

# Optimisation à grande échelle

Joon Kwon

Master 2 — MathSV

14 octobre 2021

# Motivation

Soient deux entiers potentiellement **très grands** :

- $d \geq 1$  représentant la **dimension**
- $n \geq 1$  représentant le **nombre de données**

On s'intéresse typiquement à la minimisation sur  $\mathbb{R}^d$  de fonctions objectifs de la forme :

$$f(x) = \frac{1}{n} \sum_{i=1}^n f_i(x),$$

où les  $f_i: \mathbb{R}^d \rightarrow \mathbb{R}$  sont différentiables.

Le calcul d'un gradient  $\nabla f(x)$  correspond en fait au **calcul de  $n$  gradients**  $\nabla f_i(x)$ .

Exemples :

- **Minimisation du risque empirique** en apprentissage statistique
- Estimation par **maximum de vraisemblance**

Méthode du gradient stochastique

Données massives et mini-batches

Réduction de variance

Algorithmes à mise à l'échelle diagonale

# Méthode du gradient stochastique

# Définition

- $f: \mathbb{R}^d \rightarrow \mathbb{R}$  différentiable
- $x^{(1)} \in \mathbb{R}^d$  quelconque
- $(\gamma^{(t)})_{t \geq 1}$  une suite de pas strictement positifs

$$x^{(t+1)} = x^{(t)} - \gamma^{(t)} \hat{g}^{(t)}, \quad t \geq 1,$$

où  $\hat{g}^{(t)}$  est un **estimateur non biaisé** de  $\nabla f(x^{(t)})$ , autrement dit une variable aléatoire telle que :

$$\mathbb{E} \left[ \hat{g}^{(t)} \mid x^{(t)} \right] = \nabla f(x^{(t)}).$$

- Les itérées sont aléatoires.
- On perd en précision mais  $\hat{g}^{(t)}$  peut être beaucoup moins coûteux à obtenir que  $\nabla f(x^{(t)})$ .

# Hypothèse : second moment borné

On suppose que les estimateurs du gradient  $\hat{g}^{(t)}$  ont un second moment borné

Soit  $\sigma > 0$  tel que :

$$\mathbb{E} \left[ \left\| \hat{g}^{(t)} \right\|^2 \right] \leq \sigma^2, \quad t \geq 1.$$

# Garantie dans le cas régulière et non-convexe

## Théorème

Soit  $T \geq 1$  et  $L > 0$ . On suppose que  $f$  est  $L$ -régulière. La méthode du *gradient stochastique* avec  $\gamma^{(t)} = \sqrt{2/(L\sigma^2 t)}$  (pour  $t \geq 1$ ) :

$$\mathbb{E} \left[ \frac{\sum_{t=1}^T \gamma^{(t)} \|\nabla f(x^{(t)})\|^2}{\sum_{t=1}^T \gamma^{(t)}} \right] \leq \sigma \sqrt{\frac{L}{2}} \times \frac{f(x^{(1)}) - f(x^*) + 1 + \log T}{\sqrt{T+1}}.$$

Cela garantit un point stationnaire approché.

# Garantie dans le cas convexe

## Théorème

Soit  $T \geq 1$ . On suppose que  $f$  est convexe. La méthode du gradient stochastique avec  $\gamma^{(t)} = \sqrt{1/(\sigma^2 t)}$  (pour  $t \geq 1$ ) :

$$\mathbb{E} \left[ f \left( \frac{\sum_{t=1}^T \gamma^{(t)} x^{(t)}}{\sum_{t=1}^T \gamma^{(t)}} \right) \right] - f(x^*) \leq \frac{\sigma}{2} \times \frac{\|x^{(1)} - x^*\|^2 + 1 + \log T}{\sqrt{T+1}}.$$

Cela garantit un minimiseur approché.



# Garantie dans le cas fortement convexe et régulière

## Théorème

Soit  $T \geq 1$ ,  $K, L > 0$ . On suppose que  $f$  est  $K$ -fortement convexe et  $L$ -régulière. La méthode du *gradient stochastique* avec  $\gamma^{(t)} = \sqrt{1/(Kt)}$  (pour  $t \geq 1$ ) :

$$\mathbb{E} \left[ f \left( x^{(T)} \right) \right] - f(x^*) \leq \frac{2L\sigma^2}{K^2 T}.$$

Cela garantit un minimiseur approché.

# Données massives et mini-batches

# Mini-batches

On considère des fonctions objectifs de la forme :

$$f(x) = \frac{1}{n} \sum_{i=1}^n f_i(x),$$

- **Idée :** construire des estimateurs du gradient en ne tirant aléatoirement **qu'un petit nombre de données à chaque itération**.
- Soit  $k \geq 1$  un entier.
- À chaque étape  $t \geq 1$ , on **tire uniformément** un sous-ensemble :

$$B^{(t)} \subset \{1, \dots, n\} \quad \text{tel que} \quad |B^{(t)}| = k.$$

- $B^{(t)}$  est appelé un **mini-batch**.
- On construit :

$$\hat{g}^{(t)} = \frac{1}{k} \sum_{i \in B^{(t)}} \nabla f_i(x^{(t)}).$$

- On a bien :

$$\mathbb{E} \left[ \hat{g}^{(t)} \mid x^{(t)} \right] = \nabla f(x^{(t)}).$$

# Remarques sur les mini-batches

- Dans la pratique, on **renumérote une fois aléatoirement les  $i$** , puis on choisit les mini-batches :

$$B^{(1)} = \{1, \dots, k\}, \quad B^{(2)} = \{k+1, \dots, 2k\}, \dots$$

- Quand  $n \gg 1$ , on prend  $k \ll n$ . Alors, calculer un estimateur du gradient  $\hat{g}^{(t)}$  sur un mini-batch **est beaucoup moins coûteux** que calculer le gradient exact  $\nabla f(x^{(t)})$ .
- En revanche, on perd en vitesse de convergence :

	Gradient exact	Gradient stochastique
Régulière	$\frac{1}{T}$	$\frac{1}{\sqrt{T}}$
Fortement convexe et régulière	$\left(1 - \frac{K}{L}\right)^T$	$\frac{1}{T}$

- Reste très avantageux pour les premières itérations, pour lesquelles la précision des gradients est moins importante.
- Question :** peut-on combiner
  - l'avantage computationnel des mini-batches, et
  - des vitesses de convergence rapides ?
- Réponse :** **oui**, grace aux techniques de **réduction de variance**.

# Réduction de variance

# Principe

**Idée de base :** un estimateur peut être amélioré si on connaît une deuxième variable aléatoire positivement corrélée à la première et dont l'espérance est connue.

- Soit  $X, Y$  deux variables aléatoires réelles
- On suppose que  $\mathbb{E}[Y]$  est connu.
- But : estimer  $\mathbb{E}[X]$ .

Alors,

- $X$  est un estimateur de  $\mathbb{E}[X]$ .
- $Z = X - (Y - \mathbb{E}[Y])$  est aussi un estimateur de  $\mathbb{E}[X]$ . De plus,

$$\text{Var } Z = \text{Var } X + \text{Var } Y - 2 \text{Cov}(X, Y).$$

- Si  $\text{Cov}(X, Y) > \frac{1}{2} \text{Var } Y$ , on a  $\text{Var } Z < \text{Var } X$  et alors  $Z$  est un “meilleur” estimateur que  $X$ .

# SAGA (Defazio, Bach & Lacoste-Julien, 2014)

$$f(x) = \frac{1}{n} \sum_{i=1}^n f_i(x).$$

- $x^{(1)} \in \mathbb{R}^d$  quelconque. Calculer et stocker le gradient de chaque fonction  $f_i$  en  $x^{(1)}$  :

$$g^{[i]} \leftarrow \nabla f_i(x^{(1)}), \quad 1 \leq i \leq n.$$

- Pour  $t \geq 1$ ,
  - On tire  $i^{(t)} \sim \mathcal{U}(\{1, \dots, n\})$
  - On calcule l'estimateur  $\hat{g}^{(t)} = \nabla f_{i^{(t)}}(x^{(t)}) - g^{[i^{(t)}]} + \frac{1}{n} \sum_{i=1}^n g^{[i]}$
  - On itère  $x^{(t+1)} = x^{(t)} - \gamma^{(t)} \hat{g}^{(t)}$
  - On met à jour  $g^{[i^{(t)}]} \leftarrow \nabla f_{i^{(t)}}(x^{(t)})$

On calcule un seul gradient par itération.

Il est nécessaire de stocker  $n$  gradients.

Dans la pratique, on utilise des minibatches qui ne se chevauchent pas.

$$B^{(1)} = \{1, \dots, k\}, \quad B^{(2)} = \{k+1, \dots, 2k\}, \dots$$

On ne doit alors stocker qu'un nombre de gradients correspondant au nombre de mini-batches différents utilisés ( $\simeq n/k$ ).

# SVRG (Johnson & Zhang, 2013)

- $\tau \geq 1$  un paramètre entier
- $x^{(1)} \in \mathbb{R}^d$  quelconque

Pour  $t \geq 1$ ,

- On calcule  $\mu^{(t)} = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^{(t)})$
- On pose  $x^{(t,1)} = x^{(t)}$ .
- Pour  $1 \leq s \leq \tau$ ,
  - On tire  $i^{(t,s)} \sim \mathcal{U}(\{1, \dots, n\})$ .
  - On calcule l'estimateur  $\hat{g}^{(t,s)} = \nabla f_{i^{(t,s)}}(x^{(t,s)}) - \nabla f_{i^{(t,s)}}(x^{(t)}) + \mu^{(t)}$
  - On itère  $x^{(t,s+1)} = x^{(t,s)} - \gamma^{(t,s)} \hat{g}^{(t,s)}$
- On pose  $x^{(t+1)} = x^{(t,\tau)}$

On calcule  $n$  gradients tous les  $\tau$  itérations, et 2 gradients à chaque itération.  
Il est nécessaire de stocker 1 gradient.



# Vitesses de convergence

SAGA et SVRG garantissent

	Vitesse de convergence
Régulière	$\frac{1}{T}$
Fortement convexe et régulière	$\alpha^T$

pour un certain  $\alpha$  dépendant des propriétés des fonctions  $f_i$  (et moins avantageux que  $(1 - K/L)$ )

# Algorithmes à mise à l'échelle diagonale

# Rappel : Méthodes de Newton et quasi-Newton

- Descente de gradient :

$$\begin{aligned}x^{(t+1)} &= x^{(t)} - \gamma^{(t)} \nabla f(x^{(t)}) \\&= \arg \min_{x \in \mathbb{R}^d} \left\{ f(x^{(t)}) + \nabla f(x^{(t)})^\top (x - x^{(t)}) + \frac{1}{2\gamma^{(t)}} \|x - x^{(t)}\|^2 \right\}\end{aligned}$$

- Méthode de Newton :

$$\begin{aligned}x^{(t+1)} &= x^{(t)} - \left( \nabla^2 f(x^{(t)}) \right)^{-1} \nabla f(x^{(t)}) \\&= \arg \min_{x \in \mathbb{R}^d} \left\{ f(x^{(t)}) + \nabla f(x^{(t)})^\top (x - x^{(t)}) + \frac{1}{2} (x - x^{(t)})^\top \nabla^2 f(x^{(t)}) (x - x^{(t)}) \right\}\end{aligned}$$

- Méthode quasi-Newton :

$$\begin{aligned}x^{(t+1)} &= x^{(t)} - \left( B^{(t)} \right)^{-1} \nabla f(x^{(t)}) \\&= \arg \min_{x \in \mathbb{R}^d} \left\{ f(x^{(t)}) + \nabla f(x^{(t)})^\top (x - x^{(t)}) + \frac{1}{2} (x - x^{(t)})^\top B^{(t)} (x - x^{(t)}) \right\}\end{aligned}$$

- $B^{(t)}$  est une approximation de  $\nabla^2 f(x^{(t)})$  construite à partir des seuls gradients,
- $B^{(t)}$  est une matrice de taille  $d \times d$ , dont l'utilisation est coûteuse quand  $d \gg 1$  (à l'exception de L-BFGS).

# AdaGrad et RMSProp

**Idée** : se restreindre pour  $B^{(t)}$  aux **matrices diagonales** pour avoir un coût computationnel par étape de  $O(d)$ .

Cela revient à appliquer un pas (step-size) différent à chaque composante. Si on écrit  $B^{(t)} = \text{diag}(b_1^{(t)}, \dots, b_d^{(t)})$ ,

$$x^{(t+1)} = x^{(t)} - \left(B^{(t)}\right)^{-1} \nabla f(x^{(t)}) = x^{(t)} - \begin{pmatrix} (b_1^{(t)})^{-1} \frac{\partial f}{\partial x_1}(x^{(t)}) \\ \vdots \\ (b_d^{(t)})^{-1} \frac{\partial f}{\partial x_d}(x^{(t)}) \end{pmatrix}$$

- **AdaGrad** correspond à :

$$b_i^{(t)} = \frac{1}{\gamma} \left( \varepsilon + \sqrt{\sum_{s=1}^t \left( \frac{\partial f}{\partial x_i}(x^{(s)}) \right)^2} \right), \quad 1 \leq i \leq d.$$

- **RMSProp** correspond à :

$$b_i^{(t)} = \frac{1}{\gamma} \left( \varepsilon + \sqrt{\sum_{s=1}^t \beta^{t-s} \left( \frac{\partial f}{\partial x_i}(x^{(s)}) \right)^2} \right), \quad 1 \leq i \leq d.$$

où  $\varepsilon = 10^{-10}$  et  $\beta = 0.999$  en pratique.

# Adam

Adam est une sophistication de RMSProp :

- les matrices  $B^{(t)}$  sont les mêmes
- le gradient courant  $\nabla f(x^{(t)})$  est remplacé par une somme des gradients précédemment observés, pondérés par récence.

$$b_i^{(t)} = \frac{1}{\gamma} \left( \varepsilon + \sqrt{\sum_{s=1}^t \beta_2^{t-s} \left( \frac{\partial f}{\partial x_i}(x^{(s)}) \right)^2} \right), \quad 1 \leq i \leq d$$

$$x^{(t+1)} = x^{(t)} - (B^{(t)})^{-1} \left( \sum_{s=1}^t \beta_1^{t-s} \nabla f(x^{(s)}) \right)$$

où  $\varepsilon = 10^{-10}$ ,  $\beta_1 = 0.9$  et  $\beta_2 = 0.999$  en pratique.

# Remarques

- On peut remplacer dans ces algorithmes  $\nabla f(x^{(t)})$  par un estimateur  $\hat{g}^{(t)}$ .
- RMSProp et Adam font partie des algorithmes les plus performants pour, entre autres, l'entraînement des réseaux de neurones.