Stochastic Optimization

Hi! PARIS Summer School 2021 on
AI & Data for Science, Business and Society

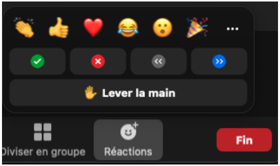
Aymeric Dieuleveut

July 2021

Today's Roadmap

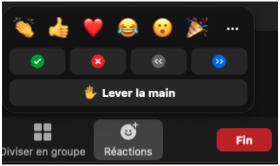
- Motivation: why is Optimization important and why it is useful?
- From GD to SGD.
- Advanced algorithms: Variance Reduction, Deep Learning
- Statistical point of view on Optimization.

Some questions for you first :D



Who knows ?

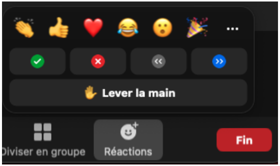
Some questions for you first :D



Who knows ?

- His/her own name

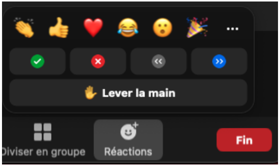
Some questions for you first :D



Who knows ?

- His/her own name
- What GD is?

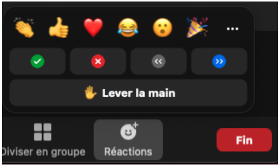
Some questions for you first :D



Who knows ?

- His/her own name
- What GD is?
- What a smooth function is?

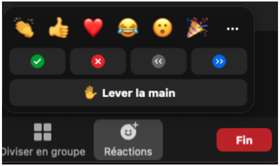
Some questions for you first :D



Who knows ?

- His/her own name
- What GD is?
- What a smooth function is?
- How fast GD converges for smooth functions?

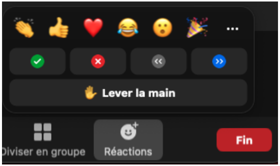
Some questions for you first :D



Who knows ?

- His/her own name
- What GD is?
- What a smooth function is?
- How fast GD converges for smooth functions?
- Which algorithm is fastest SGD or GD?

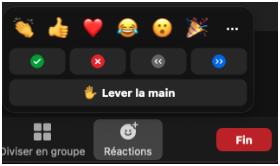
Some questions for you first :D



Who knows ?

- His/her own name
- What GD is?
- What a smooth function is?
- How fast GD converges for smooth functions?
- Which algorithm is fastest SGD or GD?
- What SVRG is?

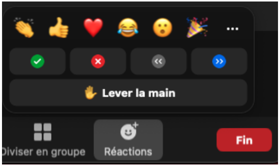
Some questions for you first :D



Who knows ?

- His/her own name
- What GD is?
- What a smooth function is?
- How fast GD converges for smooth functions?
- Which algorithm is fastest SGD or GD?
- What SVRG is?
- About Rademacher complexities?

Some questions for you first :D



Who knows ?

- His/her own name
- What GD is?
- What a smooth function is?
- How fast GD converges for smooth functions?
- Which algorithm is fastest SGD or GD?
- What SVRG is?
- About Rademacher complexities?

Outline

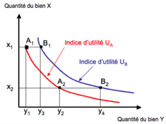
Optimization : finding the minimal (maximal) value of a function over a set

$$\min_{w \in \Theta \subset \mathbb{R}^d} \quad f(w)$$

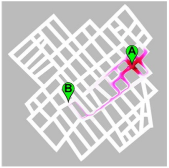
Optimization is everywhere

Many problems are formalized as finding the **optimum** of a function: $\min_w f(w)$.
In various domains:

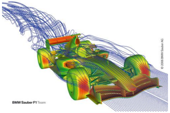
Economics



GPS

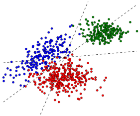


Aeronautics

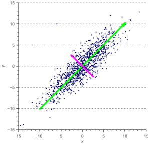


In Machine learning related applications

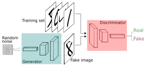
Supervised Learning



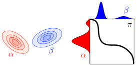
Unsupervised



Gans



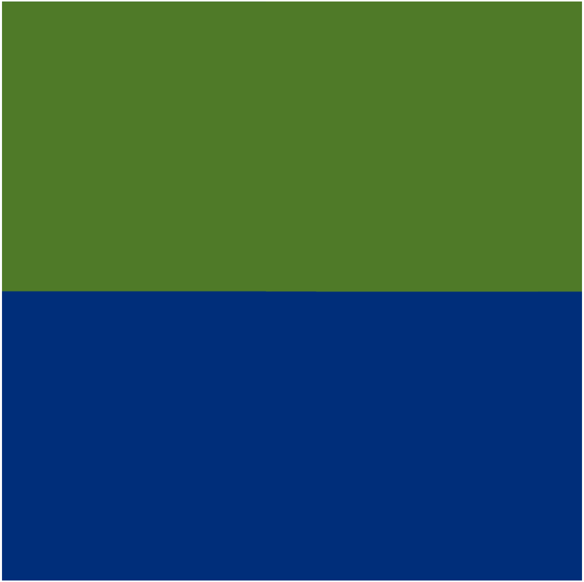
Optimal transport



Is it difficult ? Why study it?

Is Optimization a (hard) problem? Why study it

↬It depends !



Is Optimization a (hard) problem? Why study it

↬It depends !

The problem can be easily solved numerically

Yet, important to understand the methods

Is Optimization a (hard) problem? Why study it

↬It depends !

The problem can be easily solved numerically

Yet, important to understand the methods

The problem is hard to solve

The choice of the algorithm impacts the performance

$\Rightarrow$ Crucial to understand the algorithms !

Is Optimization a (hard) problem? Why study it

↬It depends !

The problem can be easily solved numerically

Yet, important to understand the methods

The problem is hard to solve

The choice of the algorithm impacts the performance

⇒ Crucial to understand the algorithms !

Last 20 years?

1. More computational power
2. More data
3. New algorithms, new models

↔ **Large scale framework**

↔ **Deep Learning**

Example 1: Logistic regression on Scikit-Learn

***solver** : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default='lbfgs'*

Algorithm to use in the optimization problem.

- For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones.
- For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss; 'liblinear' is limited to one-versus-rest schemes.
- 'newton-cg', 'lbfgs', 'sag' and 'saga' handle L2 or no penalty
- 'liblinear' and 'saga' also handle L1 penalty
- 'saga' also supports 'elasticnet' penalty
- 'liblinear' does not support setting `penalty='none'`

Note that 'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.

New in version 0.17: Stochastic Average Gradient descent solver.

New in version 0.19: SAGA solver.

Changed in version 0.22: The default solver changed from 'liblinear' to 'lbfgs' in 0.22.

Figure: Scikit-Learn documentation, logistic regression.

Example 2: Neural Network Playground

Neural Network playground (try it!)

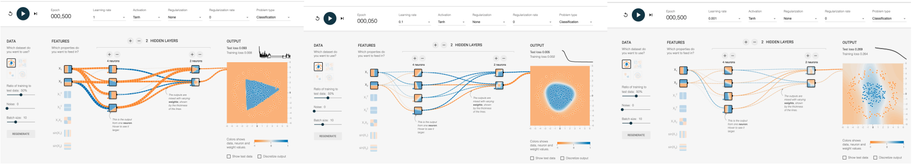


Figure: Model learned after 500 epochs depending on the learning rate, deep Learning

Example 3: Federated Learning

SCAFFOLD: CORRECTING LOCAL UPDATES [Karimireddy et al., 2020]

Algorithm Scaffold (server-side)

Parameters: client sampling rate ρ, global learning rate η_g

initialize θ, $c = c_1, \dots, c_K = 0$
for each round $t = 0, 1, \dots$ **do**
  $\mathcal{S}_t \leftarrow$ random set of $m = \lceil \rho K \rceil$ clients
  for each client $k \in \mathcal{S}_t$ in parallel **do**
    $(\Delta\theta_k, \Delta c_k) \leftarrow$ ClientUpdate (k, θ, c)
  $\theta \leftarrow \theta + \frac{\eta_g}{m} \sum_{k \in \mathcal{S}_t} \Delta\theta_k$
  $c \leftarrow c + \frac{1}{K} \sum_{k \in \mathcal{S}_t} \Delta c_k$

Algorithm ClientUpdate(k, θ, c)

Parameters: batch size B, # of local steps L, local learning rate η_l

Initialize $\theta_k \leftarrow \theta$
for each local step $1, \dots, L$ **do**
  $\mathcal{B} \leftarrow$ mini-batch of B examples from $\mathcal{D}_k$
  $\theta_k \leftarrow \theta_k - \eta_l(\frac{n_k}{B} \sum_{d \in \mathcal{B}} \nabla f(\theta; d) - c_k + c)$
$c_k^+ \leftarrow c_k - c + \frac{1}{L\eta_l}(\theta - \theta_k)$
send $(\theta_k - \theta, c_k^+ - c_k)$ to server
$c_k \leftarrow c_k^+$

- Correction terms $c_1, \dots, c_K$ are a form of variance reduction (cf Aymeric's tutorial)
- Can show convergence rates which beat parallel SGD

18

Figure: In Federated Learning, crucial to adapt the algorithm!

Today's Approach

Part 1: Introduction

- Understand what can make optimization hard
- Briefly review some classical learning situations from this perspective

Part 2: From GD to SGD

- First order Optimization, Stochastic Optimization
- Tradeoffs
- What influences the convergence of SGD

Part 3: Advanced Stochastic Optimization methods*

- Variance Reduction
- Methods for Deep Learning

Part 4: Insights from Statistical Learning theory*

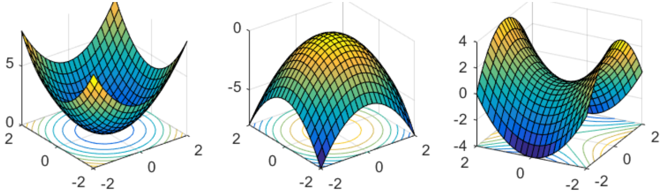
- How precisely should I optimize?
- Rademacher complexities

What makes optimization hard:

$$\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$$

What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 1. Convexity.

Why?

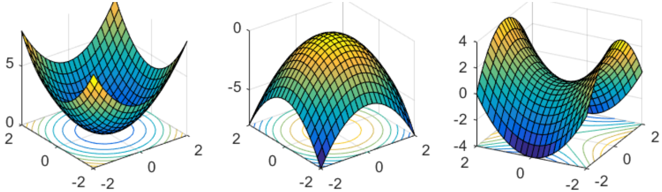


- A non-convex function can have many local minima
- For a convex function, a local minimum is always global.

Challenges: Non-convexity, ...

What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 1. Convexity.

Why?



- A non-convex function can have many local minima
- For a convex function, a local minimum is always global.

Challenges: Non-convexity, ...

What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 2. Dimension of w, set Θ, complexity of f

a. Dimension d: $\Theta \subset \mathbb{R}^d$, d might be very large (typically millions)

What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 2. Dimension of w, set Θ, complexity of f

a. Dimension d: $\Theta \subset \mathbb{R}^d$, d might be very large (typically millions)

b. Set Θ: (if Θ is a convex set.)

- May be described implicitly (via equations):
  $\Theta = \{w \in \mathbb{R}^d \text{ s.t.} \|w\|_2 \leqslant R \text{ and } \langle w, 1 \rangle = r\}$.
  ↬ Use dual formulation of the problem.
- Projection might be difficult or impossible.
  ↬ use only first order methods

What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 2. Dimension of w, set Θ, complexity of f

a. Dimension d: $\Theta \subset \mathbb{R}^d$, d might be very large (typically millions)

b. Set Θ: (if Θ is a convex set.)

- May be described implicitly (via equations): $\Theta = \{w \in \mathbb{R}^d \text{ s.t.} \|w\|_2 \leqslant R \text{ and } \langle w, 1 \rangle = r\}$.
  ↬ Use dual formulation of the problem.
- Projection might be difficult or impossible.
  ↬ use only first order methods

c. Structure of f. If $f(w) = \frac{1}{n} \sum_{i=1}^{n} F_i(w)$, is the average of n functions, computing a gradient has a cost proportional to n.

Challenges: Non-convexity of f, large d, large n, implicit set Θ, ...

What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 3. Irregularity of the function

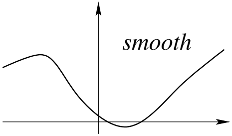
a. Smoothness

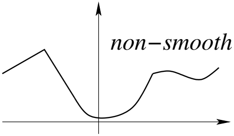
- A function f is L-smooth iif it is twice differentiable and $\forall w \in \mathbb{R}^d, \text{ eig.}\left[f''(w)\right] \leqslant L$

What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 3. Irregularity of the function

a. Smoothness

- A function f is L-smooth iif it is twice differentiable and $\forall w \in \mathbb{R}^d,\ \text{eig.}\left[f''(w)\right] \leqslant L$

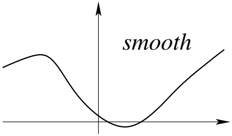


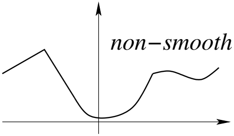


What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 3. Irregularity of the function

a. Smoothness

- A function f is L-smooth iif it is twice differentiable and $\forall w \in \mathbb{R}^d,\ \text{eig.}\left[f''(w)\right] \leqslant L$





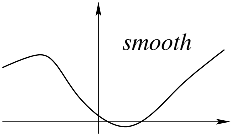
b. Strong Convexity

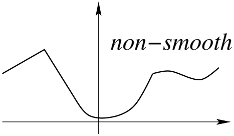
- A twice differentiable f is μ-strongly convex iif. $\forall w \in \mathbb{R}^d,\ \text{eig}\left[f''(w)\right] \geqslant \mu$.

What makes optimizing $\min_{w \in \Theta \subset \mathbb{R}^d} f(w)$ hard: 3. Irregularity of the function

a. Smoothness

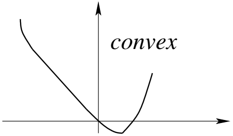
- A function f is L-smooth iif it is twice differentiable and $\forall w \in \mathbb{R}^d,\ \text{eig.}\big[f''(w)\big] \leqslant L$

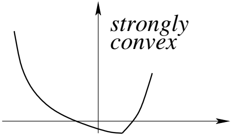




b. Strong Convexity

- A twice differentiable f is μ-strongly convex iif. $\forall w \in \mathbb{R}^d,\ \text{eig}\big[f''(w)\big] \geqslant \mu$.



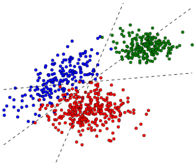


Challenges: Non-convexity of f, large d, large n, implicit set Θ, non-smoothness, non-strongly-convex.

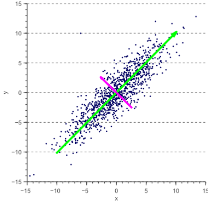
Conclusion: Those are the most frequent challenges. What happens for the examples?

Focus on the 4 Machine learning examples given before

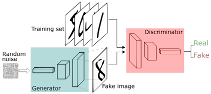
Supervised Learning



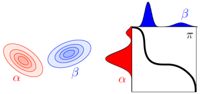
Unspervised



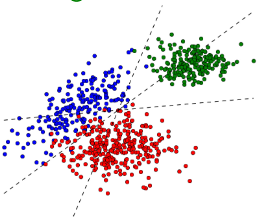
Gans



Optimal transport



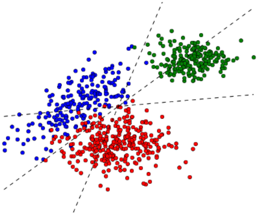
Examples and Challenges 1/4 , Supervised Machine Learning



Consider an input/output pair $(X, Y) \in \mathcal{X} \times \mathcal{Y}$, $(X, Y) \sim \rho$.

Function $w : \mathcal{X} \to \mathbb{R}$, s.t. $w(X)$ good prediction for Y.
Model w parametrized in R^d

Examples and Challenges 1/4 , Supervised Machine Learning



Consider an input/output pair $(X, Y) \in \mathcal{X} \times \mathcal{Y}$, $(X, Y) \sim \rho$.

Function $w : \mathcal{X} \to \mathbb{R}$, s.t. $w(X)$ good prediction for Y.
Model w parametrized in R^d

Consider a loss function $\ell : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}_+$

Define the Generalization risk:

$$\mathcal{R}(w) := \mathbb{E}_\rho \left[\ell(Y, w(X)\rangle)\right].$$

Empirical Risk minimization
Data: n observations $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \ldots, n$, **i.i.d.**
Find $\hat{w}$ solution of

$$\min_{w \in \Theta \subset \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, w(x_i)\big) \quad + \quad \mu\Omega(w).$$

convex data fitting term $+$ regularizer

Challenges: n potentially large (very often!)

ERM:

$$\min_{w \in \Theta \subset \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, w(x_i)\big) \quad + \quad \mu \Omega(w).$$

Encompasses many methods:

Model $w(X)$	Linear Models $\langle w, \Phi(X) \rangle^*$					Non-linear
Name	Least Squares	Lasso	Logistic Reg.	SVM	Binary	Neural Nets
Loss ℓ	Square loss		Logistic loss	Hinge loss	01	(Sq. loss)
Regul. $\Omega(w)$	(Ridge)	$\|\cdot\|_1$				

Examples and Challenges 1/4 , Supervised Machine Learning

ERM:

$$\min_{w \in \Theta \subset \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, w(x_i)) \quad + \quad \mu \Omega(w).$$

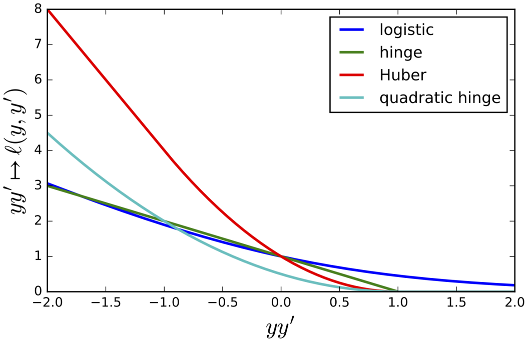
Encompasses many methods:

Model $w(X)$	Linear Models $\langle w, \Phi(X) \rangle^*$					Non-linear
Name	Least Squares	Lasso	Logistic Reg.	SVM	Binary	Neural Nets
Loss ℓ	Square loss		Logistic loss	Hinge loss	01	(Sq. loss)
Regul. $\Omega(w)$	(Ridge)	$\|\| \cdot \|\|_1$				
Large d, n						
Convex						
Smooth						
Strongly convex						

*for features $\Phi(X) \in \mathbb{R}^d$.

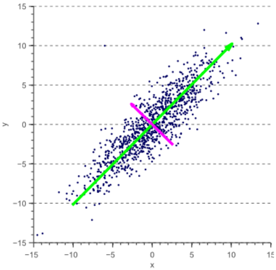
Reminder: Different losses for classification

- Logistic loss, $\ell(y, y') = \log(1 + e^{-yy'})$
- Hinge loss, $\ell(y, y') = (1 - yy')_+$
- Quadratic hinge loss, $\ell(y, y') = \frac{1}{2}(1 - yy')_+^2$
- Huber loss $\ell(y, y') = -4yy'\mathbb{1}_{yy'<-1} + (1 - yy')_+^2\mathbb{1}_{yy'\geqslant -1}$



- These losses can be understood as a convex approximation of the 0/1 loss $\ell(y, y') = \mathbb{1}_{yy'\leqslant 0}$

Examples and Challenges 2/4 Unspervised



PCA ($k = 1$):

1. $\max_{w/||w||\leqslant 1} w^\top A w$.
2. Set $\Theta = \mathcal{B}(0, 1) \subset \mathbb{R}^d$ is convex
3. Convex function $w \mapsto w^\top A w$
4. we look for the max:
   this is thus equivalent to minimizing a concave function and not a "convex problem".

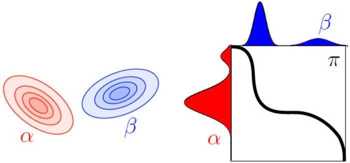
Challenges:

- Non convex
- Large d

Examples and Challenges 3/4: Optimal transport

Objective function:

$$\min_{\pi \in \Pi} \int c(x, y) \mathrm{d}\pi(x, y)$$



- Π set of probability distributions
- $c(x, y)$ "distance" from x to y.

$+$ regularization

Kantorovic formulation of OT.

$\looparrowright$ alternating directions algorithms, ...

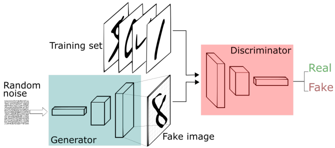
Challenges:

- Non convex
- Optimization over a complex set (measures), etc.

Examples and Challenges 4/4: Generative Adversarial Networks

Objective function:

$$\min_{G} \max_{D} \{\mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]\}$$

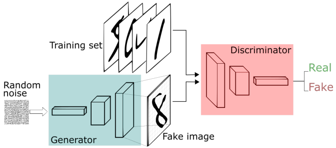


- D discriminator: tries to discriminate between real and fake images
- G generator: tries to fool the discriminator.

Examples and Challenges 4/4: Generative Adversarial Networks

Objective function:

$$\min_G \max_D \{\mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]\}$$



- D discriminator: tries to discriminate between real and fake images
- G generator: tries to fool the discriminator.

Challenges:

- minimax optimization →non convex optimization
- Deep networks for generator and discriminator: non convex functions, extremely high dimension d
- Trained with extremely large quantities of data (large n)...

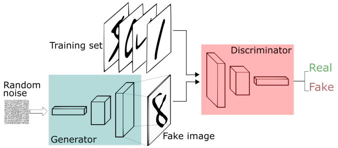
Overall Summary

- We express problems as minimizing a function over a set
- We have listed the main challenges and given examples in classical frameworks esp. Supervised Learning.
- We have to propose algorithms that can be efficient :
  - In large dimension
  - With a high number of observations n

Examples and Challenges 4/4: Generative Adversarial Networks

Objective function:

$$\min_G \max_D \{\mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]\}$$



- D discriminator: tries to discriminate between real and fake images
- G generator: tries to fool the discriminator.

Challenges:

- minimax optimization →non convex optimization
- Deep networks for generator and discriminator: non convex functions, extremely high dimension d
- Trained with extremely large quantities of data (large n)...

Overall Summary

- We express problems as minimizing a function over a set
- We have listed the main challenges and given examples in classical frameworks esp. Supervised Learning.
- We have to propose algorithms that can be efficient :
  - In large dimension
  - With a high number of observations n

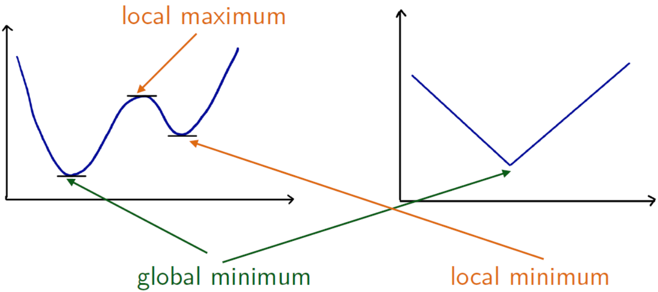
Let's now dive into the optimization algorithms themselves !

Outline

1. Motivation: what is Optimization and why study it?
   - What makes optimization difficult?
   - Detailed Examples
2. Gradient descent procedures
   - Visualization and intuition
   - Gradient Descent
   - Convergence rates for GD and interpretation
   - Stochastic Gradient Descent
3. Advanced Stochastic Optimization Algorithms
   - Variance reduced methods
   - Gradient descent for neural networks
4. Insights from Statistical Learning Theory
   - Set-up
   - Convex functions: basic ideas
   - Empirical risk minimization: convergence rates

Aim: minimizing a function $f : \mathbb{R}^d \to \mathbb{R}$

d: dimension of the search space.

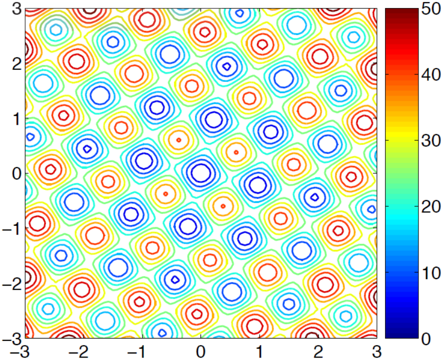


Level sets

One-dimensional (1-D) representations are often misleading, we therefore often represent level-sets of functions

$$\mathcal{C}_c = \{w \in \mathbb{R}^d, f(w) = c\}.$$

Example of level sets in dimension two



Gradient - Definition

The gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ in w denoted as $\nabla f(w)$ is the vector of partial derivatives

$$\nabla f(w) = \begin{pmatrix} \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_d} \end{pmatrix}$$

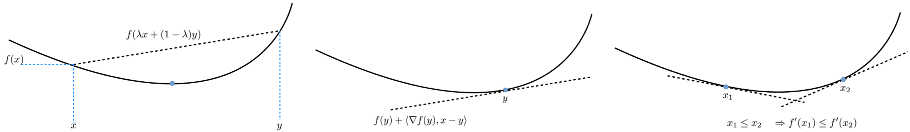
Exercise

- If $f : \mathbb{R} \to \mathbb{R}$, $\nabla f(w) = f'(w)$
- $f(w) = \langle a, w \rangle$: $\nabla f(w) = a$
- $f(w) = w^T A w$: $\nabla f(w) = (A + A^T) w$
- Particular case: $f(w) = \|w\|^2$, $\nabla f(w) = 2w$.

Optimality conditions with convexity

Convexity - Three characterizations

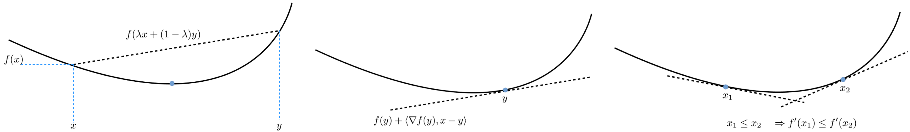
1. We say that $f : \mathbb{R}^d \to \mathbb{R}$ is convex if ($\mathbb{R}^d$ is convex and if)
$$f(\lambda x + (1-\lambda)y) \leqslant \lambda f(x) + (1-\lambda) f(y), \quad \text{for all } x, y \in \mathbb{R}^d, \lambda \in [0,1].$$
2. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if
$$f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle, \quad \text{for all } x, y \in \mathbb{R}^d.$$
3. A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if
$$\nabla^2 f(x) \succeq 0, \quad \text{for all } x,$$
that is $h^T \nabla^2 f(x) h \geqslant 0$, for all $h \in \mathbb{R}^d$.



Optimality conditions with convexity

Convexity - Three characterizations

1. We say that $f : \mathbb{R}^d \to \mathbb{R}$ is convex if ($\mathbb{R}^d$ is convex and if)
$$f(\lambda x + (1-\lambda)y) \leqslant \lambda f(x) + (1-\lambda) f(y), \quad \text{for all } x, y \in \mathbb{R}^d, \lambda \in [0,1].$$
2. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if
$$f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle, \quad \text{for all } x, y \in \mathbb{R}^d.$$
3. A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if
$$\nabla^2 f(x) \succeq 0, \quad \text{for all } x,$$
that is $h^T \nabla^2 f(x) h \geqslant 0$, for all $h \in \mathbb{R}^d$.

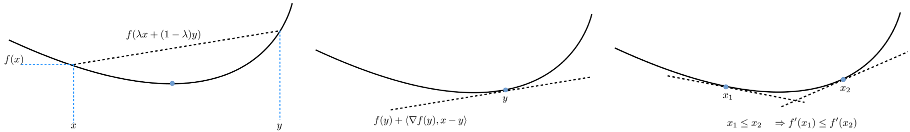


For a convex function, any local minimum is a global minimum.

Optimality conditions with convexity

Convexity - Three characterizations

1. We say that $f : \mathbb{R}^d \to \mathbb{R}$ is convex if ($\mathbb{R}^d$ is convex and if)
$$f(\lambda x + (1-\lambda)y) \leqslant \lambda f(x) + (1-\lambda) f(y), \quad \text{for all } x, y \in \mathbb{R}^d, \lambda \in [0,1].$$
2. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if
$$f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle, \quad \text{for all } x, y \in \mathbb{R}^d.$$
3. A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if
$$\nabla^2 f(x) \succeq 0, \quad \text{for all } x,$$
that is $h^T \nabla^2 f(x) h \geqslant 0$, for all $h \in \mathbb{R}^d$.



For a convex function, any local minimum is a global minimum.

⇒ Algorithmically, how to can we find the optimal point

First attempt: Exhaustive search

Consider the problem

$$w^\star \in \operatorname*{argmin}_{w \in [0,1]^d} f(w).$$

One can optimize this problem on a grid of $[0,1]^d$. For example, if the function f is regular enough, in dimension 1, to achieve a precision of ε we need $\lfloor 1/\varepsilon \rfloor$ evaluation of f. In dimension d, we need $\lfloor 1/\varepsilon \rfloor^d$ evaluations.

For example, evaluating the expression

$$f(w) = \|w\|_2^2,$$

to obtain a precision of $\varepsilon = 10^{-2}$ requires:

- $1,75.10^{-3}$ seconds in dimension 1
- $1,75.10^{15}$ seconds in dimension 10, i.e., nearly 32 millions years.

→ Prohibitive in high dimensions (curse of dimensionality, term introduced by **bellman1961adaptive**)

→ Solution Use **local information**.

Use local information: two Classes of algorithms

Key idea: At any point w_0 we can compute the value of the function $f(w_0)$, but also the direction in which the function increases the most $\nabla f(w_0)$ and the curvature $\nabla^2 f(w_0)$.

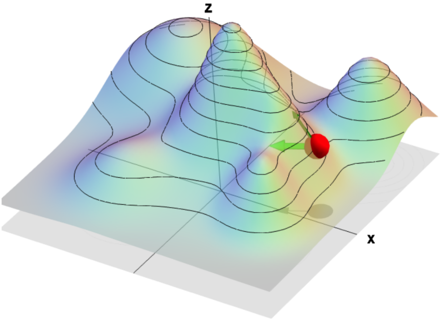
First-order algorithms that use f and ∇f. Standard algorithms when f is differentiable and convex.

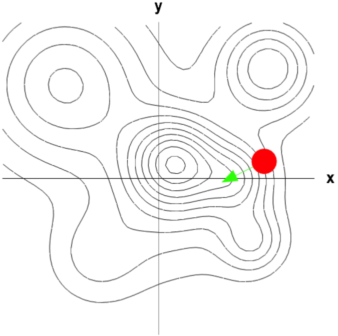
Second-order algorithms that use $f, \nabla f$ and $\nabla^2 f$. They are useful when computing the Hessian matrix is not too costly.

First fundamental characteristic of algorithms.

Gradient - Level sets

The gradient is orthogonal to level sets.





Reminder: Taylor expansion around a point

$$f(w) = f(w^{(0)}) + \langle \nabla f(w^{(0)}), w - w^{(0)} \rangle + O(\|w - w^{(0)}\|^2).$$

Gradient descent algorithm

Gradient descent

Input: Function f to minimize.

Initialization: initial weight vector $w^{(0)}$

Parameters: step size $\eta > 0$.

While *not converge* do

- $w^{(k+1)} \leftarrow w^{(k)} - \eta \nabla f(w^{(k)})$
- $k \leftarrow k + 1$.

Output: $w^{(k)}$.

Gradient Descent on a convex function

For a function $f : \mathbb{R}^d \to \mathbb{R}$, define the level sets:

$$\mathcal{C}_c = \{w \in \mathbb{R}^d, f(w) = c\}.$$

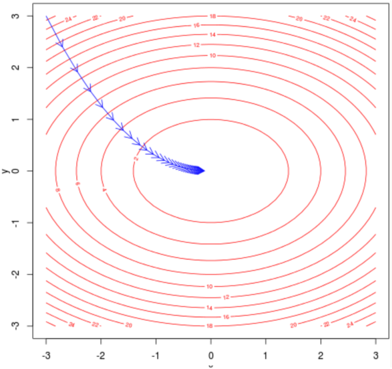


Figure: Gradient descent for function $f : (x, y) \mapsto x^2 + 2y^2$

Gradient Descent on a Bad objective functions

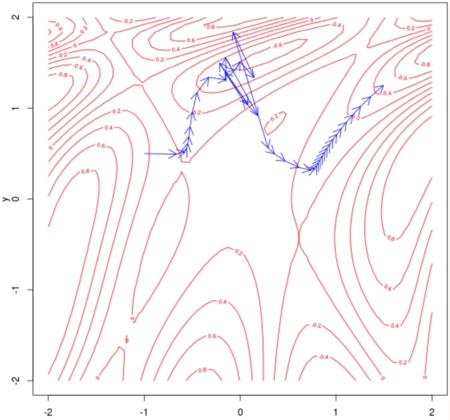


Figure: Gradient descent for $f : (x, y) \mapsto \mathrm{sinks}(1/(2x^2) - 1/(4y^2) + 3)\cos(2x + 1 - \exp(y))$

http://yulijia.net/vistat/2013/03/gradient-descent-algorithm-with-r

When does gradient descent converge?

Informal statement: GD converges, for a correct choice of steps, for most convex functions.

Why do we want convergence rates and proofs:

- Proofs help us choose hyperparameters (the learning rate sequence)
- Rates allow us to **compare algorithms**.

Today, we will see convergence results (without proofs) for :

1. GD and SGD
2. For convex and smooth functions, and smooth and strongly convex functions.

Thanks to those rates, we will be able to say in which situation GD or SGD should be preferred.

Formal definition: smoothness

L-smooth function

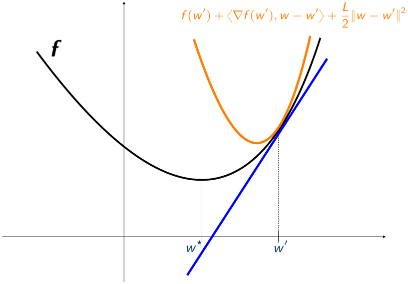
A function f is said to be L-smooth if f is differentiable and if, for all $x, y \in \mathbb{R}^d$,

$$\|\nabla f(x) - \nabla f(y)\| \leqslant L\|x - y\|.$$

Equivalently,

$$f(w) \leqslant f(w') + \langle \nabla f(w'), w - w' \rangle + \frac{L}{2}\|w - w'\|^2 \quad (1)$$

Smooth-convex: the function above the tangent and below the tangent line + quadratic:



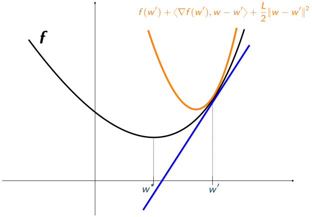
Co-coercivity: $\|\nabla f'(w) - \nabla f'(w')\|^2 \leqslant L\langle \nabla f(w') - \nabla f(w'), w - w' \rangle$

Interpretation of GD in the smooth case

Assuming the descent Lemma holds, remark that

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{L}{2} \|w - w^k\|_2^2 \right\}$$

$$= \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \left\| w - \left(w^k - \frac{1}{L} \nabla f(w^k) \right) \right\|_2^2$$

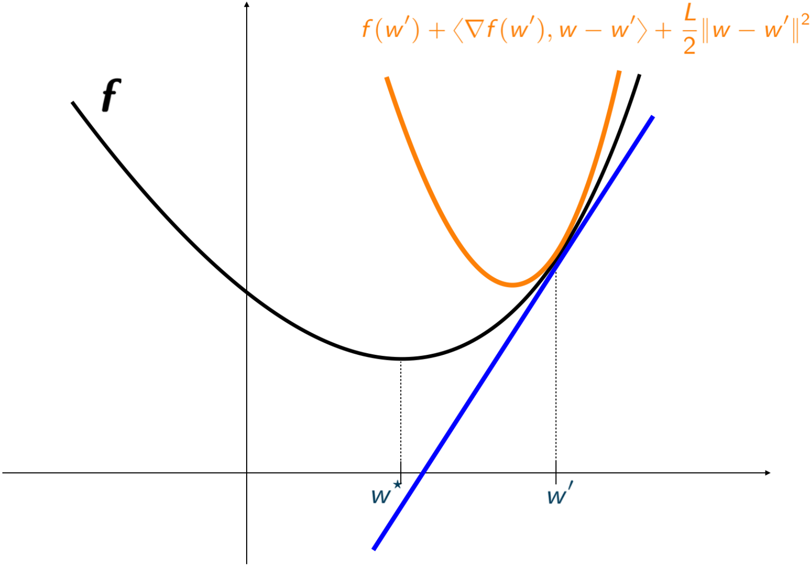


Hence, it is natural to choose

$$w^{k+1} = w^k - \frac{1}{L} \nabla f(w^k)$$

This is the basic **gradient descent** algorithm

Interpretation of GD in the smooth case



Convergence of GD

Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $w^\star$ be the minimum of f on $\mathbb{R}^d$. Then, Gradient Descent with step size $\eta \leqslant 1/L$ satisfies

$$f(w^{(k)}) - f(w^\star) \leqslant \frac{\|w^{(0)} - w^\star\|_2^2}{2\eta k}.$$

In particular, for $\eta = 1/L$,

$$L\|w^{(0)} - w^\star\|_2^2/2$$

iterations are sufficient to get an ε-approximation of the minimal value of f.

Faster rate for strongly convex function

Strong convexity: function above the tangent line + $\mu\times$ quadratic.

A function $f : \mathbb{R}^d \to R$ is μ-strongly convex if

$$w \mapsto f(w) - \frac{\mu}{2}\|w\|_2^2$$

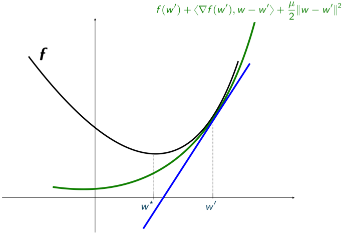
is convex.

If f is differentiable it is equivalent to writing, for all $w \in \mathbb{R}^d$,

$$\lambda_{min}(\nabla^2 f(w)) \geqslant \mu.$$

This is also equivalent to, for all $w, w' \in \mathbb{R}^d$:

$$f(w) \geqslant f(w') + \langle \nabla f(w'), w - w' \rangle + \frac{\mu}{2}\|w - w'\|^2$$



Useful inequality in the proofs:

$$\langle \nabla f'(w') - \nabla f'(w), w' - w \rangle \geqslant \mu\|w - w'\|^2 \quad (2)$$

Convergence of GD with strong convexity

Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth, μ strongly convex function. Let $w^\star$ be the minimum of f on $\mathbb{R}^d$. Then, Gradient Descent with step size $\eta \leqslant 1/L$ satisfies

$$f(w^{(k)}) - f(w^\star) \leqslant \frac{L}{2}\Big(1 - \eta\mu\Big)^k \|w^{(0)} - w^\star\|_2^2.$$

Condition number

Gradient descent uses iterations

$$w^{(k+1)} \leftarrow w^{(k)} - \eta \nabla f(w^{(k)})$$

- For L smooth convex function and $\eta = 1/L$,

$$f(w^{(k)}) - f(w^\star) \leqslant \frac{L\|w^{(0)} - w^\star\|_2^2}{2k}.$$

- For L smooth, μ strongly convex function and $\eta = 1/L$,

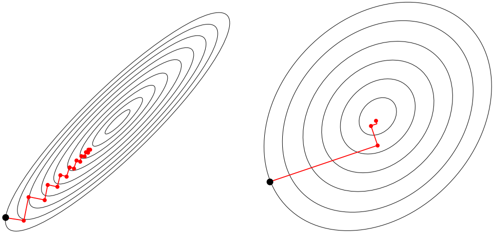
$$f(w^{(k)}) - f(w^\star) \leqslant \left(1 - \frac{\mu}{L}\right)^k \|f(w^{(0)}) - f(w^\star)\|_2^2.$$

Condition number $\kappa = L/\mu \geqslant 1$ stands for the difficulty of the learning problem.

Convergence vs condition number

Why?

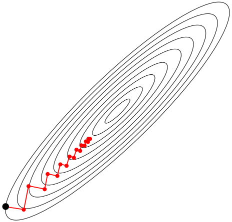
Rates typically depend on the condition number $\kappa = \frac{L}{\mu}$:



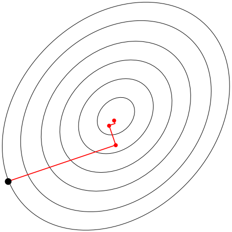
Convergence vs condition number

Why?

Rates typically depend on the condition number $\kappa = \frac{L}{\mu}$:



Large κ
harder to optimize



Small κ
easier to optimize

Full gradients...

We say that these methods are based on **full gradients**, since at each iteration we need to compute

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

which depends on the whole dataset

Question. If n is large, computing $\nabla f(w)$ is long: need to pass on the whole data before doing a step towards the minimum!

Idea. Large datasets make your modern computer look old

Go back to "old" algorithms.

Stochastic Gradient Descent (SGD)

Stochastic gradients

If I choose uniformly at random $I \in \{1, \ldots, n\}$, then

$$\mathbb{E}[\nabla f_I(w)] = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w)$$

$\nabla f_I(w)$ is an **unbiased** but very noisy estimate of the full gradient $\nabla f(w)$

Computation of $\nabla f_I(w)$ only requires the I-th line of data

$\rightarrow O(d)$ and smaller for sparse data

Crucial Balance:

- Noise
- Initial Condition

Impact of the learning rate?

Stochastic Gradient Descent (SGD)

[**robbins1985stochastic robbins1985stochastic**]

Stochastic gradient descent algorithm

Initialization: initial weight vector $w^{(0)}$,

Parameter: step size/learning rate η_k

For $k = 1, 2, \ldots$ until *convergence* do

- Pick at random (uniformly) i_k in $\{1, \ldots, n\}$
- Compute

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla f_{i_k}(w^{(k-1)})$$

Output: Return last $w^{(k)}$

Remarks

- Each iteration has complexity $O(d)$ instead of $O(nd)$ for full gradient methods
- Possible to reduce this to $O(s)$ when features are s-sparse using **lazy-updates**.

Convergence rate of SGD

Consider the stochastic gradient descent algorithm introduced previously but where each iteration is projected into the ball $B(0, R)$ with $R > 0$ fixed.

Let

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

Theorem

Assume that f is convex and that there exists $b > 0$ satisfying, for all $x \in B(0, R)$,

$$\|\nabla f_i(x)\| \leqslant b.$$

Besides, assume that all minima of f belong to $B(0, R)$. Then, setting $\eta_k = 2R/(b\sqrt{k})$,

$$\mathbb{E}\left[f\left(\frac{1}{k} \sum_{t=1}^{k} w^{(t)} \right) \right] - f(w^\star) \leqslant \frac{3Rb}{\sqrt{k}}$$

Convergence rate of SGD

Consider the stochastic gradient descent algorithm introduced previously but where each iteration is projected into the ball $B(0, R)$ with $R > 0$ fixed.

Let

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

Theorem

Assume that f is μ strongly convex and that there exists $b > 0$ satisfying, for all $x \in B(0, R)$,

$$\|\nabla f_i(x)\| \leqslant b.$$

Besides, assume that all minima of f belong to $B(0, R)$. Then, setting $\eta_k = 2/(\mu(k+1))$,

$$\mathbb{E}\left[f\left(\frac{2}{k(k+1)} \sum_{t=1}^{k} t \; w^{(t-1)}\right)\right] - f(w^\star) \leqslant \frac{2b^2}{\mu(k+1)}.$$

Comparison of GD and SGD

Full gradient descent

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_k \Big(\frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w^{(k)}) \Big)$$

- $O(nd)$ iterations
- Upper bound $O((1-(\mu/L))^k)$
- Numerical complexity $O(n\frac{L}{\mu}\log(\frac{1}{\varepsilon}))$

Stochastic gradient descent

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_k \nabla f_{i_k}(w^{(k)}).$$

- $O(d)$ iterations
- Upper bound $O(1/(\mu k))$
- Numerical complexity $O(\frac{1}{\mu\varepsilon})$

It does not depend on n for SGD !

Comparison GD versus SGD

Under strong convexity, GD versus SGD is

$$O\left(\frac{nL}{\mu}\log\left(\frac{1}{\varepsilon}\right)\right) \quad \text{versus} \quad O\left(\frac{1}{\mu\varepsilon}\right)$$

GD leads to a more accurate solution, but what if n is very large?

Recipe

- SGD is extremely fast in the early iterations (first two passes on the data)
- But it fails to converge accurately to the minimum

Beyond SGD

- Bottou and LeCun (2005),
- Shalev-Shwartz et al (2007, 2009),
- Nesterov et al. (2008, 2009),
- Bach et al. (2011, 2012, 2014, 2015),
- T. Zhang et al. (2014, 2015).

Summary of the first part

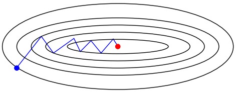
Convergence rates for GD and SGD: no universal algorithm !

Convergence rates for smooth functions (see previous slides for model and learning rate):

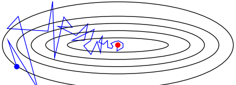
$\min \hat{\mathcal{R}}$

	SGD	GD
Convex	$O\left(\frac{1}{\sqrt{k}}\right)$	$O\left(\frac{1}{k}\right)$
Stgly-Cvx	$O\left(\frac{1}{\mu k}\right)$	$O(e^{-\mu k})$

- **Batch** gradient descent: $w_t = w_{t-1} - \eta_t f'(w_{t-1}) = w_{t-1} - \frac{\eta_t}{n} \sum_{i=1}^{n} f_i'(w_{t-1})$



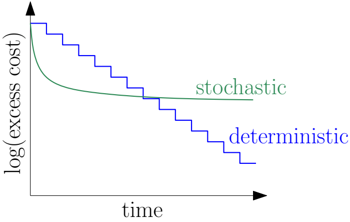
- **Stochastic** gradient descent: $w_t = w_{t-1} - \eta_t f'_{i(t)}(w_{t-1})$



Comparison of convergence : SGD vs GD

Which one to choose?

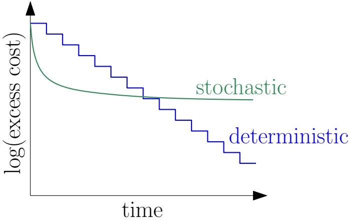
1. Depends on the precision we want.



Comparison of convergence : SGD vs GD

Which one to choose?

1. Depends on the precision we want.



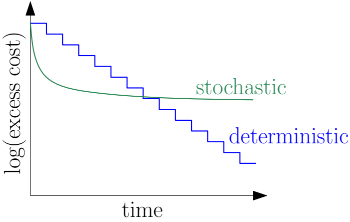
Example: non strongly convex case.

2. If our goal is to get a convergence of $1/\sqrt{n}$, then
    - Complexity of GD: $n^{3/2}d$
    - Complexity of SGD: nd.

Comparison of convergence : SGD vs GD

Which one to choose?

1. Depends on the precision we want.



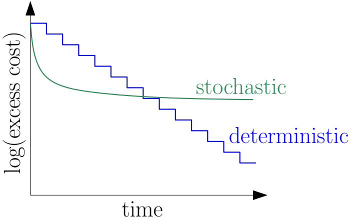
Example: non strongly convex case.

2. If our goal is to get a convergence of $1/\sqrt{n}$, then
    - Complexity of GD: $n^{3/2}d$
    - Complexity of SGD: nd.
3. If our goal is to get a convergence of $1/n^2$, then
    - Complexity of GD: $n^3 d$ (n^2 iterations)
    - Complexity of SGD: $n^4 d$ (n^4 iterations).

Comparison of convergence : SGD vs GD

Which one to choose?

1. Depends on the precision we want.



Example: non strongly convex case.

2. If our goal is to get a convergence of $1/\sqrt{n}$, then
   - Complexity of GD: $n^{3/2}d$
   - Complexity of SGD: nd.
3. If our goal is to get a convergence of $1/n^2$, then
   - Complexity of GD: $n^3 d$ (n^2 iterations)
   - Complexity of SGD: $n^4 d$ (n^4 iterations).

	Cplxty/step	Best Cplxty, low precision	Best Cplxty, high precision
GD	nd		✓
SGD	d	✓	

SGD vs GD

Recipe

- SGD is extremely fast in the early iterations (first two passes on the data)
- But it fails to converge accurately to the minimum

Machine Learning $\Rightarrow$ Low complexity is **often** enough !

Indeed,

- the minimization of the empirical risk is mostly a surrogate for the unknown generalization risk.
- no need to optimize below statistical error

Outline

Improving stochastic gradient descent

Goal: best of both worlds
The problem

- Let $X = \nabla f_I(w)$ with I uniformly chosen at random in $\{1, \ldots, n\}$
- In SGD we use $X = \nabla f_I(w)$ as an approximation of $\mathbb{E}X = \nabla f(w)$
- How to reduce $\mathbf{V}X$?

Improving stochastic gradient descent

An idea

- Reduce it by finding C s.t. $\mathbb{E}C$ is “easy” to compute and such that C is highly correlated with X
- Let $Z_\alpha = \alpha(X - C) + \mathbb{E}C$ for $\alpha \in [0, 1]$. We have

  $$\mathbb{E}Z_\alpha = \alpha\mathbb{E}X + (1 - \alpha)\mathbb{E}C$$

  and

  $$\mathbf{V}Z_\alpha = \alpha^2(\mathbf{V}X + \mathbf{V}C - 2\mathbb{C}(X, C))$$

- Standard variance reduction: $\alpha = 1$, so that $\mathbb{E}Z_\alpha = \mathbb{E}X$ (unbiased)

Improving stochastic gradient descent

Variance reduction of the gradient

In the iterations of SGD, replace $\nabla f_{i_k}(w^{(k-1)})$ by

$$\alpha(\nabla f_{i_k}(w^{(k-1)}) - \nabla f_{i_k}(\tilde{w})) + \nabla f(\tilde{w})$$

where $\tilde{w}$ is an "old" value of the iterate.

Several cases

- $\alpha = 1/n$: SAG (Bach et al. 2013)
- $\alpha = 1$: SVRG (T. Zhang et al. 2015, 2015)
- $\alpha = 1$: SAGA (Bach et al., 2014)

Important remark

- In these algorithms, the step-size η is kept **constant**
- Leads to **linearly convergent algorithms**, with a numerical complexity comparable to SGD!

Methods for finite sum minimization

- **GD**: at step k, use $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_k)$

Methods for finite sum minimization

- **GD**: at step k, use $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_k)$
- **SGD**: at step k, sample $i_k \sim \mathcal{U}[1; n]$, use $\nabla f_{i_k}(w_k)$

Methods for finite sum minimization

- **GD**: at step k, use $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_k)$
- **SGD**: at step k, sample $i_k \sim \mathcal{U}[1; n]$, use $\nabla f_{i_k}(w_k)$
- **SAG**: at step k,
  - keep a "full gradient" $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_{k_i})$, with $w_{k_i} \in \{w_1, \dots w_k\}$

Methods for finite sum minimization

- **GD**: at step k, use $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_k)$
- **SGD**: at step k, sample $i_k \sim \mathcal{U}[1;n]$, use $\nabla f_{i_k}(w_k)$
- **SAG**: at step k,
  - keep a "full gradient" $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_{k_i})$, with $w_{k_i} \in \{w_1, \ldots w_k\}$
  - sample $i_k \sim \mathcal{U}[1;n]$, use

$$\frac{1}{n}\left(\sum_{i=0}^{n} \nabla f_i(w_{k_i}) - \nabla f_{i_k}(w_{k_{i_k}}) + \nabla f_{i_k}(w_k)\right),$$

Methods for finite sum minimization

- **GD**: at step k, use $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_k)$
- **SGD**: at step k, sample $i_k \sim \mathcal{U}[1;n]$, use $\nabla f_{i_k}(w_k)$
- **SAG**: at step k,
  - keep a "full gradient" $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_{k_i})$, with $w_{k_i} \in \{w_1, \ldots w_k\}$
  - sample $i_k \sim \mathcal{U}[1;n]$, use

$$\frac{1}{n}\left(\sum_{i=0}^{n} \nabla f_i(w_{k_i}) - \nabla f_{i_k}(w_{k_{i_k}}) + \nabla f_{i_k}(w_k)\right),$$

Methods for finite sum minimization

- **GD**: at step k, use $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_k)$
- **SGD**: at step k, sample $i_k \sim \mathcal{U}[1;n]$, use $\nabla f_{i_k}(w_k)$
- **SAG**: at step k,
  - keep a "full gradient" $\frac{1}{n}\sum_{i=0}^{n} \nabla f_i(w_{k_i})$, with $w_{k_i} \in \{w_1, \dots w_k\}$
  - sample $i_k \sim \mathcal{U}[1;n]$, use

$$\frac{1}{n}\left(\sum_{i=0}^{n} \nabla f_i(w_{k_i}) - \nabla f_{i_k}(w_{k_{i_k}}) + \nabla f_{i_k}(w_k)\right),$$

In other words:

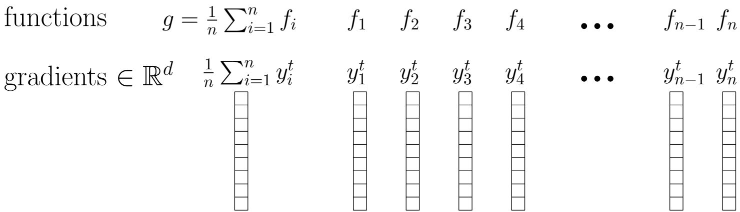
- Keep in memory past gradients of all functions f_i, $i = 1, \dots, n$
- Random selection $i_k \in \{1, \dots, n\}$ with replacement
- Iteration: $w_k = w_{k-1} - \frac{\eta}{n}\sum_{i=1}^{n} g_k(i)$ with $g_k(i) = \begin{cases} \nabla f_i(w_{k-1}) & \text{if } i = i_k \\ g_{k-1}(i) & \text{otherwise} \end{cases}$

SAG

- Keep in memory past gradients of all functions f_i, $i = 1, \dots, n$
- Random selection $i_k \in \{1, \dots, n\}$ with replacement
- Iteration: $w_k = w_{k-1} - \frac{\eta}{n} \sum_{i=1}^{n} g_k(i)$ with $g_k(i) = \begin{cases} \nabla f_i(w_{k-1}) & \text{if } i = i_k \\ g_{k-1}(i) & \text{otherwise} \end{cases}$

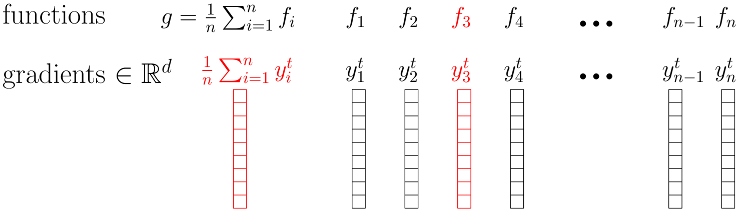
SAG

- Keep in memory past gradients of all functions f_i, $i = 1, \ldots, n$
- Random selection $i_k \in \{1, \ldots, n\}$ with replacement
- Iteration: $w_k = w_{k-1} - \frac{\eta}{n} \sum_{i=1}^{n} g_k(i)$ with $g_k(i) = \begin{cases} \nabla f_i(w_{k-1}) & \text{if } i = i_k \\ g_{k-1}(i) & \text{otherwise} \end{cases}$



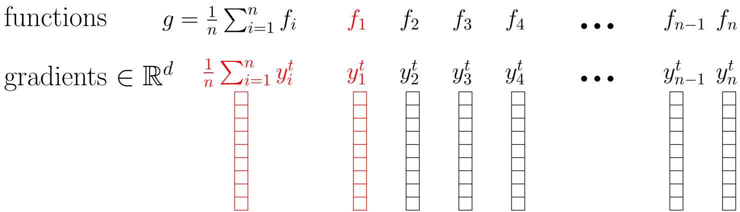
SAG

- Keep in memory past gradients of all functions f_i, $i = 1, \dots, n$
- Random selection $i_k \in \{1, \dots, n\}$ with replacement
- Iteration: $w_k = w_{k-1} - \frac{\eta}{n} \sum_{i=1}^{n} g_k(i)$ with $g_k(i) = \begin{cases} \nabla f_i(w_{k-1}) & \text{if } i = i_k \\ g_{k-1}(i) & \text{otherwise} \end{cases}$



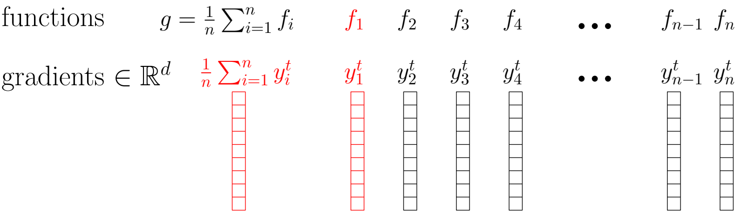
SAG

- Keep in memory past gradients of all functions f_i, $i = 1, \ldots, n$
- Random selection $i_k \in \{1, \ldots, n\}$ with replacement
- Iteration: $w_k = w_{k-1} - \frac{\eta}{n} \sum_{i=1}^{n} g_k(i)$ with $g_k(i) = \begin{cases} \nabla f_i(w_{k-1}) & \text{if } i = i_k \\ g_{k-1}(i) & \text{otherwise} \end{cases}$



SAG

- Keep in memory past gradients of all functions f_i, $i = 1, \dots, n$
- Random selection $i_k \in \{1, \dots, n\}$ with replacement
- Iteration: $w_k = w_{k-1} - \frac{\eta}{n} \sum_{i=1}^{n} g_k(i)$ with $g_k(i) = \begin{cases} \nabla f_i(w_{k-1}) & \text{if } i = i_k \\ g_{k-1}(i) & \text{otherwise} \end{cases}$



↬ ⊕ update costs the same as SGD
↬ ⊖ needs to store all gradients $\nabla f_i(w_{k_i})$ at “points in the past”

Improving stochastic gradient descent

Stochastic Average Gradient

Initialization: initial weight vector $w^{(0)}$

Parameter: learning rate $\eta > 0$

For $k = 1, 2, \ldots$ until *convergence* do

- Pick uniformly at random i_k in $\{1, \ldots, n\}$
- Put

$$g_k(i) = \begin{cases} \nabla f_i(w^{(k-1)}) & \text{if } i = i_k \\ g_{k-1}(i) & \text{otherwise} \end{cases}$$

- Compute

$$w^{(k)} = w^{(k-1)} - \eta\left(\frac{1}{n}\sum_{i=1}^{n} g_k(i)\right)$$

Output: Return last $w^{(k)}$

Improving stochastic gradient descent

Stochastic Variance Reduced Gradient (SVRG)

Initialization: initial weight vector $\tilde{w}$

Parameters: learning rate $\eta > 0$, phase size (typically $m = n$ or $m = 2n$).

For $k = 1, 2, \ldots$ until *convergence* do

- Compute $\nabla f(\tilde{w})$
- Put $w^{(0)} \leftarrow \tilde{w}$
- For $t = 1, \ldots, m$
  - Pick uniformly at random i_t in $\{1, \ldots, n\}$
  - Apply the step

$$w^{(t+1)} \leftarrow w^{(t)} - \eta(\nabla f_{i_t}(w^{(t)}) - \nabla f_{i_t}(\tilde{w}) + \nabla f(\tilde{w}))$$

- Set

$$\tilde{w} \leftarrow \frac{1}{m} \sum_{t=1}^{m} w^{(t)}$$

Output: Return $\tilde{w}$.

Improving stochastic gradient descent

SAGA

Initialization: initial weight vector $w^{(0)}$

Parameter: learning rate $\eta > 0$

For all $i = 1, \dots, n$, compute $g_0(i) \leftarrow \nabla f_i(w^{(0)})$

For $k = 1, 2, \dots$ until *convergence* do

- Pick uniformly at random i_k in $\{1, \dots, n\}$
- Compute $\nabla f_{i_k}(w^{(k-1)})$
- Apply

$$w^{(k)} \leftarrow w^{(k-1)} - \eta\Big(\nabla f_{i_k}(w^{(k-1)}) - g_{k-1}(i_k) + \frac{1}{n}\sum_{i=1}^{n} g_{k-1}(i)\Big)$$

- Store $g_k(i_k) \leftarrow \nabla f_{i_k}(w^{(k-1)})$

Output: Return last $w^{(k)}$

Variance reduced methods

Some references:

- SAG **Sch_LeR_Bac_2013** SAGA **Def_Bac_Lac_2014**
- SVRG **Joh_Zha_2013** (reduces memory cost but 2 epochs...)
- FINITO **Def_Dom_Cae_2014**
- S2GD **Kon_Ric_2013**..

And many others... See for example Niao He's lecture notes for a nice overview.

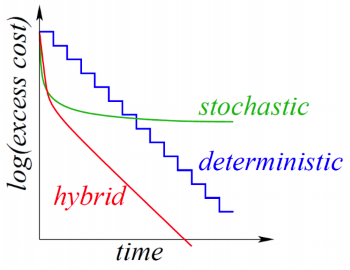
Convergence rate for $f(\tilde{w}_k) - f(\theta_*)$, **smooth** objective f.

		$\min \hat{\mathcal{R}}$	
	SGD	GD	SAG
Convex	$O\left(\frac{1}{\sqrt{k}}\right)$	$O\left(\frac{1}{k}\right)$	

Convergence rate for $f(\tilde{w}_k) - f(\theta_*)$, **smooth** objective f.

min $\hat{\mathcal{R}}$

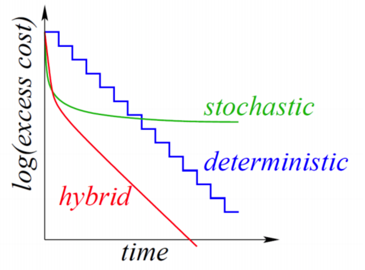
	SGD	GD	SAG
Convex	$O\left(\frac{1}{\sqrt{k}}\right)$	$O\left(\frac{1}{k}\right)$	
Stgly-Cvx	$O\left(\frac{1}{\mu k}\right)$	$O(e^{-\mu k})$	$O\left(1 - (\mu \wedge \frac{1}{n})\right)^k$



GD, SGD, SAG (Fig. from **Sch_LeR_Bac_2013**)

Convergence rate for $f(\tilde{w}_k) - f(\theta_*)$, **smooth** objective f.

		$\min \hat{\mathcal{R}}$	
	SGD	GD	SAG
Convex	$O\left(\frac{1}{\sqrt{k}}\right)$	$O\left(\frac{1}{k}\right)$	
Stgly-Cvx	$O\left(\frac{1}{\mu k}\right)$	$O(e^{-\mu k})$	$O\left(1 - (\mu \wedge \frac{1}{n})\right)^k$



GD, SGD, SAG (Fig. from **Sch_LeR_Bac_2013**)

Remarks:

- Proof technique
- Related to control variates in Federated Learning (*Scaffold*, *DIANA*, etc.)!

Summary

1. Variance reduced algorithms can have both:
    - low iteration cost
    - fast asymptotic convergence

However:

1. High precision is not always useful
2. Typically not used in deep learning:
    - Memory constraints for SAG
    - Convergence to "bad" (?) minima $\Rightarrow$ bad generalization...

Bad generalization in Deep Learning

Reasoning:

1. There are 2 types of local minima: flat and sharp.
2. Algorithm that converge tp “high precision” may converge to sharper minima.

Bad generalization in Deep Learning

Reasoning:

1. There are 2 types of local minima: flat and sharp.
2. Algorithm that converge tp “high precision” may converge to sharper minima.
3. Sharp minima have poorer generalization performance.

Challenges in Deep Learning

Challenges

1. Non convex $\Rightarrow$ Local minima
2. Extremely large dimension
3. Extremely large number of parameters (+ different scales)
4. Bad conditioning + flat areas + saddle points

Ingredients of popular algorithms:

1. First order
2. Stochastic
3. Momentum
4. Different steps per coordinates : adaptive methods

Challenges in Deep Learning

Challenges

1. Non convex $\Rightarrow$ Local minima
2. Extremely large dimension
3. Extremely large number of parameters (+ different scales)
4. Bad conditioning + flat areas + saddle points

Ingredients of popular algorithms:

1. First order
2. Stochastic
3. Momentum
4. Different steps per coordinates : adaptive methods

Generalization and overfitting problems are poorly understood but:

1. Noise helps
2. "Too precise" methods (e.g. variance reduction, second order) are not used. e.g.: SVRG is great for convex, but not even implemented in Keras.

Adaptation: notations

1. Same learning rate for all coordinates. Could we use a different learning rate for all coordinates ?
   i.e., for $1 \leqslant j \leqslant d$:
   $$(w^k)_j = (w^{k-1})_j - \eta_{k,j}(\nabla f_k(w^{k-1}))_j$$

   Equivalently:
   $$w^k = w^{k-1} - \begin{pmatrix} \eta_{k,1} \\ \eta_{k,2} \\ \dots \\ \eta_{k,d} \end{pmatrix} \odot \begin{pmatrix} (\nabla f_k(w^{k-1}))_1 \\ (\nabla f_k(w^{k-1}))_2 \\ \dots \\ (\nabla f_k(w^{k-1}))_d \end{pmatrix}$$
2. Indexes:
   $$(w_t)_j = (w_{k-1})_j - \eta_{k,k}(\nabla f_{I_k}(w_{k-1}))_j$$
   1. $g_k = \nabla f_{I_k}(w_{k-1})$ stochastic gradient at time t
      $$(w_k)_j = (w_{k-1})_j - \eta_{k,j}(g_k)_j$$
   2. Avoiding double subscript:
      $$(w^k)_j = (w^{k-1})_j - \eta^k_j(g^k)_j$$
      $$w^k_j = w^{k-1}_j - \eta^k_j g^t_j$$

ADAGRAD

Most following algos are in the following framework: First order method.

$$w_j^k = w_j^{k-1} - \eta_j^k g_j^k + (momentum)$$

Special choice for step-sizes:

$$w_j^k = w_j^{k-1} - \frac{\eta}{\sqrt{C_{k,j} + \epsilon}} g_j^k$$

[**duchi2011adaptive duchi2011adaptive**]

ADAptive GRADient algorithm

Initialization: initial weight vector $w^{(0)}$

Parameter: learning rate $\eta > 0$

For $k = 1, 2, \ldots$ until *convergence* do, component-wise.

- For all $j = 1, \ldots, d$,

$$w_j^k \leftarrow w_j^{k-1} - \frac{\eta}{\sqrt{\sum_{\tau=1}^k (g_j^\tau)^2 + \epsilon}} g_j^k$$

- Equivalently

$$w^k \leftarrow \tilde{w}^{(k-1)} - \frac{\eta}{\sqrt{\sum_{\tau=1}^k (\nabla f_{i_\tau}(w^{(\tau-1)}))^2 + \epsilon}} \odot g^k$$

Output: Return last $w^{(k)}$

ADAGRAD

Update equation for ADAGRAD

$$w^k \leftarrow \tilde{w}^{(k-1)} - \frac{\eta}{\sqrt{\sum_{t=1}^{k}(g_j^{\tau})^2 + \epsilon}} \odot g^k$$

Pros:

- Different dynamic rates on each coordinate
- Dynamic rates grow as the inverse of the gradient magnitude:
    1. Large/small gradients have small/large learning rates
    2. The dynamic over each dimension tends to be of the same order
    3. Interesting for neural networks in which gradient at different layers can be of different order of magnitude.
- Accumulation of gradients in the denominator act as a decreasing learning rate.

Cons:

- Very sensitive to initial condition: large initial gradients lead to small learning rates.
- Can be fought by increasing the learning rate thus making the algorithm sensitive to the choice of the learning rate.

ADAGRAD - Summary of parameters

ADAGRAD:

$$w_j^k = w_j^{k-1} - \eta_j^k g_j^k + \beta(momentum)$$

Special choice for step-sizes:

$$w_j^k = w_j^{k-1} - \frac{\eta}{\sqrt{C_{k,j} + \epsilon}} g_j^k$$

ADAGRAD - Summary of parameters

ADAGRAD:

$$w_j^k = w_j^{k-1} - \eta_j^k g_j^k + \beta(momentum)$$

Special choice for step-sizes:

$$w_j^k = w_j^{k-1} - \frac{\eta}{\sqrt{C_{k,j} + \epsilon}} g_j^k$$

ADAptive GRADient algorithm

1. starting point w^0,
2. learning rate $\eta > 0$, (default value of 0.01)
3. momentum β, constant ε.

For $t = 1, 2, \ldots$ until *convergence* do for $1 \leqslant j \leqslant d$

$$w_j^k \leftarrow w_j^{k-1} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{k} (g_j^\tau)^2 + \epsilon}} g_j^k$$

Return last w^k

Improving upon AdaGrad: RMS-prop

Idea : restricts the window of accumulated past gradients to some limited size through moving average.

1. starting point w^0, constant ε,
2. new params : decay rate $\rho > 0$

Update:

$$w_j^{k+1} = w_j^k - \frac{\eta_j^k}{\sqrt{C_{j,k} + \varepsilon}} g_j^k$$

Adagrad:

1. $C_{j,k} = \sum_{\tau=1}^{k} (g_j^\tau)^2$
2. $\eta_j^k = \eta$

RMS prop:

1. $C_{j,k} = \rho C_j^{k-1} + (1-\rho)(g_j^k)^2$
2. $\eta_j^k = \eta$ constant.

RMSprop

Unpublished method, from the course of Geoff Hinton

`http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

RMSprop algorithm

Initialization: initial weight vector $w^{(0)}$

Parameters: learning rate $\eta > 0$ (default $\eta = 0.001$), decay rate ρ (default $\rho = 0.9$)

For $k = 1, 2, \ldots$ until *convergence* do

- First, compute the accumulated gradient

$$\overline{(\nabla f)^2}^{(k)} = \rho \overline{(\nabla f)^2}^{(k-1)} + (1 - \rho)(g^k)^2$$

- Compute

$$w^{(k)} \leftarrow w^{(k-1)} - \frac{\eta}{\sqrt{\overline{(\nabla f)^2}^{(k)} + \varepsilon}} \odot g^k$$

Output: Return last $w^{(k)}$

Improving upon AdaGrad & RMS prop: AdaDelta

Idea :RMS-prop + Second order style approach.
Less sensitivity to initial parameters.
Update:

$$w_j^{k+1} = w_j^k - \frac{\eta_j^k}{\sqrt{C_{j,k} + \varepsilon}} g_j^k$$

Adagrad:

1. $C_{j,k} = \sum_{\tau=1}^{t} (g_j^{\tau})^2$
2. $\eta_j^k = \eta$

RMS prop:

1. $C_{j,k} = \rho {C_j}^{k-1} + (1-\rho)(g_j^k)^2$
2. $\eta_j^k = \eta$ constant.

Adadelta:

1. $C_{j,k} = \rho {C_j}^{k-1} + (1-\rho)(g_j^k)^2$
2. η_j^k variable.

ADADELTA

Determining a good learning rate becomes more of an art than science for many problems.

M.D. Zeiler

Update equation for adadelta

$$w^{(k+1)} = w^{(k)} - \frac{\sqrt{\overline{(\Delta w)^2}^{(k-1)} + \varepsilon}}{\sqrt{\overline{(\nabla f)^2}^{(k)} + \varepsilon}} \odot g^k$$

Interpretation:

- The numerator keeps the size of the previous step in memory and enforce larger steps along directions in which large steps were made.
- The denominator keeps the size of the previous gradients in memory and acts as a decreasing learning rate. Weights are lower than in Adagrad due to the decay rate ρ.

Inspired by second order methods (unit invariance + Hessian approximation)

$$\Delta w \simeq (\nabla^2 f)^{-1} \nabla f.$$

Roughly,

$$\Delta w = \frac{\frac{\partial f}{\partial w}}{\frac{\partial^2 f}{\partial w^2}} \Leftrightarrow \frac{1}{\frac{\partial^2 f}{\partial w^2}} = \frac{\Delta w}{\frac{\partial f}{\partial w}}.$$

See also **zeiler2012adadelta**; **schaul2013no**

ADADELTA

AdaDelta algorithm

Initialization: initial weight vector $w^{(0)}$, $\overline{(\nabla f)^2}^0 = 0$, $\overline{(\Delta x)^2}^0 = 0$

Parameters: decay rate $\rho > 0$, constant ε,

For $k = 1, 2, \ldots$ until *convergence* do

- For all $j = 1, \ldots, d$,
  1. Compute the accumulated gradient
  $$\overline{(\nabla f)^2}^{(k)} = \rho \overline{(\nabla f)^2}^{(k-1)} + (1-\rho)(g^k)^2$$
  2. Compute the update
  $$w^{(k)} = w^{(k-1)} - \frac{\sqrt{\overline{(\Delta w)^2}^{(k-1)} + \varepsilon}}{\sqrt{\overline{(\nabla f)^2}^{(k)} + \varepsilon}} \odot g^k$$
  3. Compute the aggregated update
  $$\overline{(\Delta w)^2}^{(k)} = \rho \overline{(\Delta w)^2}^{(k-1)} + (1-\rho)(w^{(k+1)} - w^{(k)})^2$$

Output: Return last $w^{(k)}$

ADAM: ADAptive Moment estimation

[kingma2014adam kingma2014adam]
General idea: store the estimated first and second moment of the gradient and use them to update the parameters.

Equations - first and second moment

Let m_k be an exponentially decaying average over the past gradients

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g^k$$

Similarly, let v_t be an exponentially decaying average over the past square gradients

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2)(g^k)^2.$$

Initialization: $m_0 = v_0 = 0$.

With this initialization, estimates m_t and v_t are biased towards zero in the early steps of the gradient descent.

Final equations

$$\tilde{m}_k = \frac{m_k}{1 - \beta_1^k} \quad \tilde{v}_k = \frac{v_k}{1 - \beta_2^k}.$$

$$w^{(k)} = w^{(k-1)} - \frac{\eta}{\sqrt{\tilde{v}_k} + \varepsilon} \tilde{m}_k.$$

Adam algorithm

Initialization: $m_0 = 0$ (Initialization of the first moment vector), $v_0 = 0$ (Initialization of the second moment vector), w_0 (initial vector of parameters).

Parameters: stepsize η (default $\eta = 0.001$), exponential decay rates for the moment estimates $\beta_1, \beta_2 \in [0, 1)$ (default: $\beta_1 = 0.9$, $\beta_2 = 0.999$), numeric constant ε (default $\varepsilon = 10^{-8}$).

For $k = 1, 2, \ldots$ until *convergence* do

- Compute first and second moment estimate
$$m^{(k)} = \beta_1 m^{(k-1)} + (1 - \beta_1) g^k \quad v^{(k)} = \beta_2 v_{(k-1)} + (1 - \beta_2)(g^k)^2.$$
- Compute their respective correction
$$\tilde{m}^{(k)} = \frac{m^{(k)}}{1 - \beta_1^k} \quad \tilde{v}^{(k)} = \frac{v^{(k)}}{1 - \beta_2^k}.$$
- Update the parameters accordingly
$$w^{(k)} = w^{(k-1)} - \frac{\eta}{\sqrt{\tilde{v}^{(k)}} + \varepsilon} \odot \tilde{m}^{(k)}.$$

Output: Return last $w^{(k)}$

Convergence results: **[kingma2014adam kingma2014adam]**, **[reddi2018convergence reddi2018convergence]**.

Adamax algorithm

Initialization: $m_0 = 0$ (Initialization of the first moment vector), $u_0 = 0$ (Initialization of the exponentially weighted infinity norm), w_0 (initial vector of parameters).

Parameters: stepsize η (default $\eta = 0.001$), exponential decay rates for the moment estimates $\beta_1, \beta_2 \in [0, 1)$ (default: $\beta_1 = 0.9$, $\beta_2 = 0.999$)

For $k = 1, 2, \ldots$ until *convergence* do

- Compute first moment estimate and its correction

$$m^{(k)} = \beta_1 m_{(k-1)} + (1 - \beta_1) g^k, \qquad \tilde{m}^{(k)} = \frac{m^{(k)}}{1 - \beta_1^k}$$

- Compute the quantity

$$u^{(k)} = \max(\beta_2 u^{(k-1)}, |g^k|).$$

- Update the parameters accordingly

$$w^{(k+1)} = w^{(k)} - \frac{\eta}{u^{(k)}} \odot \tilde{m}^{(k)}.$$

Output: Return last $w^{(k)}$

[**kingma2014adam kingma2014adam**]

Animation of Stochastic Gradient algorithms

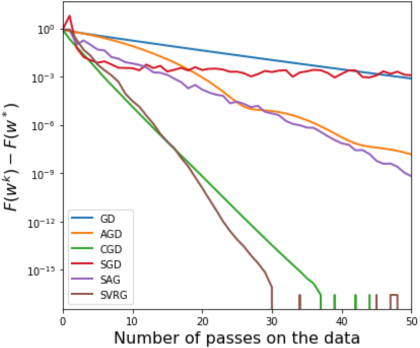
https://imgur.com/a/Hqolp Credits to Alec Radford for the animations.

The Notebook

Goal: Code

1. gradient descent (GD)
2. accelerated gradient descent (AGD)
3. coordinate gradient descent (CD)
4. stochastic gradient descent (SGD)
5. stochastic variance reduced gradient descent (SAG)
6. Adagrad

for the linear regression and logistic regression models, with the ridge penalization.



Summary

What we have seen so far !

- Why optimization is important, what makes it difficult
- Simple first order methods, from GD to SGD
- Advanced first order methods, variance reduction and coordinate adaptive step-sizes

What we have missed and won't cover

- Acceleration techniques (momentum, Nesterov)
- Second order methods
- Federated Learning algorithms.

What's next

- Statistical approach.

Outline

Supervised machine learning

- **Data**: n observations $(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \dots, n$, **i.i.d.**
- Prediction as a linear function $\langle \theta, \Phi(x) \rangle$ of features $\Phi(x) \in \mathbb{R}^d$
- **(regularized) empirical risk minimization**: find $\hat{\theta}$ solution of

$$\min_{\theta \in \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \ell\big(Y_i, \langle \theta, \Phi(X_i) \rangle\big) \quad + \quad \mu \Omega(\theta)$$

convex data fitting term + regularizer

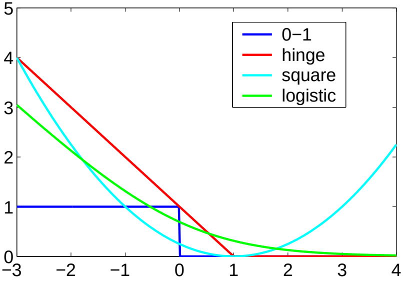
Usual losses

- **Regression**: $y \in \mathbb{R}$, prediction $\phi_\theta(x) = \langle \theta, \Phi(x) \rangle$
  - quadratic loss $\ell(y, \langle \theta, \Phi(x) \rangle) = \frac{1}{2}(y - \langle \theta, \Phi(x) \rangle)^2$

Usual losses

- **Regression**: $y \in \mathbb{R}$, prediction $\phi_\theta(x) = \langle \theta, \Phi(x) \rangle$
  - quadratic loss $\ell(y, \langle \theta, \Phi(x) \rangle) = \frac{1}{2}(y - \langle \theta, \Phi(x) \rangle)^2$
- **Classification** : $y \in \{-1, 1\}$, prediction $\phi_\theta(x) = \text{sign}(\langle \theta, \Phi(x) \rangle)$
  - $0 - 1$ loss: $\ell(y, \langle \theta, \Phi(x) \rangle) = \mathbf{1}_{\{y \cdot \langle \theta, \Phi(x) \rangle < 0\}}$.
  - **convex** losses

Convex loss



- **Support vector machine** (hinge loss)
  $$\ell(Y, \langle\theta, \Phi(x)\rangle) = \max\{1 - Y\langle\theta, \Phi(x)\rangle, 0\}$$

- **Logistic regression:**
  $$\ell(Y, \langle\theta, \Phi(x)\rangle) = \log(1 + \exp(-Y\langle\theta, \Phi(x)\rangle))$$

- **Least-squares regression**
  $$\ell(Y, \langle\theta, \Phi(x)\rangle) = \frac{1}{2}(Y - \langle\theta, \Phi(x)\rangle)^2$$

Usual regularizers

- **Main goal**: avoid overfitting
- **(squared) Euclidean norm**: $\|\theta\|_2^2 = \sum_{j=1}^{d} |\theta_j|^2$
- **Sparsity-inducing norms**
  - LASSO : ℓ_1-norm $\|\theta\|_1 = \sum_{j=1}^{d} |\theta_j|$
  - Perform model selection as well as regularization
  - Non-smooth optimization and structured sparsity
  - See, e.g., Bach, Jenatton, Mairal and Obozinski (2012a,b)

Supervised machine learning

- **Data**: n observations $(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \dots, n$, **i.i.d.**
- Prediction as a linear function $\langle \theta, \Phi(x) \rangle$ of features $\Phi(x) \in \mathbb{R}^d$
- **(regularized) empirical risk minimization**: find $\hat{\theta}$ solution of

$$\min_{\theta \in \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \ell\big(Y_i, \langle \theta, \Phi(X_i) \rangle\big) \text{ such that } \Omega(\theta) \leqslant D$$

convex data fitting term + constraint

Supervised machine learning

- **Data**: n observations $(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \dots, n$, **i.i.d.**
- Prediction as a linear function $\langle \theta, \Phi(x) \rangle$ of features $\Phi(x) \in \mathbb{R}^d$
- **(regularized) empirical risk minimization**: find $\hat{\theta}$ solution of

$$\min_{\theta \in \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \ell\big(Y_i, \langle \theta, \Phi(X_i) \rangle\big) \text{ such that } \Omega(\theta) \leqslant D$$

convex data fitting term + constraint

- **Empirical risk**: $\hat{f}(\theta) = n^{-1} \sum_{i=1}^{n} \ell(Y_i, \langle \theta, \Phi(X_i) \rangle)$

Supervised machine learning

- **Data**: n observations $(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \dots, n$, **i.i.d.**
- Prediction as a linear function $\langle \theta, \Phi(x) \rangle$ of features $\Phi(x) \in \mathbb{R}^d$
- **(regularized) empirical risk minimization**: find $\hat{\theta}$ solution of

$$\min_{\theta \in \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \ell\big(Y_i, \langle \theta, \Phi(X_i) \rangle\big) \text{ such that } \Omega(\theta) \leqslant D$$

convex data fitting term + constraint

- **Empirical risk**: $\hat{f}(\theta) = n^{-1} \sum_{i=1}^{n} \ell(Y_i, \langle \theta, \Phi(X_i) \rangle)$
- **Expected risk**: $f(\theta) = \mathbb{E}[\ell(Y, \langle \theta, \Phi(X) \rangle)]$.

General assumptions

- **Data**: n observations $(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \dots, n$, **i.i.d.**
- Bounded features $\Phi(x) \in \mathbb{R}^d$: $\|\Phi(x)\|_2 \leqslant R$
- **Empirical risk** $\hat{f}(\theta) = n^{-1} \sum_{i=1}^n \ell(Y_i, \langle \theta, \Phi(X_i) \rangle)$
- **Expected risk** $f(\theta) = \mathbb{E}[\ell(Y, \langle \theta, \Phi(X) \rangle)]$
- **Loss for a single observation**: $f_i(\theta) = \ell(Y_i, \langle \theta, \Phi(X_i) \rangle)$. For all i, $f(\theta) = \mathbb{E}[f_i(\theta)]$
- **Properties of** $f_i, f, \hat{f}$
  - Convex on $\mathbb{R}^d$
  - Additional regularity assumptions: Lipschitz-continuity, smoothness and strong convexity

Lipschitz continuity

- **Bounded gradients of g ($\Leftrightarrow$ Lipschitz-continuity)**: the function g if convex, differentiable and has gradients uniformly bounded by B on the ball of center 0 and radius D: for all $\theta \in \mathbb{R}^d$,

$$\|\theta\|_2 \leqslant D \Rightarrow \|\nabla g(\theta)\|_2 \leqslant B$$

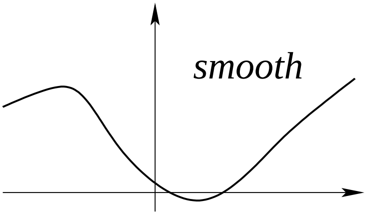
$$\Leftrightarrow$$

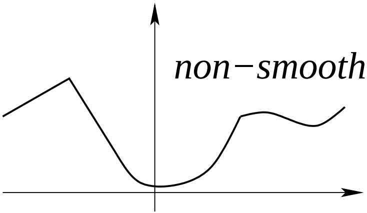
$$|g(\theta) - g(\theta')| \leqslant B\|\theta - \theta'\|_2$$

- **Machine learning**
  - $g(\theta) = n^{-1} \sum_{i=1}^{n} \ell(Y_i, \langle\theta, \Phi(X_i)\rangle)$
  - G-Lipschitz loss and R-bounded data: $B = GR$

Smoothness and strong convexity

- A function $g : \mathbb{R}^d \to \mathbb{R}$ is L-smooth if and only if it is differentiable and its gradient is L-Lipschitz: for all $\theta, \theta' \in \mathbb{R}^d$;
$$\|\nabla g(\theta_1) - \nabla g(\theta')\|_2 \leqslant L\|\theta - \theta'\|_2$$
- If g is twice differentiable, for all $\theta \in \mathbb{R}^d$, $\nabla^{\otimes 2} g(\theta) \preccurlyeq L \cdot \mathrm{Id}$





Smoothness and strong convexity

- A function $g : \mathbb{R}^d \to \mathbb{R}$ is L-smooth if and only if it is differentiable and its gradient is L-Lipschitz: for all $\theta, \theta' \in \mathbb{R}^d$;
$$\|\nabla g(\theta_1) - \nabla g(\theta')\|_2 \leqslant L \|\theta - \theta'\|_2$$

- If g is twice differentiable, for all $\theta \in \mathbb{R}^d$, $\nabla^{\otimes 2} g(\theta) \preccurlyeq L \cdot \mathrm{Id}$

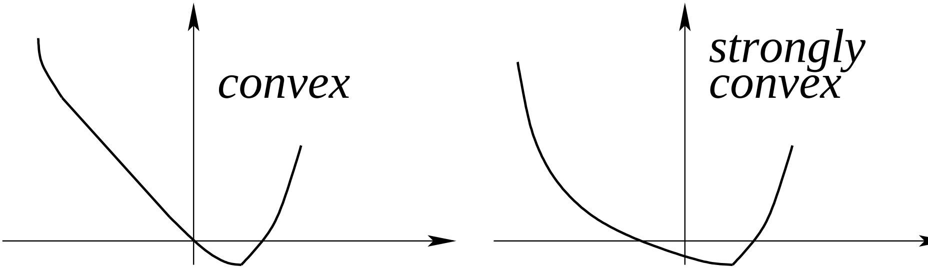
Machine learning

- $g(\theta) = n^{-1} \sum_{i=1}^n \ell(Y_i, \langle \theta, \Phi(X_i) \rangle)$
- Hessian $\approx$ covariance matrix
$$n^{-1} \sum_{i=1}^n \Phi(X_i) \Phi^\top (X_i) \ddot{\ell}(Y_i, \langle \theta, \Phi(X_i) \rangle)$$

- L_{loss}-smooth loss and R-bounded data: $L = L_{\text{loss}} R^2$

Smoothness and strong convexity

- A function $g : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly convex if and only if, for all $\theta, \theta' \in \mathbb{R}^d$,
$$g(\theta) \geqslant g(\theta') + \langle \nabla g(\theta'), \theta - \theta' \rangle + \frac{\mu}{2} \|\theta - \theta'\|_2^2$$
- If g is twice differentiable: for all $\theta \in \mathbb{R}^d$, $\nabla^2 g(\theta) \succcurlyeq \mu \cdot \mathrm{Id}$



Smoothness and strong convexity

- A function $g : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly convex if and only if, for all $\theta, \theta' \in \mathbb{R}^d$,
$$g(\theta) \geqslant g(\theta') + \langle \nabla g(\theta'), \theta - \theta' \rangle + \frac{\mu}{2} \|\theta - \theta'\|_2^2$$
- If g is twice differentiable: for all $\theta \in \mathbb{R}^d$, $\nabla^2 g(\theta) \succcurlyeq \mu \cdot \mathrm{Id}$

Machine learning

- $g(\theta) = n^{-1} \sum_{i=1}^n \ell(Y_i, \langle \theta, \Phi(X_i) \rangle)$
- Hessian $\approx$ covariance matrix
$$n^{-1} \sum_{i=1}^n \Phi(X_i) \Phi(X_i)^\top \ddot{\ell}(Y_i, \langle \theta, \Phi(X_i) \rangle)$$
- Data with invertible covariance matrix

Smoothness and strong convexity

- A function $g : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly convex if and only if, for all $\theta, \theta' \in \mathbb{R}^d$,

$$g(\theta) \geqslant g(\theta') + \langle \nabla g(\theta'), \theta - \theta' \rangle + \frac{\mu}{2} \|\theta - \theta'\|_2^2$$

- If g is twice differentiable: for all $\theta \in \mathbb{R}^d$, $\nabla^2 g(\theta) \succcurlyeq \mu \cdot \mathrm{Id}$

Machine learning

- $g(\theta) = n^{-1} \sum_{i=1}^n \ell(Y_i, \langle \theta, \Phi(X_i) \rangle)$
- Hessian $\approx$ covariance matrix

$$n^{-1} \sum_{i=1}^n \Phi(X_i)\Phi(X_i)^\top \ddot{\ell}(Y_i, \langle \theta, \Phi(X_i) \rangle)$$

- Data with invertible covariance matrix

Adding regularization by $\frac{\mu}{2}\|\theta\|^2$ [! creates a bias (controlled by μ)]

Smoothness/convexity assumptions: summary

- **Bounded gradients of g (Lipschitz-continuity)**: the function g if convex, differentiable and has gradients uniformly bounded by B on the ball of center 0 and radius D:

$$\text{for all } \theta \in \mathbb{R}^d, \|\theta\|_2 \leqslant D \Rightarrow \|\nabla g(\theta)\|_2 \leqslant B$$

- **Smoothness of g**: the function g is convex, differentiable with L-Lipschitz-continuous gradient ∇g:

$$\text{for all } \theta, \theta' \in \mathbb{R}^d, \quad \|\nabla g(\theta) - \nabla g(\theta')\|_2 \leqslant L\|\theta - \theta'\|_2$$

- **Strong convexity of g**: The function f is strongly convex with respect to the norm $\|\cdot\|_2$, with convexity constant $\mu > 0$: for all $\theta, \theta'` \in \mathbb{R}^d$,

$$g(\theta) \geqslant g(\theta') + \langle \nabla g(\theta'), \theta - \theta' \rangle + \frac{\mu}{2}\|\theta - \theta'\|_2^2$$

Empirical risk minimization: rationale

- The expected risk $f(\theta) = \mathbb{E}[\ell(Y, \langle\theta, X, \rangle\rangle]$ is not tractable.
- Only the empirical risk $\hat{f}(\theta) = n^{-1} \sum_{i=1}^{n}[\ell(Y_i, \langle\theta, X_i, \rangle\rangle]$ is.
- **Minimizing $\hat{f}$ instead of f?**
- A simple observation:

$$f(\hat{\theta}) - \min_{\theta\in\Theta} f(\theta) \leqslant \sup_{\theta\in\Theta}\{\hat{f}(\theta) - f(\theta)\} + \sup_{\theta\in\Theta}\{f(\theta) - \hat{f}(\theta)\}$$

- Can we have a bound on $\sup_{\theta\in\Theta} |\hat{f}(\theta) - f(\theta)|$?

Motivation from least-squares

- For least-squares, we have $\ell(y, \langle\theta, \Phi(x)\rangle) = \frac{1}{2}(y - \langle\theta, \Phi(x)\rangle)^2$, and

$$f(\theta) - \hat{f}(\theta) = \frac{1}{2}\theta^\top\Big(\frac{1}{n}\sum_{i=1}^n \Phi(X_i)\Phi(X_i)^\top - \mathbb{E}\Phi(X)\Phi(X)^\top\Big)\theta$$

$$- \theta^\top\Big(\frac{1}{n}\sum_{i=1}^n Y_i\Phi(X_i) - \mathbb{E}Y\Phi(X)\Big) + \frac{1}{2}\Big(\frac{1}{n}\sum_{i=1}^n Y_i^2 - \mathbb{E}Y^2\Big),$$

$$\sup_{\|\theta\|_2\leqslant D} |f(\theta) - \hat{f}(\theta)| \leqslant \frac{D^2}{2}\Big\|\frac{1}{n}\sum_{i=1}^n \Phi(X_i)\Phi(X_i)^\top - \mathbb{E}\Phi(X)\Phi(X)^\top\Big\|_{\rm op}$$

$$+ D\Big\|\frac{1}{n}\sum_{i=1}^n Y_i\Phi(X_i) - \mathbb{E}Y\Phi(X)\Big\|_2 + \frac{1}{2}\Big|\frac{1}{n}\sum_{i=1}^n Y_i^2 - \mathbb{E}Y^2\Big|,$$

$$\sup_{\|\theta\|_2\leqslant D} |f(\theta) - \hat{f}(\theta)| \leqslant O(1/\sqrt{n}) \text{ with high probability}$$

Slow rate for supervised learning

Assumptions (f is the expected risk, $\hat{f}$ the empirical risk)

- $\Omega(\theta) = \|\theta\|_2$ (Euclidean norm)
- "Linear" predictors: $\phi_\theta(x) = \langle \theta, \Phi(x) \rangle$, with $\|\Phi(x)\|_2 \leqslant R$
- G-Lipschitz loss: $f(\theta) = \ell(Y, \langle \theta, \Phi(X) \rangle)$ is GR-Lipschitz on $\Theta = \{\|\theta\|_2 \leqslant D\}$
- **No convexity assumption**

Slow rate for supervised learning

Assumptions (f is the expected risk, $\hat{f}$ the empirical risk)

- $\Omega(\theta) = \|\theta\|_2$ (Euclidean norm)
- "Linear" predictors: $\phi_\theta(x) = \langle \theta, \Phi(x) \rangle$, with $\|\Phi(x)\|_2 \leqslant R$
- G-Lipschitz loss: $f(\theta) = \ell(Y, \langle \theta, \Phi(X) \rangle)$ is GR-Lipschitz on $\Theta = \{\|\theta\|_2 \leqslant D\}$
- **No convexity assumption**

High-probability bounds: With probability greater than $1 - \delta$,

$$\sup_{\theta \in \Theta} |\hat{f}(\theta) - f(\theta)| \leqslant \frac{\sup |\ell(Y, 0)| + GRD}{\sqrt{n}} \left[2 + \sqrt{2 \log \frac{2}{\delta}} \right]$$

Slow rate for supervised learning

Assumptions (f is the expected risk, $\hat{f}$ the empirical risk)

- $\Omega(\theta) = \|\theta\|_2$ (Euclidean norm)
- "Linear" predictors: $\phi_\theta(x) = \langle \theta, \Phi(x) \rangle$, with $\|\Phi(x)\|_2 \leqslant R$
- G-Lipschitz loss: $f(\theta) = \ell(Y, \langle \theta, \Phi(X) \rangle)$ is GR-Lipschitz on $\Theta = \{\|\theta\|_2 \leqslant D\}$
- **No convexity assumption**

Risk bounds

$$\mathbb{E}\Big[\sup_{\theta \in \Theta} |\hat{f}(\theta) - f(\theta)|\Big] \leqslant \frac{4 \sup |\ell(Y, 0)| + 4GRD}{\sqrt{n}}$$

Slow rate for supervised learning

Assumptions (f is the expected risk, $\hat{f}$ the empirical risk)

- $\Omega(\theta) = \|\theta\|_2$ (Euclidean norm)
- "Linear" predictors: $\phi_\theta(x) = \langle \theta, \Phi(x) \rangle$, with $\|\Phi(x)\|_2 \leqslant R$
- G-Lipschitz loss: $f(\theta) = \ell(Y, \langle \theta, \Phi(X) \rangle)$ is GR-Lipschitz on $\Theta = \{\|\theta\|_2 \leqslant D\}$
- **No convexity assumption**

Method

- **Tools**: Symmetrization, Rademacher complexity (see Boucheron et al., 2012), McDiarmid inequality.
- **Lipschitz functions $\Rightarrow$ slow rate**

Symmetrization with Rademacher variables

- Let $\mathcal{D}' = \{X_1', Y_1', \dots, X_n', Y_n'\}$ an independent copy of the data $\mathcal{D} = \{X_1, Y_1, \dots, X_n, Y_n\}$, with corresponding loss functions $f_i'(\theta)$,

$$\mathbb{E}\left[\sup_{\theta\in\Theta}\left\{f(\theta) - \hat{f}(\theta)\right\}\right] = \mathbb{E}\left[\sup_{\theta\in\Theta}\left\{f(\theta) - \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)\right\}\right]$$

$$= \mathbb{E}\left[\sup_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[f_i'(\theta) - f_i(\theta)\middle|\mathcal{D}\right]\right]$$

$$\leqslant \mathbb{E}\left[\mathbb{E}\left[\sup_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\{f_i'(\theta) - f_i(\theta)\}\middle|\mathcal{D}\right]\right]$$

Symmetrization with Rademacher variables

- Let $\mathcal{D}' = \{X_1', Y_1', \ldots, X_n', Y_n'\}$ an independent copy of the data $\mathcal{D} = \{X_1, Y_1, \ldots, X_n, Y_n\}$, with corresponding loss functions $f_i'(\theta)$,

$$\mathbb{E}\left[\sup_{\theta\in\Theta}\left\{f(\theta) - \hat{f}(\theta)\right\}\right] = \mathbb{E}\left[\sup_{\theta\in\Theta}\left\{f(\theta) - \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)\right\}\right]$$

$$= \mathbb{E}\left[\sup_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\left\{f_i'(\theta) - f_i(\theta)\right\}\right]$$

$$= \mathbb{E}\left[\sup_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\varepsilon_i\left\{f_i'(\theta) - f_i(\theta)\right\}\right] \text{ with } \varepsilon_i \text{ uniform in } \{-1,1\}$$

$$\leqslant 2\mathbb{E}\left[\sup_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}\varepsilon_i f_i(\theta)\right] = \text{Rademacher complexity}$$

Rademacher complexity

- Define the **Rademacher complexity** of the class of functions $(x, y) \mapsto \ell(y, \langle\theta, \Phi(x)\rangle)$ as

$$R_n = \mathbb{E}\left[\sup_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} \varepsilon_i f_i(\theta)\right], \quad f_i(\theta) = \ell(Y_i, \langle\theta, \Phi(X_i)\rangle)$$

- **Main property**:

$$\mathbb{E}\left[\sup_{\theta\in\Theta}\left\{f(\theta) - \hat{f}(\theta)\right\}\right] = \mathbb{E}\left[\sup_{\theta\in\Theta}\left\{\hat{f}(\theta) - f(\theta)\right\}\right] \leqslant 2R_n$$

From Rademacher complexity to uniform bound

$$Z = \sup_{\theta \in \Theta} \{ f(\theta) - \hat{f}(\theta) \}$$
$$= \sup_{\theta \in \Theta} \left\{ f(\theta) - n^{-1} \sum_{i=1}^{n} \ell(Y_i, \langle \theta, \Phi(X_i) \rangle) \right\}$$

- By changing one pair (X_i, Y_i), Z may only change by
$$\frac{2}{n} \sup |\ell(Y, \langle \theta, \Phi(x) \rangle)| \leqslant \frac{2}{n} \big(\sup |\ell(Y, 0)| + GRD \big) \leqslant \frac{2}{n} \big(\ell_0 + GRD \big) = c$$
with $\sup |\ell(Y, 0)| = \ell_0$
- **MacDiarmid inequality**: with probability greater than $1 - \delta$,
$$Z \leqslant \mathbb{E}Z + \sqrt{\frac{n}{2}} c \cdot \sqrt{\log \frac{1}{\delta}} \leqslant 2R_n + \frac{\sqrt{2}}{\sqrt{n}} \big(\ell_0 + GRD \big) \sqrt{\log \frac{1}{\delta}}$$

Bounding the Rademacher average

- Empirical Rademacher averages

$$\hat{R}_n = \mathbb{E}\left[\sup_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i f_i(\theta) \middle| \mathbb{X} \right]$$

$$\leqslant \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} \varepsilon_i f_i(0) \middle| \mathbb{X} \right] + \mathbb{E}\left[\sup_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i \big[f_i(\theta) - f_i(0) \big] \middle| \mathbb{X} \right]$$

$$\leqslant 0 + \mathbb{E}\left[\sup_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i \big[f_i(\theta) - f_i(0) \big] \middle| \mathbb{X} \right]$$

$$= 0 + \mathbb{E}\left[\sup_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i \varphi_i(\langle \theta, \Phi(X_i) \rangle) \middle| \mathbb{X} \right]$$

- Using Ledoux-Talagrand concentration results for Rademacher averages (since φ_i is G-Lipschitz), we get:

$$\hat{R}_n \leqslant 2G \cdot \mathbb{E}\left[\sup_{\|\theta\|_2 \leqslant D} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i \langle \theta, \Phi(X_i) \rangle \middle| \mathbb{X} \right]$$

Bounding the Rademacher average - II

$$
\begin{aligned}
R_n &\leqslant 2G\mathbb{E}\left[\sup_{\|\theta\|_2 \leqslant D} \frac{1}{n}\sum_{i=1}^{n} \varepsilon_i \langle \theta, \Phi(X_i)\rangle\right] \\
&= 2GD\mathbb{E}\left\|\frac{1}{n}\sum_{i=1}^{n} \varepsilon_i \Phi(X_i)\right\|_2 \\
&\leqslant 2GD\sqrt{\mathbb{E}\left\|\frac{1}{n}\sum_{i=1}^{n} \varepsilon_i \Phi(X_i)\right\|_2^2} \\
&\leqslant \frac{2GRD}{\sqrt{n}}
\end{aligned}
$$

With probability $1-\delta$:

$$
\sup_{\theta \in \Theta} \left| f(\theta) - \hat{f}(\theta) \right| \leqslant \frac{1}{\sqrt{n}} \left(\ell_0 + GRD\right)\left(4 + \sqrt{2\log(1/\delta)}\right)
$$

Empirical Risk vs Fluctuation

- We have, with probability $1-\delta$, for all $\theta \in \Theta$:

$$f(\hat{\theta}) - \min_{\theta\in\Theta} f(\theta) \leqslant \sup_{\theta\in\Theta}\{\hat{f}(\theta) - f(\theta)\} + \sup_{\theta\in\Theta}\{f(\theta) - \hat{f}(\theta)\}$$
$$\leqslant \frac{2}{\sqrt{n}}(\ell_0 + GRD)(4 + \sqrt{2\log\frac{1}{\delta}})$$

- Only need to optimize with precision $\approx 1/\sqrt{n}$

Slow rate for supervised learning

Assumptions (f is the expected risk, $\hat{f}$ the empirical risk)

- $\Omega(\theta) = \|\theta\|_2$ (Euclidean norm)
- "Linear" predictors: $\phi_\theta(x) = \langle \theta, \Phi(x) \rangle$, with $\|\Phi(x)\|_2 \leqslant R$ a.s.
- G-Lipschitz loss: f and $\hat{f}$ are GR-Lipschitz on $\Theta = \{\|\theta\|_2 \leqslant D\}$
- No assumptions regarding convexity

- With probability greater than $1 - \delta$

$$\sup_{\theta \in \Theta} |\hat{f}(\theta) - f(\theta)| \leqslant \frac{\ell_0 + GRD}{\sqrt{n}} \left[2 + \sqrt{2 \log \frac{2}{\delta}} \right]$$

- Expected estimation error: $\mathbb{E}\Big[\sup_{\theta \in \Theta} |\hat{f}(\theta) - f(\theta)|\Big] \leqslant \dfrac{4\ell_0 + 4GRD}{\sqrt{n}}$
- Under other conditions on the model, can we improve the rate $1/\sqrt{n}$?

Motivation from mean estimation

Estimator

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} Z_i = \arg\min_{\theta \in \mathbb{R}} \hat{f}(\theta)$$

where

$$\hat{f}(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(Z_i - \theta)^2 \quad f(\theta) = \mathbb{E}\Big[(Z - \theta)^2\Big]$$

Slow rate

$$f(\theta) = \frac{1}{2}(\theta - \mathbb{E}[Z])^2 + \frac{1}{2}\,\mathrm{var}(Z) = \hat{f}(\theta) + O(n^{-1/2})$$

Motivation from mean estimation

Estimator

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} Z_i = \arg\min_{\theta\in\mathbb{R}} \hat{f}(\theta)$$

where

$$\hat{f}(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(Z_i - \theta)^2 \quad f(\theta) = \mathbb{E}\Big[(Z-\theta)^2\Big]$$

Fast rate

$$f(\hat{\theta}) - f(\mathbb{E}[Z]) = \frac{1}{2}(\hat{\theta} - \mathbb{E}[Z])^2$$

$$\mathbb{E}\big[f(\hat{\theta}) - f(\mathbb{E}[Z])\big] = \frac{1}{2}\mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} Z_i - \mathbb{E}[Z]\right)^2 = \frac{1}{2n}\,\mathrm{var}(Z)$$

Bound only at $\hat{\theta}$ + strong convexity

Fast rate for supervised learning

Assumptions (f is the expected risk, $\hat{f}$ the empirical risk)

- Same as before (bounded features, Lipschitz loss) + **strong convexity**

For any $a > 0$, with probability greater than $1 - \delta$, for all $\theta \in \mathbb{R}^d$,

$$f(\hat{\theta}) - \min_{\eta \in \mathbb{R}^d} f(\eta) \leqslant \frac{8(1 + a^{-1})G^2R^2(32 + \log(\delta^{-1}))}{\mu n}$$

- Results from (Sridharan et al., 2008), (Boucheron et al., 2012).
- **Strongly convex functions $\Rightarrow$ fast rate**

Minimization of the expected and empirical risk

- Conclusion: $\boxed{\hat{\theta} \in \arg\min_{\theta \in \Theta} \hat{f}(\theta)}$ is a good proxy as a minimizer of f as n is large.
- Question: How to find $\hat{\theta}$?
- Answer: gradient descent algorithms!
- Recall $\hat{f}$ is assumed to be convex.
- Very efficient methods from convex optimization are available: see part 2 and 3!

Conclusion

SLT insights

- Statistical approach sheds light on optimization techniques
- High precision is not (always) very relevant in ML

Directions:

- Faster Rates (Least squares regression)
- Markov chain interpretations
- Beyond Convex, beyond gradients (EM algorithm)

References

- Sebastien Bubeck's book and blog on optimization.
- Francis Bach's book on Learning.