R}^{d_2} & \longrightarrow & \mathbb{R} \ (\hat{m{y_1}}, \ldots, \hat{m{y_N}}, m{y_i} \ldots, m{y_N}) & \longmapsto & \mathcal{L}\left(\hat{m{y_1}}, \ldots, \hat{m{y_N}}, m{y_i} \ldots, m{y_N}
ight) \end{array}$$

- Pour différents paramètres Θ , on aura différentes prédictions $\hat{y}_i = \hat{f}(x_i; \Theta)$, et donc différentes erreurs $\ell(\dots)$ et $\mathcal{L}(\dots)$.
- ▶ Ainsi, ℓ et \mathcal{L} sont aussi des fonctions des paramètres Θ :

$$egin{array}{lll} \ell: \mathbb{R}^{|\Theta|} & \longrightarrow & \mathbb{R} \ & oldsymbol{\Theta} & \longmapsto & \ellig(\hat{oldsymbol{y}}_i, oldsymbol{y}_i; oldsymbol{\Theta}ig) \ & \mathcal{L}: \mathbb{R}^{|\Theta|} & \longrightarrow & \mathbb{R} \ & oldsymbol{\Theta} & \longmapsto & \mathcal{L}ig(\hat{oldsymbol{y}}_1, \ldots, \hat{oldsymbol{y}}_N, oldsymbol{y}_1, \ldots, oldsymbol{y}_N; oldsymbol{\Theta}ig) \end{array}$$

où $|\Theta|$ est le nombre de paramètres Θ .

- Pour différents paramètres Θ , on aura différentes prédictions $\hat{y}_i = \hat{f}(x_i; \Theta)$, et donc différentes erreurs $\ell(\dots)$ et $\mathcal{L}(\dots)$.
- ▶ Ainsi, ℓ et \mathcal{L} sont aussi des fonctions des paramètres Θ :

$$egin{array}{lll} \ell: \mathbb{R}^{|oldsymbol{\Theta}|} & \longrightarrow & \mathbb{R} \ oldsymbol{\Theta} & \longmapsto & \ellig(\hat{y}_i, y_i; oldsymbol{\Theta}ig) \ \mathcal{L}: \mathbb{R}^{|oldsymbol{\Theta}|} & \longrightarrow & \mathbb{R} \ oldsymbol{\Theta} & \longmapsto & \mathcal{L}(\hat{y}_1, \ldots, \hat{y}_N, y_1, \ldots, y_N; oldsymbol{\Theta}ig) \end{array}$$

où $|\Theta|$ est le nombre de paramètres $\Theta.$

▶ L'entraı̂nement du modèle $\hat{f}(\cdots;\Theta)$ consiste à déterminer des paramètres Θ qui minimisent les fonction de coût

$$\ell(\ldots; \boldsymbol{\Theta})$$
 ou $\mathcal{L}(\ldots; \boldsymbol{\Theta})$.

- Pour minimiser la fonction de coût, on utilise des descentes de gradient:
 - gradient descent
- stochastic gradient descent
- mini-batch stochastic gradient descent

 $lackbox{L'entraînement}$ du modèle $\hat{f}(\cdots;\Theta)$ consiste à déterminer des paramètres Θ qui minimisent les fonction de coût

$$\ell(\ldots; \boldsymbol{\Theta})$$
 ou $\mathcal{L}(\ldots; \boldsymbol{\Theta})$.

- ▶ Pour minimiser la fonction de coût, on utilise des descentes de gradient:
 - gradient descent
 - stochastic gradient descent
 - mini-batch stochastic gradient descent

▶ L'entraînement du modèle $\hat{f}(\cdots;\Theta)$ consiste à déterminer des paramètres Θ qui minimisent les fonction de coût

$$\ell(\ldots; \boldsymbol{\Theta})$$
 ou $\mathcal{L}(\ldots; \boldsymbol{\Theta})$.

- ▶ Pour minimiser la fonction de coût, on utilise des descentes de gradient:
- gradient descent
- stochastic gradient descent
- mini-batch stochastic gradient descent

 $lackbox{L'entraînement}$ du modèle $\hat{f}(\cdots;\Theta)$ consiste à déterminer des paramètres Θ qui minimisent les fonction de coût

$$\ell(\ldots; \boldsymbol{\Theta})$$
 ou $\mathcal{L}(\ldots; \boldsymbol{\Theta})$.

- ▶ Pour minimiser la fonction de coût, on utilise des descentes de gradient:
- gradient descent
- stochastic gradient descent
- mini-batch stochastic gradient descent

 $lackbox{L'entraînement}$ du modèle $\hat{f}(\cdots;\Theta)$ consiste à déterminer des paramètres Θ qui minimisent les fonction de coût

$$\ell(\ldots; \boldsymbol{\Theta})$$
 ou $\mathcal{L}(\ldots; \boldsymbol{\Theta})$.

- ▶ Pour minimiser la fonction de coût, on utilise des descentes de gradient:
- gradient descent
- stochastic gradient descent
- mini-batch stochastic gradient descent

DESCENTES DE GRADIENT

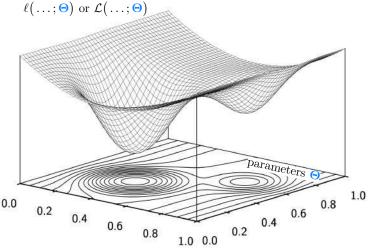
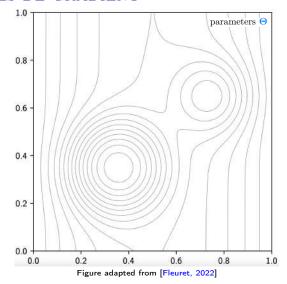
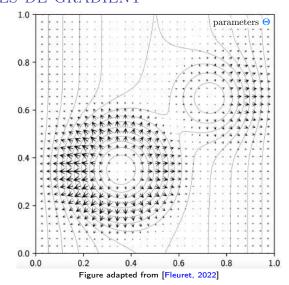
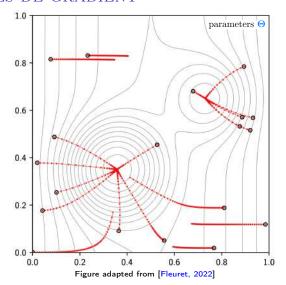


Figure adapted from [Fleuret, 2022]







- ▶ Gradient Descent (GD): on update les paramètres Θ après avoir passé le dataset en entier.
- Descente de gradient appliquée à la fonction de coût totale $\mathcal{L}(\ldots;\Theta)$.

- ▶ Gradient Descent (GD): on update les paramètres Θ après avoir passé le dataset en entier.
- Descente de gradient appliquée à la fonction de coût totale $\mathcal{L}(\ldots;\Theta)$.

```
Algorithm 2: Gradient descent (GD)
```

```
\begin{split} \text{Inputs: model } \hat{f}(\cdot; \Theta) : \mathbb{R}^{d_1} &\longrightarrow \mathbb{R}^{d_2}; \\ \text{dataset } S = \left\{ (\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N \right\}; \\ \text{differentiable loss function } \mathcal{L} : |\Theta| &\longrightarrow \mathbb{R}; \\ \text{random initial parameters } \Theta; \text{ learning rate } \lambda > 0; \text{ nb of epochs } nb\_epochs. \end{split}
```

```
\begin{array}{lll} & \text{for } e=1,\ldots,nb\_epochs \; \text{do} \\ & \text{for } i=1,\ldots,N \; \text{do} & // \; \text{compute predictions (dataset)} \\ & & \hat{y}_i=\hat{f}\left(x_i;\Theta\right) \\ & & \text{end} \\ & & \mathcal{L}\left(\Theta\right):=\mathcal{L}\left(\hat{y}_1,\ldots,\hat{y}_N,y_1,\ldots,y_N;\Theta\right) & // \; \text{compute loss (dataset)} \\ & & \Theta:=\Theta-\lambda\nabla\mathcal{L}\left(\Theta\right) & // \; \text{update gradient (dataset)} \\ & \text{end} \\ & \text{return } \hat{f}\left(:\Theta\right) \end{array}
```

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
                   dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\};
```

differentiable loss function $\mathcal{L}: |\Theta| \longrightarrow \mathbb{R}$;

random initial parameters Θ ; learning rate $\lambda > 0$; nb of epochs nb = epochs.

```
for e = 1, \dots, nb epochs do
    for i = 1, \ldots, N do
```

```
Algorithm 2: Gradient descent (GD)  \begin{aligned} & \text{Inputs: model } \hat{f}(\cdot;\Theta) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2}; \\ & \text{dataset } S = \left\{ (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N \right\}; \\ & \text{differentiable loss function } \mathcal{L} : |\Theta| \longrightarrow \mathbb{R}; \\ & \text{random initial parameters } \Theta; \text{ learning rate } \lambda > 0; \text{ nb of epochs } nb\_epochs. \end{aligned}   \begin{aligned} & \text{for } e = 1, \dots, nb\_epochs \text{ do} \\ & \text{ for } i = 1, \dots, N \text{ do} \\ & \text{ } | \hat{\boldsymbol{y}}_i = \hat{\boldsymbol{f}}(\boldsymbol{x}_i; \Theta) \\ & \text{ end} \\ & \mathcal{L}(\Theta) := \mathcal{L}(\hat{y}_1, \dots, \hat{y}_N, y_1, \dots, y_N; \Theta) \\ & \Theta := \Theta - \lambda \nabla \mathcal{L}(\Theta) \end{aligned}  \end{aligned}  // compute loss (dataset) end return \hat{\boldsymbol{f}}(:\Theta)
```

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
                  dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\};
                  differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
                  random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs.
for e = 1, \ldots, nb epochs do
          for i = 1, \ldots, N do
                                                                                                  // compute predictions (dataset)
         \hat{\boldsymbol{y}}_{i} = \hat{f}(\boldsymbol{x}_{i}; \boldsymbol{\Theta})
        \begin{array}{ll} \mathcal{L}\left(\Theta\right) := \mathcal{L}\left(\hat{\boldsymbol{y}}_{1}, \ldots, \hat{\boldsymbol{y}}_{N}, \boldsymbol{y}_{1}, \ldots, \boldsymbol{y}_{N}; \Theta\right) & \text{// compute loss (dataset)} \\ \Theta := \Theta - \lambda \vee \mathcal{L}\left(\Theta\right) & \text{// update gradient (dataset)} \end{array}
```

// update gradient (dataset)

GRADIENT DESCENT (GD)

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
             dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\};
              differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
              random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs.
for e = 1, \ldots, nb epochs do
       for i = 1, \ldots, N do
                                                                           // compute predictions (dataset)
       \hat{\boldsymbol{y}}_{i} = \hat{f}(\boldsymbol{x}_{i}; \boldsymbol{\Theta})
```

 $\mathcal{L}(\Theta) := \mathcal{L}\left(\hat{y}_1, \dots, \hat{y}_N, y_1, \dots, y_N; \Theta\right) \qquad \text{// compute loss (dataset)} \\ \Theta := \Theta - \lambda \nabla \mathcal{L}\left(\Theta\right) \qquad \text{// update gradient (dataset)}$

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
               dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\};
               differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
               random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs.
for e = 1, \ldots, nb epochs do
        for i = 1, \ldots, N do
                                                                                   // compute predictions (dataset)
        \hat{\boldsymbol{y}}_{i} = \hat{f}(\boldsymbol{x}_{i}; \boldsymbol{\Theta})
         \mathcal{L}(\Theta) := \mathcal{L}\left(\hat{y}_1, \dots, \hat{y}_N, y_1, \dots, y_N; \Theta\right) \qquad \text{// compute loss (dataset)} \\ \Theta := \Theta - \lambda \nabla \mathcal{L}(\Theta) \qquad \text{// update gradient (dataset)} 
                                                                                            // update gradient (dataset)
end
```

Algorithm 2: Gradient descent (GD)

```
\begin{array}{l} \text{Inputs: model } \hat{f}(\cdot;\Theta):\mathbb{R}^{d_1}\longrightarrow\mathbb{R}^{d_2}; \\ & \text{dataset } S=\left\{(\boldsymbol{x_i},\boldsymbol{y_i})\in\mathbb{R}^{d_1}\times\mathbb{R}^{d_2}:i=1,\dots,N\right\}; \\ & \text{differentiable loss function } \mathcal{L}:|\Theta|\longrightarrow\mathbb{R}; \\ & \text{random initial parameters }\Theta; \text{ learning rate } \lambda>0; \text{ nb of epochs } nb\_epochs. \\ \\ \text{for } e=1,\dots,nb\_epochs \text{ do} \\ & \text{ for } i=1,\dots,N \text{ do} \\ & & | \hat{\boldsymbol{y_i}}=\hat{f}(\boldsymbol{x_i};\Theta) \\ & \text{ end} \\ & & \mathcal{L}(\Theta):=\mathcal{L}(\hat{\boldsymbol{y_1}},\dots,\hat{\boldsymbol{y_N}},\boldsymbol{y_1},\dots,\boldsymbol{y_N};\Theta) \\ & & \Theta:=\Theta-\lambda\nabla\mathcal{L}(\Theta) \\ \end{array} \right. \\ // \text{ compute loss (dataset)} \\ & \text{ dataset)} \\ \text{end} \\ \text{return } \hat{f}(\cdot;\Theta) \\ \end{array}
```

- Stochastic Gradient Descent (SGD): on update les paramètres
 Θ après chaque sample du dataset.
- Descente de gradient appliquée à chaque fonction de coût particulière $\ell(\ldots; \Theta)$.

- Stochastic Gradient Descent (SGD): on update les paramètres
 Θ après chaque sample du dataset.
- ▶ Descente de gradient appliquée à chaque fonction de coût particulière $\ell(\ldots;\Theta)$.

dataset $S = \{(x_i, y_i) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, ..., N\};$

Algorithm 3: Stochastic Gradient descent (SGD)

Inputs: model $\hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2}$;

Algorithm 3: Stochastic Gradient descent (SGD)

Inputs: model $\hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2}$;

Algorithm 3: Stochastic Gradient descent (SGD)

Algorithm 3: Stochastic Gradient descent (SGD)

Algorithm 3: Stochastic Gradient descent (SGD)

- Mini-Batch Stochastic Gradient Descent (mb-SGD ou SGD): on update les paramètres Θ après chaque batch du dataset.
- ▶ Descente de gradient appliquée à la fonction de coût totale du batch $\mathcal{L}(\ldots; \Theta)$.
- Cette méthode est la plus efficace.

- Mini-Batch Stochastic Gradient Descent (mb-SGD ou SGD): on update les paramètres Θ après chaque batch du dataset.
- ▶ Descente de gradient appliquée à la fonction de coût totale du batch $\mathcal{L}(\ldots;\Theta)$.
- Cette méthode est la plus efficace.

- Mini-Batch Stochastic Gradient Descent (mb-SGD ou SGD): on update les paramètres Θ après chaque batch du dataset.
- ▶ Descente de gradient appliquée à la fonction de coût totale du batch $\mathcal{L}(\ldots;\Theta)$.
- Cette méthode est la plus efficace.

```
Algorithm 4: Mini-Batch Stochastic Gradient descent (SGD)
```

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
              dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\}:
              differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
              random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs;
              batch size B.
```

Inputs: model $\hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2}$;

MINI-BATCH STOCHASTIC GRADIENT DESCENT (MB-SGD)

```
Algorithm 4: Mini-Batch Stochastic Gradient descent (SGD)
```

nd

4□ > 4□ > 4□ > 4□ > 4□ > 900

```
Algorithm 4: Mini-Batch Stochastic Gradient descent (SGD)
```

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
             dataset S = \{(x_i, y_i) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, ..., N\};
             differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
             random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs;
             batch size B.
for e = 1, \ldots, nb epochs do
       for b = 1, \ldots, nb batches do
         \hat{\boldsymbol{y}}_{b_i} = \hat{f}\left(\boldsymbol{x}_{b_i}; \boldsymbol{\Theta}\right)
         \Theta := \Theta - \lambda \sum_{i=1}^{B} \nabla \ell_{b_i} \left(\Theta \right) // update gradient (batch)
```

```
Algorithm 4: Mini-Batch Stochastic Gradient descent (SGD)
```

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2}:
                                                      dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\};
                                                       differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
                                                       random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs;
                                                       batch size B.
for e = 1, \ldots, nb epochs do
                               for b = 1, \ldots, nb batches do
                                                                \mbox{ for } i=1,\dots,B \mbox{ do } \mbox{ } \mb
                                      \begin{vmatrix} \hat{\boldsymbol{y}}_{b_i} = \hat{f}\left(\boldsymbol{x}_{b_i}; \boldsymbol{\Theta}\right) \\ \ell_{b_i}\left(\boldsymbol{\Theta}\right) := \ell\left(\hat{y}_{b_i}, y_{b_i}; \boldsymbol{\Theta}\right) \\ \text{end} \\ \boldsymbol{\Theta} := \boldsymbol{\Theta} - \lambda \sum_{i=1}^{B} \nabla \ell_{b_i}\left(\boldsymbol{\Theta}\right) \\ \end{vmatrix} / \text{update gradient (batch)}
```

```
Algorithm 4: Mini-Batch Stochastic Gradient descent (SGD)
```

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
              dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\}:
              differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
              random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs;
              batch size B.
for e = 1, \ldots, nb epochs do
        for b = 1, \ldots, nb batches do
                for i = 1, ..., B do // compute predictions and losses (batch)
           \begin{vmatrix} \hat{y}_{b_i} = \hat{f}\left(x_{b_i}; \Theta\right) \\ \ell_{b_i}\left(\Theta\right) := \ell\left(\hat{y}_{b_i}, y_{b_i}; \Theta\right) \end{vmatrix} 
          \Theta := \Theta - \lambda \sum_{i=1}^{B} \nabla \ell_{b_i} \left( \Theta \right)
```

```
Algorithm 4: Mini-Batch Stochastic Gradient descent (SGD)
```

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
                dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\}:
                 differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
                 random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs;
                 batch size B.
for e = 1, \ldots, nb epochs do
         for b = 1, \ldots, nb batches do
                   for i = 1, ..., B do // compute predictions and losses (batch)
                 egin{aligned} \hat{oldsymbol{y}}_{oldsymbol{b_i}} &= \hat{f}\left(oldsymbol{x_{b_i}}; oldsymbol{\Theta}
ight) \ \ell_{b_i}\left(oldsymbol{\Theta}
ight) := \ell\left(\hat{oldsymbol{y}}_{oldsymbol{b_i}}, oldsymbol{y_{b_i}}; oldsymbol{\Theta}
ight) \end{aligned}
                 \mathbf{\Theta} := \mathbf{\Theta} - \lambda \sum_{i=1}^{B} \nabla \ell_{b_i} \left( \mathbf{\Theta} \right)
                                                                                                        // update gradient (batch)
```

Algorithm 4: Mini-Batch Stochastic Gradient descent (SGD)

```
Inputs: model \hat{f}(\cdot; \boldsymbol{\Theta}) : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_2};
                dataset S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : i = 1, \dots, N\}:
                differentiable loss function \mathcal{L}: |\Theta| \longrightarrow \mathbb{R};
                random initial parameters \Theta; learning rate \lambda > 0; nb of epochs nb = epochs;
                batch size B.
for e = 1, \ldots, nb epochs do
         for b = 1, \ldots, nb batches do
                  for i = 1, ..., B do // compute predictions and losses (batch)
                egin{aligned} \hat{oldsymbol{y}}_{oldsymbol{b_i}} &= \hat{f}\left(oldsymbol{x_{b_i}};oldsymbol{\Theta}
ight) \ \ell_{b_i}\left(oldsymbol{\Theta}
ight) := \ell\left(\hat{oldsymbol{y}}_{oldsymbol{b_i}}, oldsymbol{y_{b_i}};oldsymbol{\Theta}
ight) \end{aligned}
                 \Theta := \Theta - \lambda \sum_{i=1}^{B} \nabla \ell_{b_i} (\Theta)
                                                                                                     // update gradient (batch)
end
return \hat{f}(\cdot; \boldsymbol{\Theta})
```

BIBLIOGRAPHIE



Fleuret, F. (2022).

Deep Learning Course.



Wikipedia contributors (2022a).

Gradient — Wikipedia, the free encyclopedia.



Wikipedia contributors (2022b).

 $\label{eq:Gradient} \textbf{Gradient descent} \ -- \ \textbf{Wikipedia}, \ \textbf{the free encyclopedia}.$