Optimization for Machine Learning

Aymeric Dieuleveut

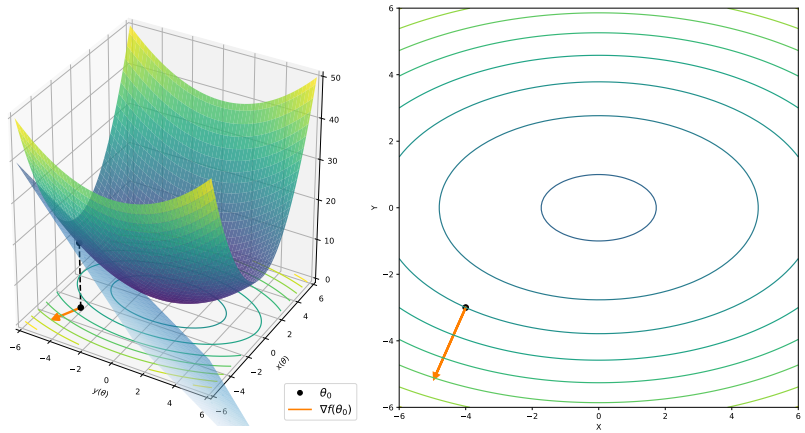
February 4, 2026

Outline

Outline

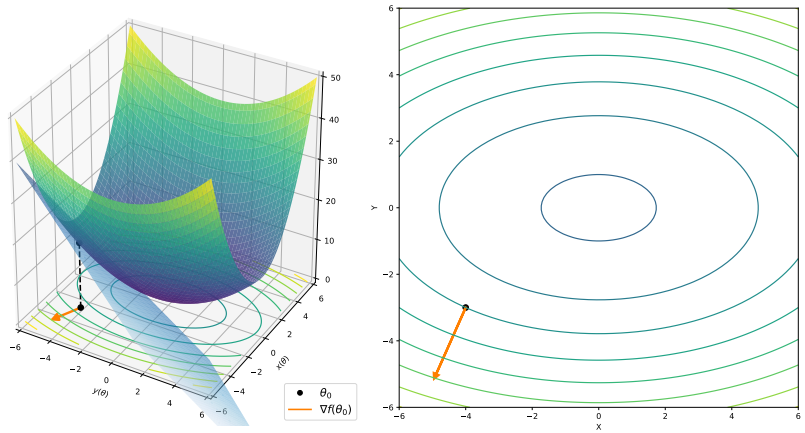
Gradient Descent: intuition

Given an initial iterate θ_0 and information $f(\theta_0), \nabla f(\theta_0)$ how to improve the model?



Gradient Descent: intuition

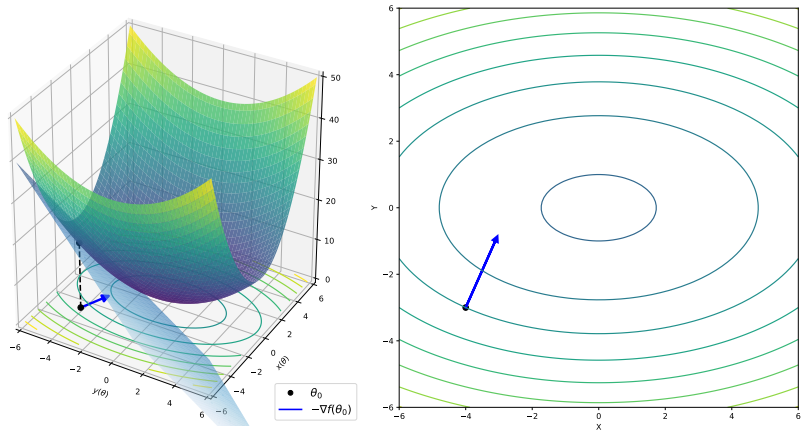
Given an initial iterate θ_0 and information $f(\theta_0), \nabla f(\theta_0)$ how to improve the model?



- $\nabla f(\theta_0)$ points in the steepest increase direction from θ_0

Gradient Descent: intuition

Given an initial iterate θ_0 and information $f(\theta_0), \nabla f(\theta_0)$ how to improve the model?

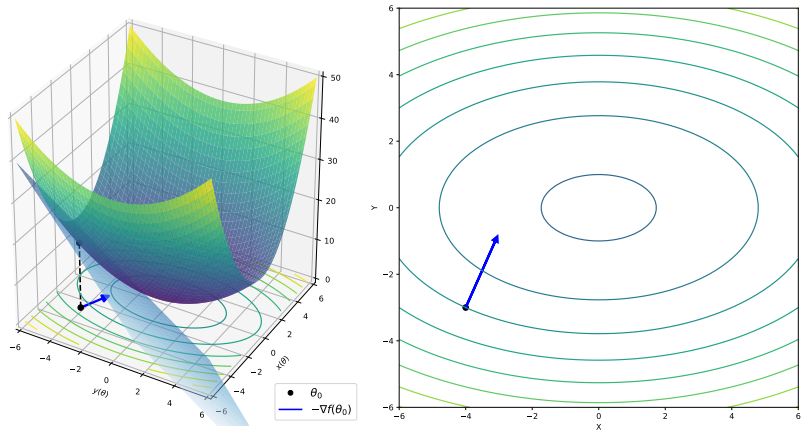


- $\nabla f(\theta_0)$ points in the steepest increase direction from θ_0

$\rightarrow$ $-\nabla f(\theta_0)$ points in the steepest **decrease** direction from θ_0

Gradient Descent: intuition

Given an initial iterate θ_0 and information $f(\theta_0), \nabla f(\theta_0)$ how to improve the model?



- $\nabla f(\theta_0)$ points in the steepest increase direction from θ_0

$\rightarrow$ $-\nabla f(\theta_0)$ points in the steepest **decrease** direction from θ_0

Algorithm : Gradient descent (GD)

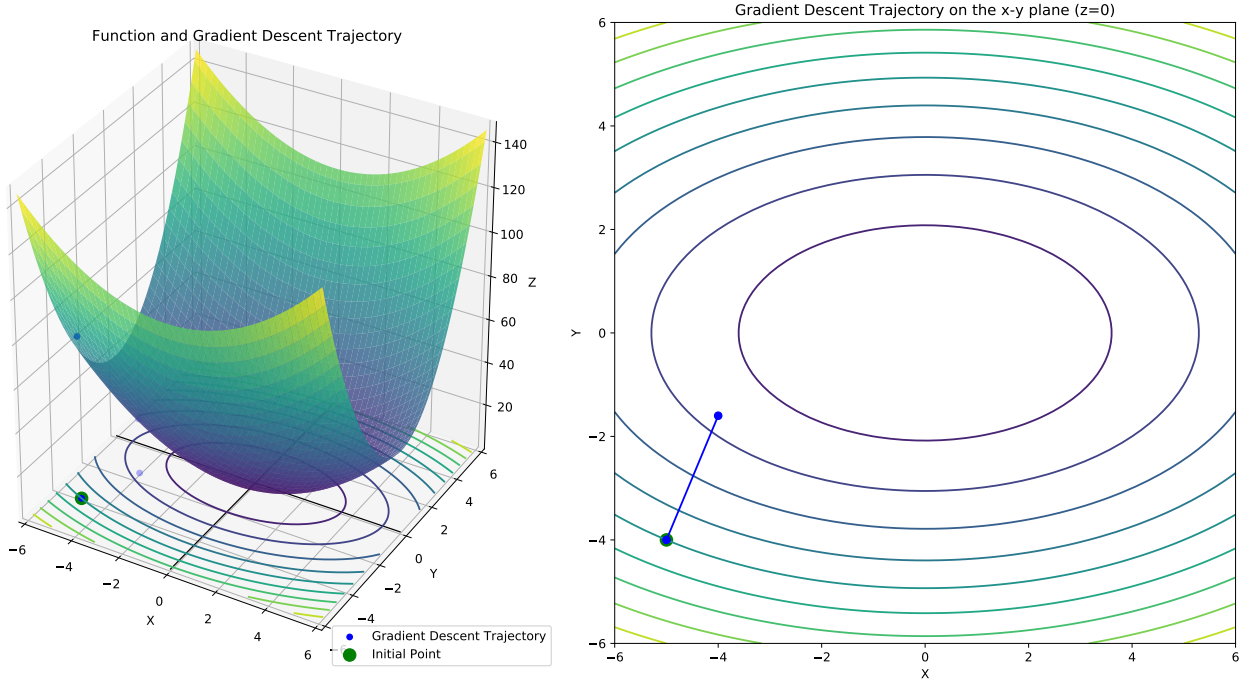
Input: Function f to minimize.
Initialization: initial weight vector $\theta_{(0)}$
Hyper-parameters: step size $\eta > 0$, number of iterations T.
For $t \in \{1, \ldots, T\}$:

- $\theta_{t+1} \leftarrow \theta_t - \eta \nabla f(\theta_t)$
- $t \leftarrow t + 1$.

Output: θ_T.

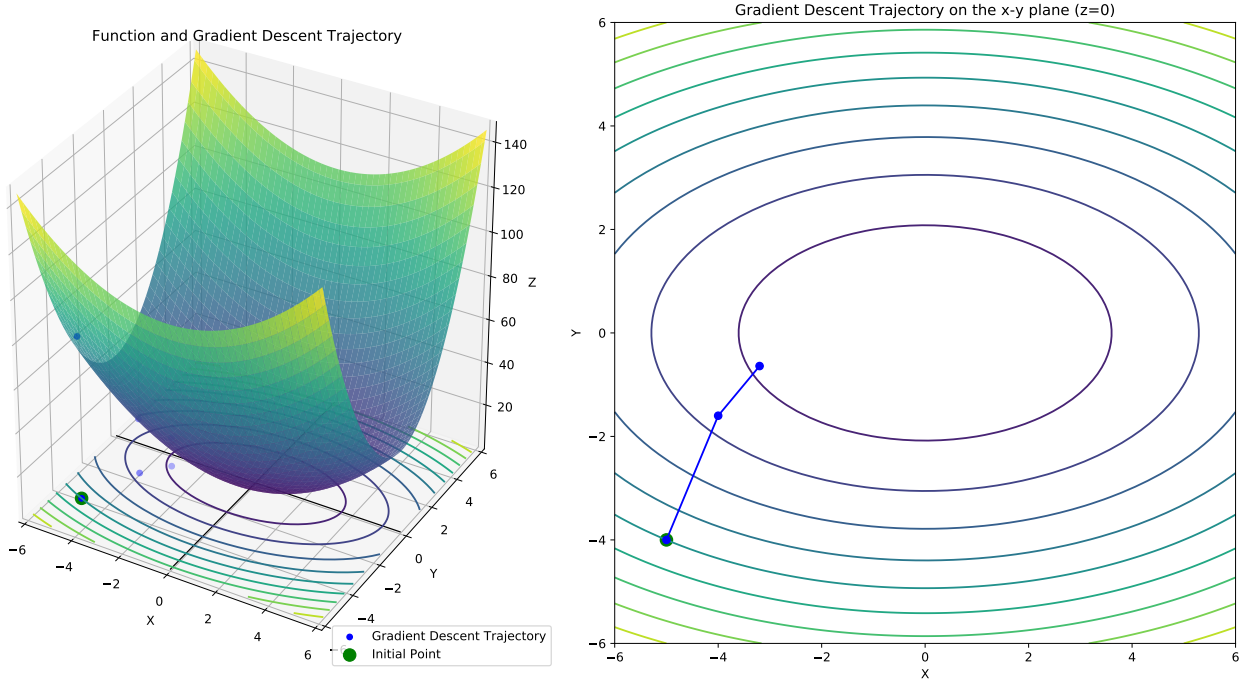
Gradient Descent: illustration and impact of hyperparameters

Impact of number of iterations (hyperparameter T)



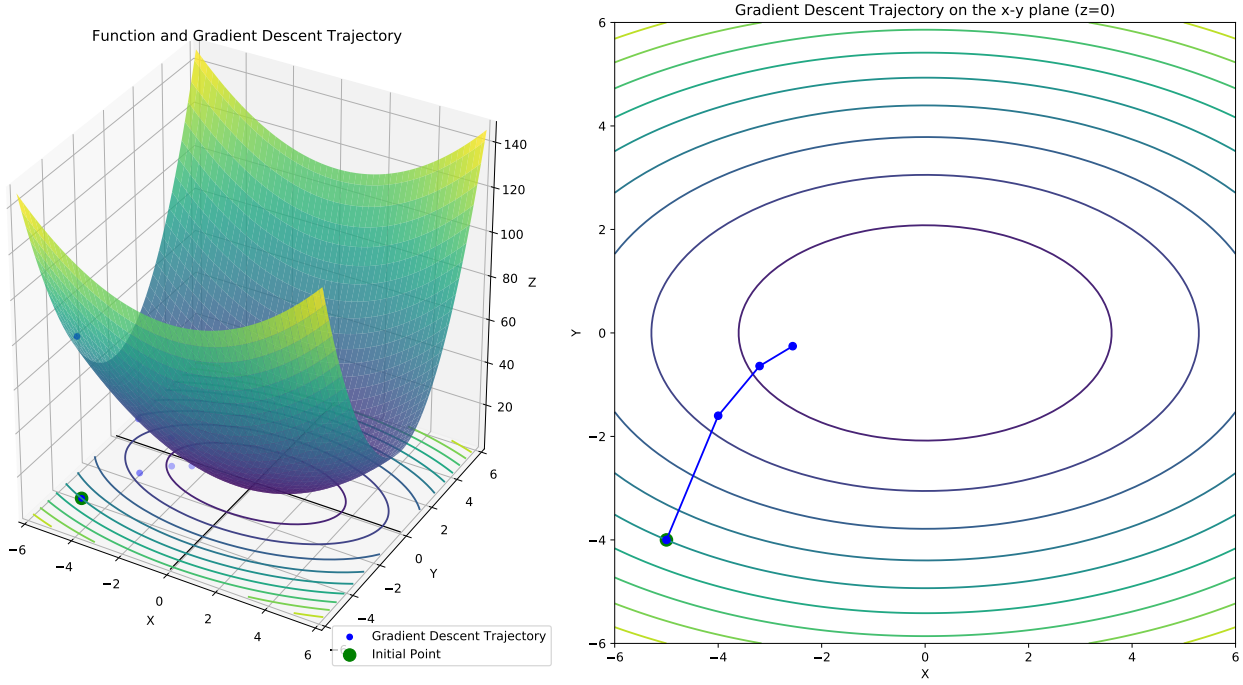
Gradient Descent: illustration and impact of hyperparameters

Impact of number of iterations (hyperparameter T)



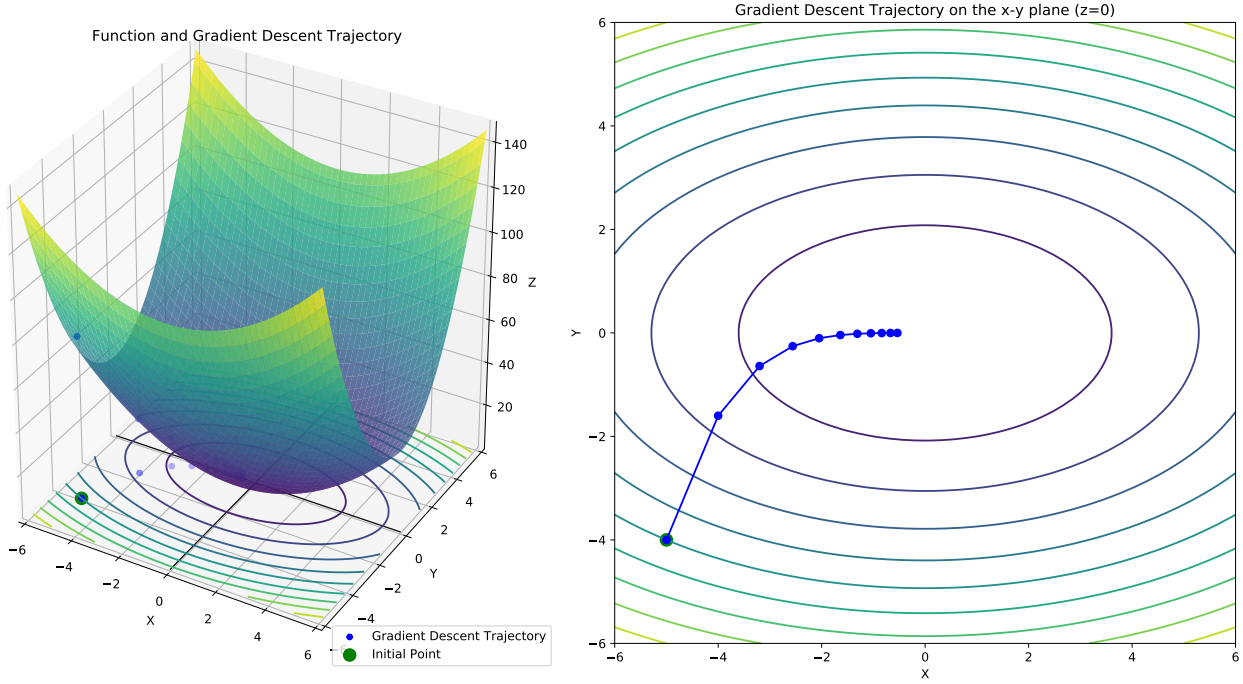
Gradient Descent: illustration and impact of hyperparameters

Impact of number of iterations (hyperparameter T)



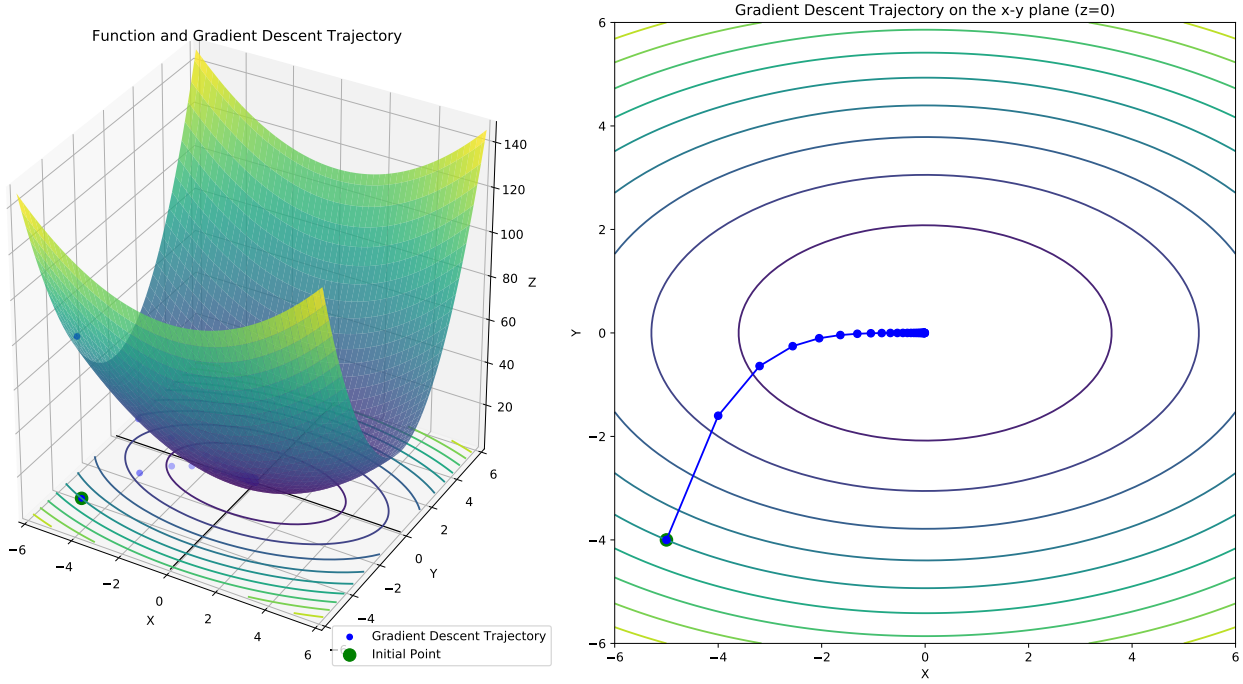
Gradient Descent: illustration and impact of hyperparameters

Impact of number of iterations (hyperparameter T)



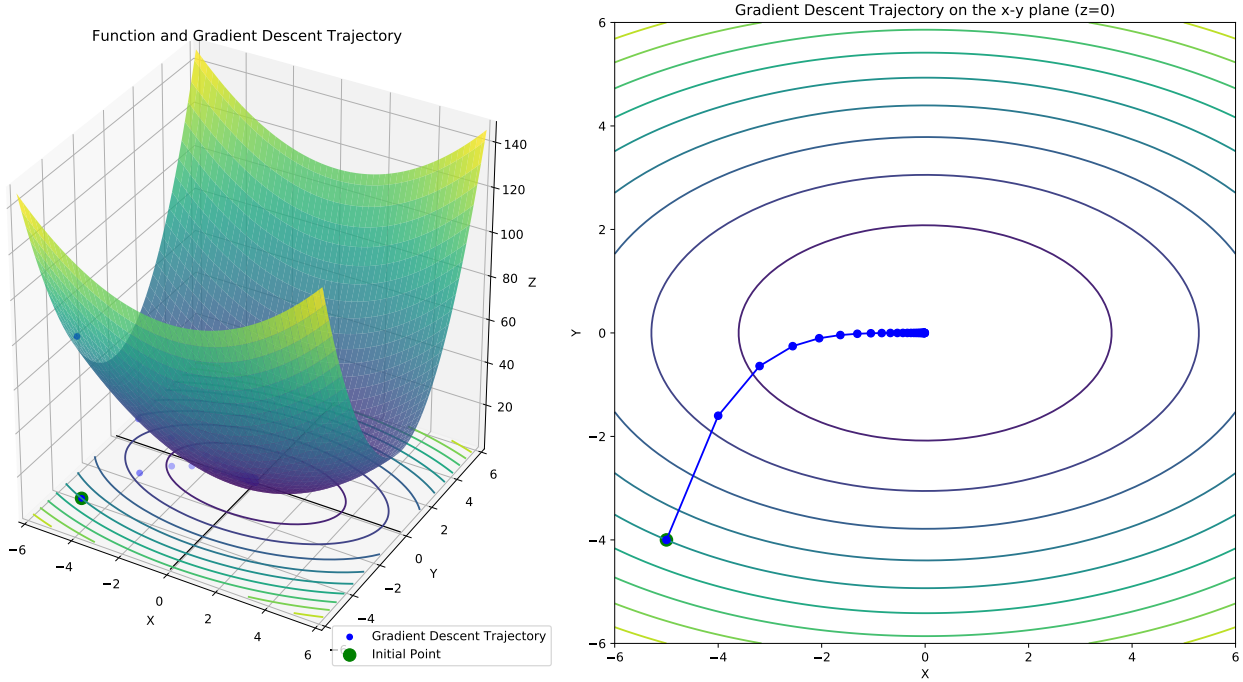
Gradient Descent: illustration and impact of hyperparameters

Impact of number of iterations (hyperparameter T)



Gradient Descent: illustration and impact of hyperparameters

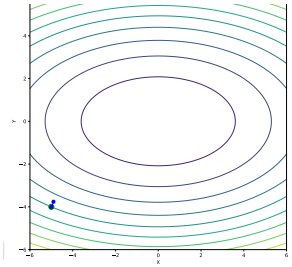
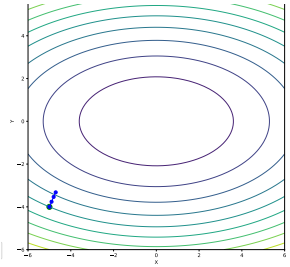
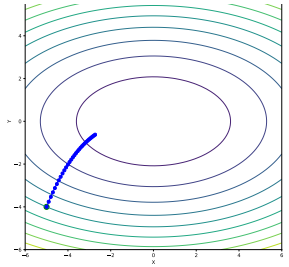
Impact of number of iterations (hyperparameter T)



Conclusion: choose T as large as possible (depending ob the available time).

Gradient Descent: illustration and impact of hyperparameters

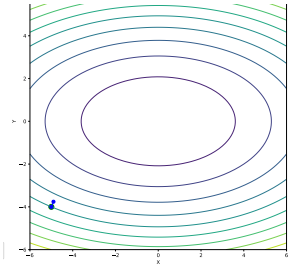
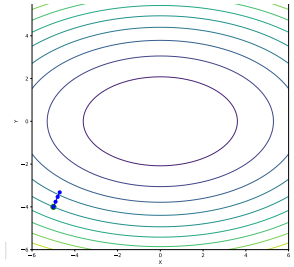
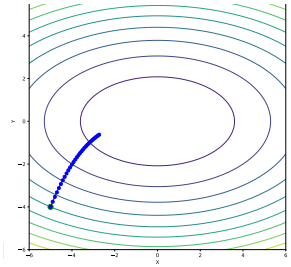
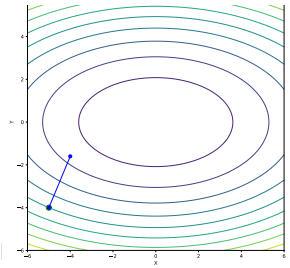
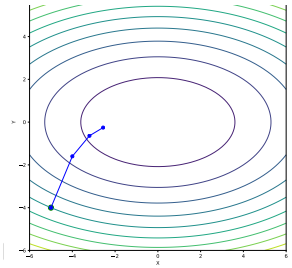
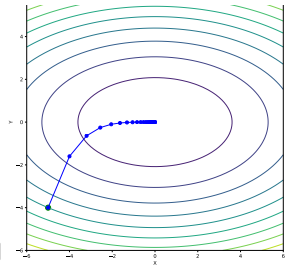
Impact of the step-size/learning-rate (hyperparameter η):

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence

Conclusion: choose η as large as possible while maintaining convergence (avoiding amplification in oscillations)

Gradient Descent: illustration and impact of hyperparameters

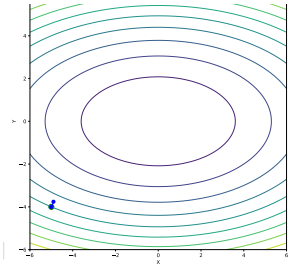
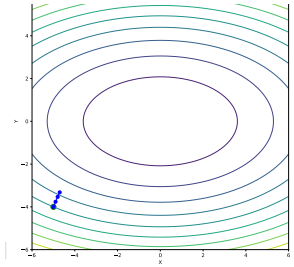
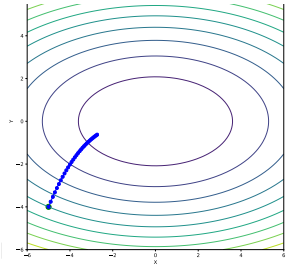
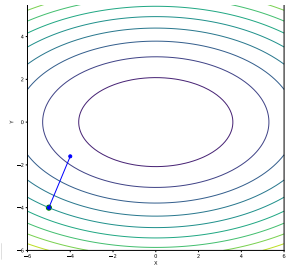
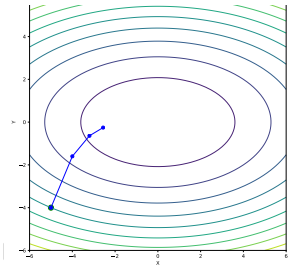
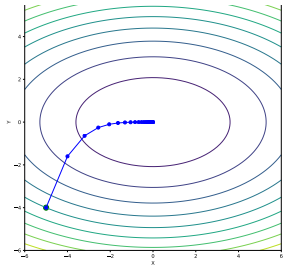
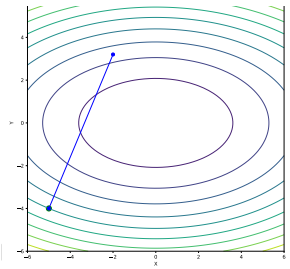
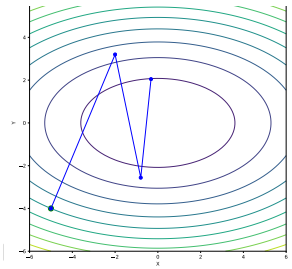
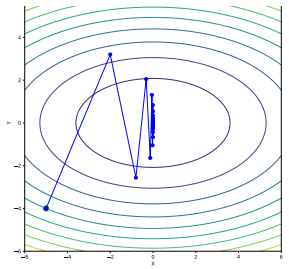
Impact of the step-size/learning-rate (hyperparameter η):

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence
ok-small				→ convergence without oscillations

Conclusion: choose η as large as possible while maintaining convergence (avoiding amplification in oscillations)

Gradient Descent: illustration and impact of hyperparameters

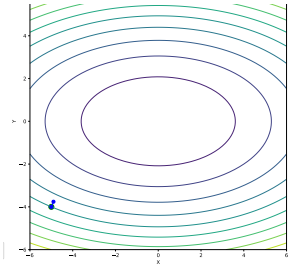
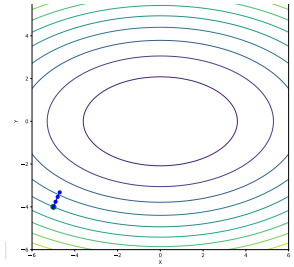
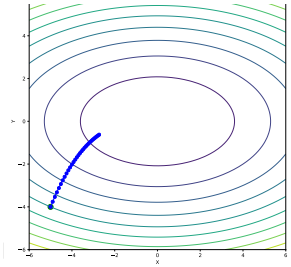
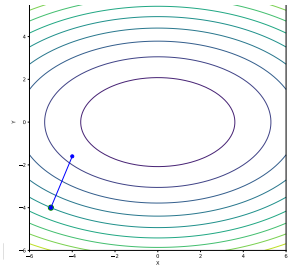
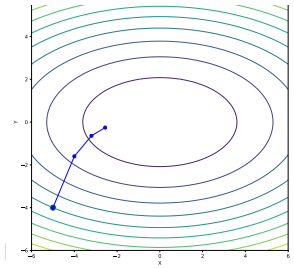
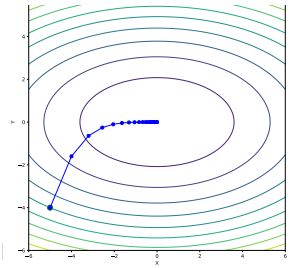
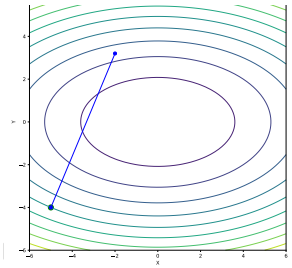
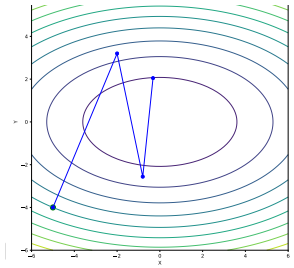
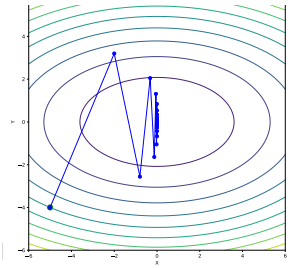
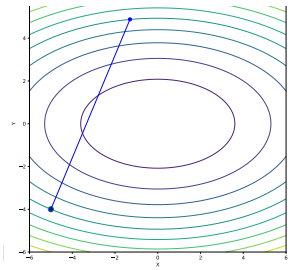
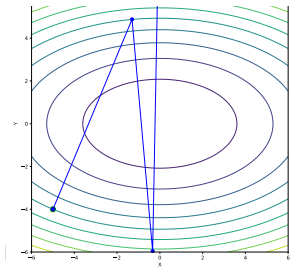
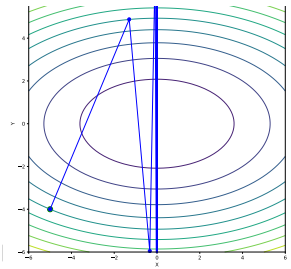
Impact of the step-size/learning-rate (hyperparameter η):

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence
ok-small				→ convergence without oscillations
ok-large				→ convergence with oscillations

Conclusion: choose η as large as possible while maintaining convergence (avoiding amplification in oscillations)

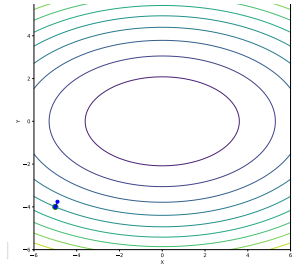
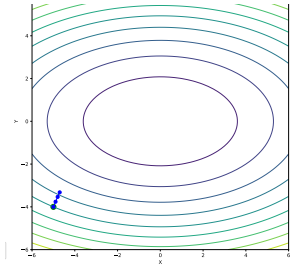
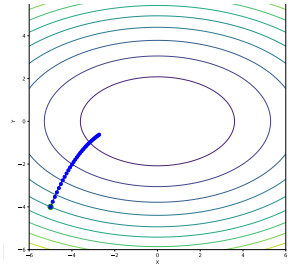
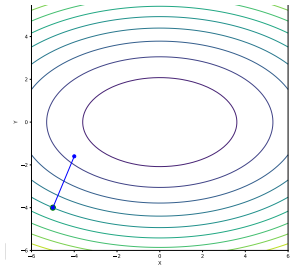
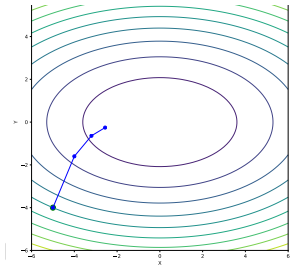
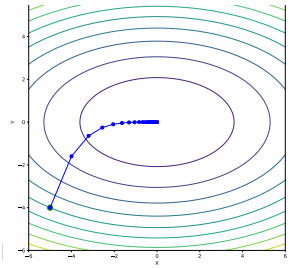
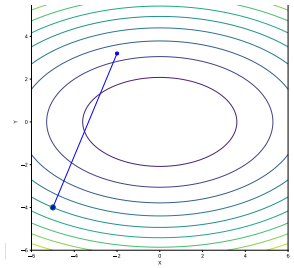
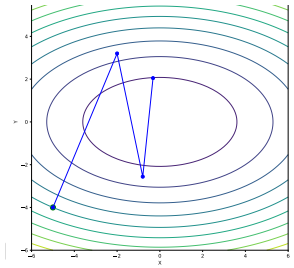
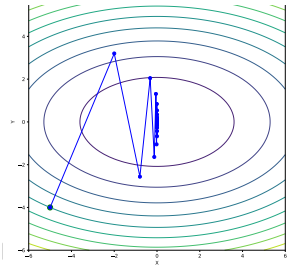
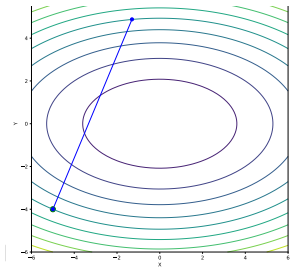
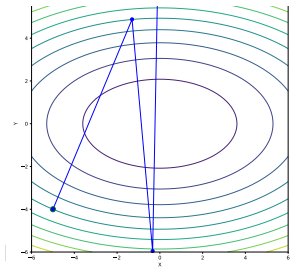
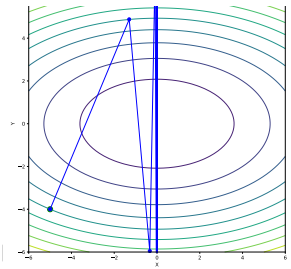
Gradient Descent: illustration and impact of hyperparameters

Impact of the step-size/learning-rate (hyperparameter η):

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence
ok-small				→ convergence without oscillations
ok-large				→ convergence with oscillations
too large				→ diverging oscillations

Gradient Descent: illustration and impact of hyperparameters

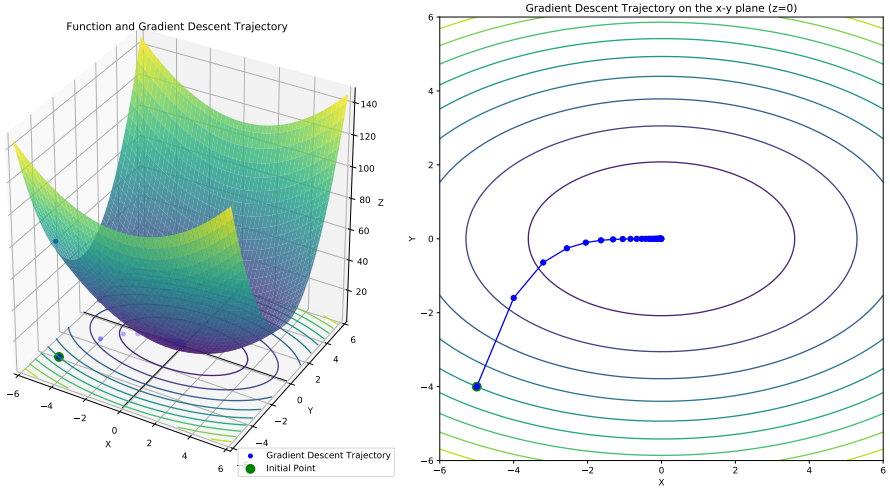
Impact of the step-size/learning-rate (hyperparameter η):

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence
ok-small				→ convergence without oscillations
ok-large				→ convergence with oscillations
too large				→ diverging oscillations

Conclusion: choose η as large as possible while maintaining convergence (avoiding amplification in oscillations)

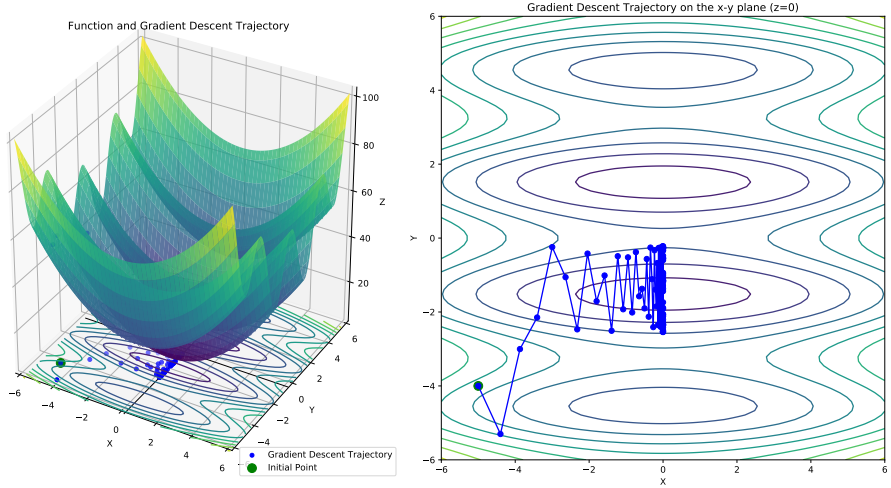
Gradient Descent: illustration and impact of the shape of the function

Convex function:



→ typically converges to the global minimum.

Non-convex function:



→ typically converges to a local minimum.

Theoretical grounds for GD method

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

→ **Formal results:**

1. Grounded formulation of gradient descent
   a. Local minimization of the first order approximation
   b. Global minimization of the quadratic upper bound for smooth functions
2. Convergence results and optimization complexity.
   a. For convex and smooth functions.
   b. For strongly convex and smooth functions.

Remark: Why do we want convergence rates and proofs:

- Rates support the choice of "optimal" hyperparameters.
- Rates allow us to **compare algorithms**.

Grounded formulation of gradient descent:
a. local minimization of the first order approximation

Recall that, around θ_0, the first-order Taylor approximation gives

$$f(\theta) = \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle}_{\text{First-order approximation}} + \underbrace{O(\|\theta - \theta_0\|^2)}_{\substack{\text{mathematical notation for a} \\ \text{negligible residual scaling as } \|\theta - \theta_0\|^2}} \qquad \text{(Taylor, 1st-order)}$$

Proposition : GD as local minimization of FOA

Consider a neighborhood $B(\theta_0, R)$ of θ_0, then:

$$\underset{\theta \in B(\theta_0, R)}{\text{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle}_{\text{First-order approximation}} = \underbrace{\theta_0 - \eta \nabla f(\theta_0)}_{\text{GD!}} \qquad \text{with } \eta = \frac{R}{\|\nabla f(\theta_0)\|}$$

Grounded formulation of gradient descent:
a. local minimization of the first order approximation

Recall that, around θ_0, the first-order Taylor approximation gives

$$f(\theta) = \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle}_{\text{First-order approximation}} + \underbrace{O(\|\theta - \theta_0\|^2)}_{\substack{\text{mathematical notation for a} \\ \text{negligible residual scaling as } \|\theta - \theta_0\|^2}} \qquad \text{(Taylor, 1st-order)}$$

Proposition : GD as local minimization of FOA

Consider a neighborhood $B(\theta_0, R)$ of θ_0, then:

$$\underset{\theta \in B(\theta_0, R)}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle}_{\text{First-order approximation}} = \underbrace{\theta_0 - \eta \nabla f(\theta_0)}_{\text{GD!}} \qquad \text{with } \eta = \frac{R}{\|\nabla f(\theta_0)\|}$$

Proof:

$$\begin{aligned} \underset{\theta \in B(\theta_0, R)}{\operatorname{argmin}} f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle &= \underset{\theta \in B(\theta_0, R)}{\operatorname{argmin}} \langle \nabla f(\theta_0), \theta - \theta_0 \rangle \\ &= \underset{\theta \in B(\theta_0, R)}{\operatorname{argmax}} \langle -\nabla f(\theta_0), \theta - \theta_0 \rangle \end{aligned}$$

And by Cauchy-Schwarz inequality $\langle -\nabla f(\theta_0), \theta - \theta_0 \rangle \leqslant \|-\nabla f(\theta_0)\| \|\theta - \theta_0\|$ with an equality iif $\theta - \theta_0$ is positively aligned with $-\nabla f(\theta_0)$, i.e. $\theta = \theta_0 - \eta - \nabla f(\theta_0)$.

Grounded formulation of gradient descent:
b. Global minimization of the quadratic upper bound for smooth functions

Consider an L-smooth function f.
Recall that, at point θ_0, uniformly over θ, we have:

$$f(\theta) \leqslant \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{L}{2}\|\theta - \theta_0\|^2}_{\text{Quadratic upper bound}} \qquad \text{(Smoothness)}$$

Proposition : GD as global minimization of QUB

We have that:

$$\operatorname*{argmin}_{\theta \in \mathbb{R}^d} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{L}{2}\|\theta - \theta_0\|^2}_{\text{Quadratic upper bound}} = \underbrace{\theta_0 - \eta \nabla f(\theta_0)}_{\text{GD!}} \qquad \text{with } \eta = \frac{1}{L}$$

Grounded formulation of gradient descent:
b. Global minimization of the quadratic upper bound for smooth functions

Consider an L-smooth function f.
Recall that, at point θ_0, uniformly over θ, we have:

$$f(\theta) \leqslant \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{L}{2}\|\theta - \theta_0\|^2}_{\text{Quadratic upper bound}} \qquad \text{(Smoothness)}$$

Proposition : GD as global minimization of QUB

We have that:

$$\operatorname*{argmin}_{\theta \in \mathbb{R}^d} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{L}{2}\|\theta - \theta_0\|^2}_{\text{Quadratic upper bound}} = \underbrace{\theta_0 - \eta \nabla f(\theta_0)}_{\text{GD!}} \qquad \text{with } \eta = \frac{1}{L}$$

Proof: let

$$g(\theta) := f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{L}{2}\|\theta - \theta_0\|^2.$$

Then g is convex and

$$\nabla g(\theta) = \nabla f(\theta_0) + L(\theta - \theta_0)$$

.
Thus g is minimized iif. $\nabla g(\theta) = 0$, i.e., iif. $\theta = \theta_0 - \frac{1}{L}\nabla f(\theta_0)$.

Grounded formulation of gradient descent:
b. Global minimization of the quadratic upper bound for smooth functions

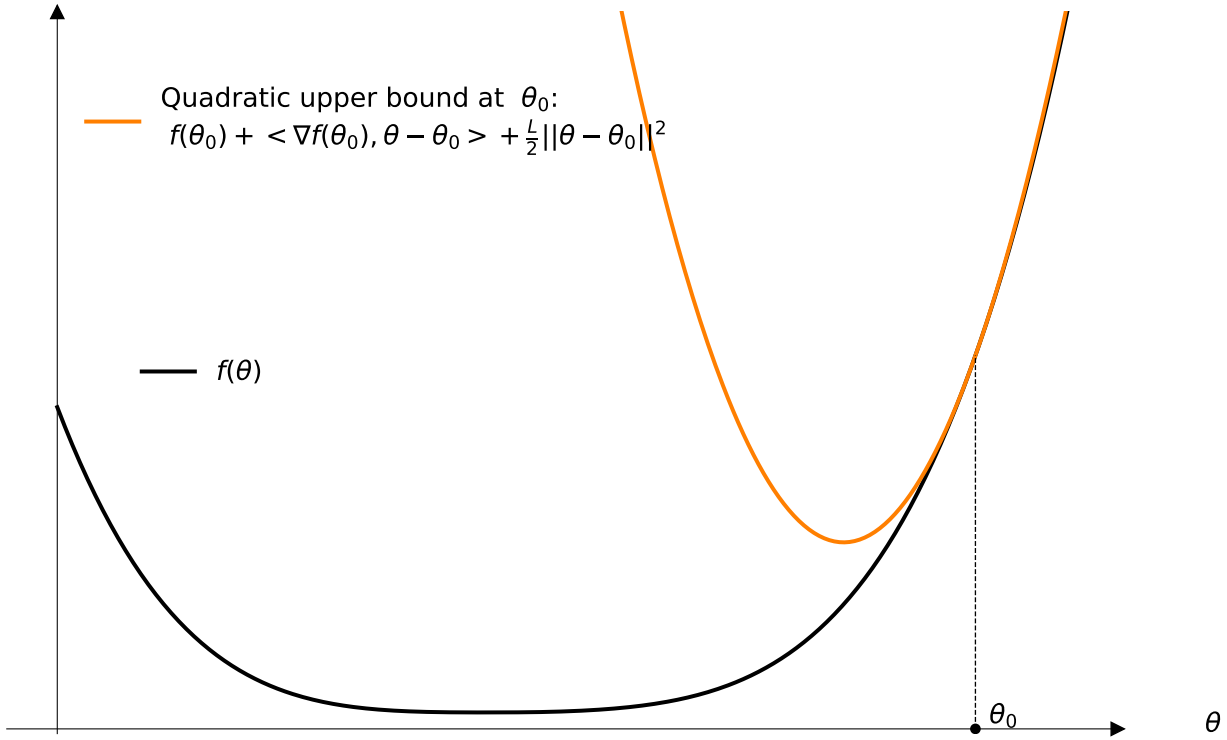
Geometric interpretation. GD with step-size $\frac{1}{L}$ corresponds to iteratively minimizing the quadratic upper bound around θ_{t-1}: for all t

$$\underbrace{\theta_{t+1} = \theta_t - \frac{1}{L}\nabla f(\theta_t)}_{\text{GD from } \theta_t} = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_t) + \langle \nabla f(\theta_t), \theta - \theta_{t-1} \rangle + \frac{L}{2}\|\theta - \theta_t\|^2}_{\mathrm{QUB}_{\theta_t} \text{ around } \theta_t}$$

Grounded formulation of gradient descent:
b. Global minimization of the quadratic upper bound for smooth functions

Geometric interpretation. GD with step-size $\frac{1}{L}$ corresponds to iteratively minimizing the quadratic upper bound around θ_{t-1}: for all t

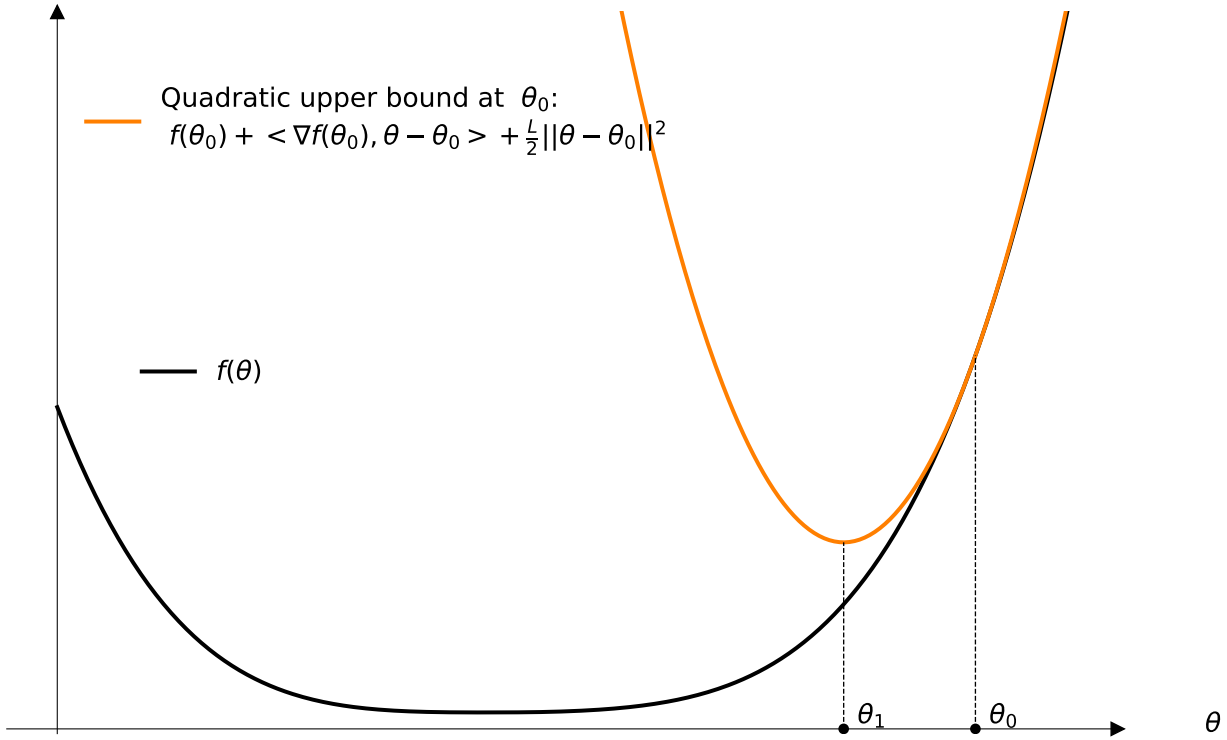
$$\underbrace{\theta_{t+1} = \theta_t - \frac{1}{L}\nabla f(\theta_t)}_{\text{GD from } \theta_t} = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_t) + \langle \nabla f(\theta_t), \theta - \theta_{t-1} \rangle + \frac{L}{2}\|\theta - \theta_t\|^2}_{\mathrm{QUB}_{\theta_t} \text{ around } \theta_t}$$



Grounded formulation of gradient descent:
b. Global minimization of the quadratic upper bound for smooth functions

Geometric interpretation. GD with step-size $\frac{1}{L}$ corresponds to iteratively minimizing the quadratic upper bound around θ_{t-1}: for all t

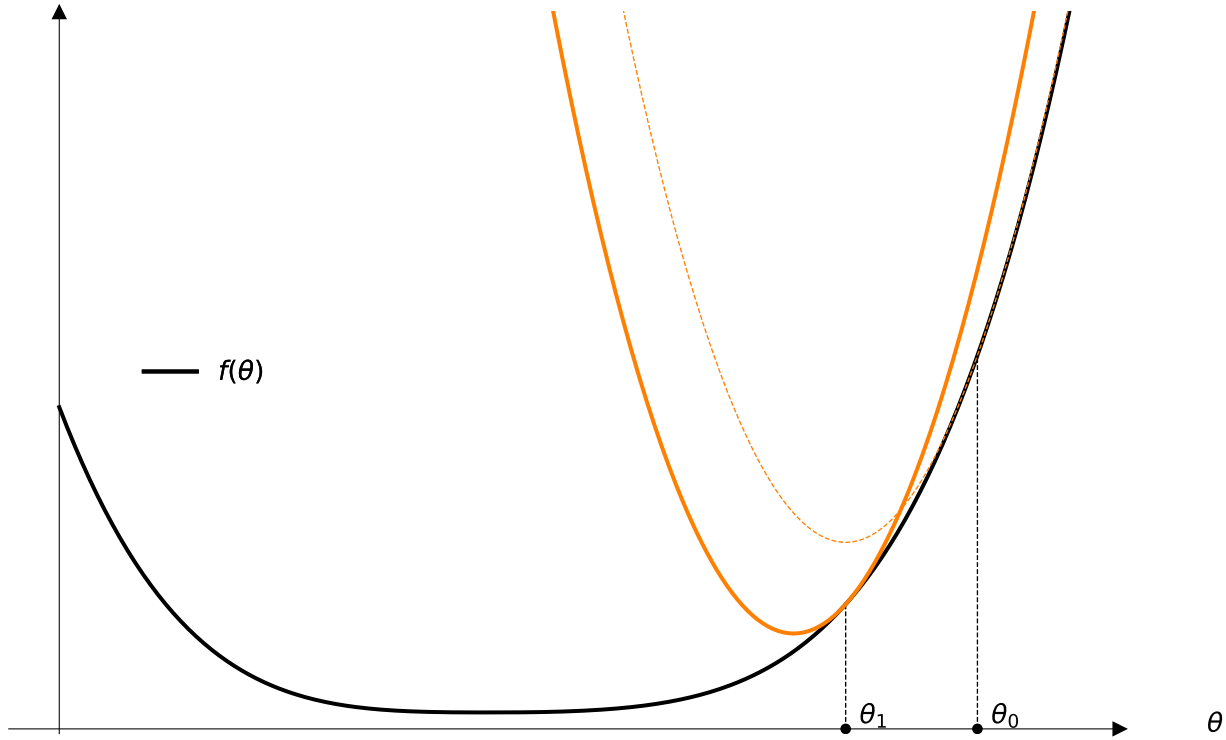
$$\underbrace{\theta_{t+1} = \theta_t - \frac{1}{L}\nabla f(\theta_t)}_{\text{GD from } \theta_t} = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_t) + \langle \nabla f(\theta_t), \theta - \theta_{t-1} \rangle + \frac{L}{2}\|\theta - \theta_t\|^2}_{\mathrm{QUB}_{\theta_t} \text{ around } \theta_t}$$



Grounded formulation of gradient descent:
b. Global minimization of the quadratic upper bound for smooth functions

Geometric interpretation. GD with step-size $\frac{1}{L}$ corresponds to iteratively minimizing the quadratic upper bound around θ_{t-1}: for all t

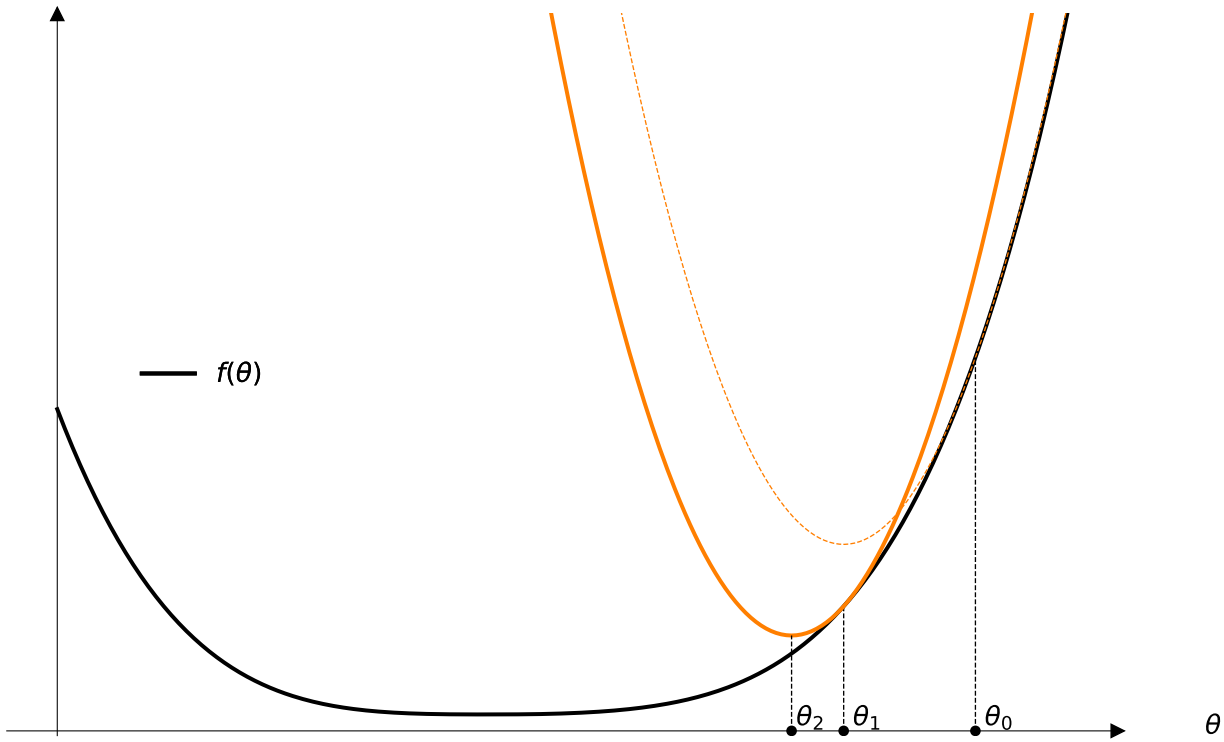
$$\underbrace{\theta_{t+1} = \theta_t - \frac{1}{L}\nabla f(\theta_t)}_{\text{GD from } \theta_t} = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_t) + \langle \nabla f(\theta_t), \theta - \theta_{t-1} \rangle + \frac{L}{2}\|\theta - \theta_t\|^2}_{\mathrm{QUB}_{\theta_t} \text{ around } \theta_t}$$



Grounded formulation of gradient descent:
b. Global minimization of the quadratic upper bound for smooth functions

Geometric interpretation. GD with step-size $\frac{1}{L}$ corresponds to iteratively minimizing the quadratic upper bound around θ_{t-1}: for all t

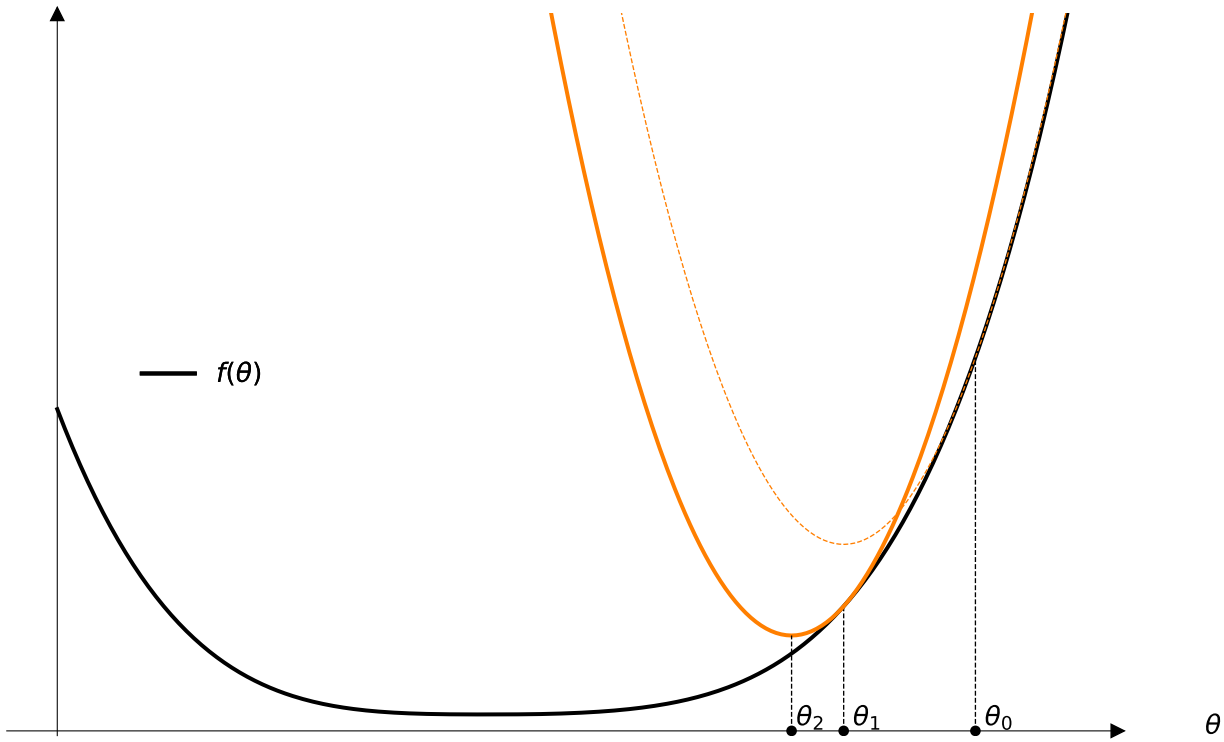
$$\underbrace{\theta_{t+1} = \theta_t - \frac{1}{L}\nabla f(\theta_t)}_{\text{GD from } \theta_t} = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_t) + \langle \nabla f(\theta_t), \theta - \theta_{t-1} \rangle + \frac{L}{2}\|\theta - \theta_t\|^2}_{\mathrm{QUB}_{\theta_t} \text{ around } \theta_t}$$



Grounded formulation of gradient descent:
b. Global minimization of the quadratic upper bound for smooth functions

Geometric interpretation. GD with step-size $\frac{1}{L}$ corresponds to iteratively minimizing the quadratic upper bound around θ_{t-1}: for all t

$$\underbrace{\theta_{t+1} = \theta_t - \frac{1}{L}\nabla f(\theta_t)}_{\text{GD from } \theta_t} = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_t) + \langle \nabla f(\theta_t), \theta - \theta_{t-1} \rangle + \frac{L}{2}\|\theta - \theta_t\|^2}_{\mathrm{QUB}_{\theta_t} \text{ around } \theta_t}$$



Consequence: *descent lemma* $f(\theta_{t+1}) \leqslant \mathrm{QUB}_{\theta_t}(\theta_{t+1}) \leqslant \mathrm{QUB}_{\theta_t}(\theta_t) = f(\theta_t)$.

Convergence & optimization complexity: a. smooth convex f.

Theorem : Convergence of GD on smooth and convex functions

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then, GD with step size $\eta \leqslant 2/L$ satisfies, after t steps:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|_2^2}{2\eta(1 - \eta L/2)t}.$$

Convergence & optimization complexity: a. smooth convex f.

Theorem : Convergence of GD on smooth and convex functions

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then, GD with step size $\eta \leqslant 2/L$ satisfies, after t steps:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|_2^2}{2\eta(1 - \eta L/2)t}.$$

Interpretation:

1. Convergence: $f(\theta_t) \to f(\theta_\star)$ ad $t \to \infty$.

Convergence & optimization complexity: a. smooth convex f.

Theorem : Convergence of GD on smooth and convex functions

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then, GD with step size $\eta \leqslant 2/L$ satisfies, after t steps:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|_2^2}{2\eta(1 - \eta L/2)t}.$$

Interpretation:

1. Convergence: $f(\theta_t) \to f(\theta_\star)$ ad $t \to \infty$.
2. Impact of η (confirms illustration/intuition):
   - $\eta > 2/L$ no guarantee (possible divergence)
   - best bound $\to$ choose η to maximize $\eta(1 - \eta L/2)$

     $\to$ choose η as large as possible while avoiding oscillations $\to \eta = \frac{1}{L}$, giving:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{L\|\theta_0 - \theta_\star\|_2^2}{t}.$$

Convergence & optimization complexity: a. smooth convex f.

Theorem : Convergence of GD on smooth and convex functions

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then, GD with step size $\eta \leqslant 2/L$ satisfies, after t steps:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|_2^2}{2\eta(1 - \eta L/2)t}.$$

Interpretation:

1. Convergence: $f(\theta_t) \to f(\theta_\star)$ ad $t \to \infty$.
2. Impact of η (confirms illustration/intuition):
   - $\eta > 2/L$ no guarantee (possible divergence)
   - best bound $\to$ choose η to maximize $\eta(1 - \eta L/2)$
     $\to$ choose η as large as possible while avoiding oscillations $\to \eta = \frac{1}{L}$, giving:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{L\|\theta_0 - \theta_\star\|_2^2}{t}.$$

3. Initial condition $\|\theta_0 - \theta_\star\|_2^2$: starting further $\to$ more iterations needed.

Convergence & optimization complexity: a. smooth convex f.

Theorem : Convergence of GD on smooth and convex functions

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then, GD with step size $\eta \leqslant 2/L$ satisfies, after t steps:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|_2^2}{2\eta(1 - \eta L/2)t}.$$

Interpretation:

1. Convergence: $f(\theta_t) \to f(\theta_\star)$ ad $t \to \infty$.
2. Impact of η (confirms illustration/intuition):
   - $\eta > 2/L$ no guarantee (possible divergence)
   - best bound $\to$ choose η to maximize $\eta(1 - \eta L/2)$
     $\to$ choose η as large as possible while avoiding oscillations $\to \eta = \frac{1}{L}$, giving:
     $$f(\theta_t) - f(\theta_\star) \leqslant \frac{L\|\theta_0 - \theta_\star\|_2^2}{t}.$$
3. Initial condition $\|\theta_0 - \theta_\star\|_2^2$: starting further $\to$ more iterations needed.
4. Rate $1/t$: to achieve an error $10\times$ smaller, need to perform $10\times$ mores iteration. Precisely, to ensure $f(\theta_t) - f(\theta_\star) \leqslant \epsilon$, we need at least T_ϵ iterations with
   $$\underbrace{T_\epsilon}_{\text{iter. complexity}} \geqslant \frac{L\|\theta_0 - \theta_\star\|_2^2}{\epsilon}.$$

Thus $T_{\frac{\epsilon}{10}} = 10 \times T_\epsilon$.

Convergence & optimization complexity: a. smooth convex f.

Theorem : Convergence of GD on smooth and convex functions

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then, GD with step size $\eta \leqslant 2/L$ satisfies, after t steps:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|_2^2}{2\eta(1 - \eta L/2)t}.$$

Interpretation:

1. Convergence: $f(\theta_t) \to f(\theta_\star)$ ad $t \to \infty$.
2. Impact of η (confirms illustration/intuition):
   - $\eta > 2/L$ no guarantee (possible divergence)
   - best bound $\to$ choose η to maximize $\eta(1 - \eta L/2)$
     $\to$ choose η as large as possible while avoiding oscillations $\to \eta = \frac{1}{L}$, giving:
     $$f(\theta_t) - f(\theta_\star) \leqslant \frac{L\|\theta_0 - \theta_\star\|_2^2}{t}.$$
3. Initial condition $\|\theta_0 - \theta_\star\|_2^2$: starting further $\to$ more iterations needed.
4. Rate $1/t$: to achieve an error $10\times$ smaller, need to perform $10\times$ mores iteration. Precisely, to ensure $f(\theta_t) - f(\theta_\star) \leqslant \epsilon$, we need at least T_ϵ iterations with
   $$\underbrace{T_\epsilon}_{\text{iter. complexity}} \geqslant \frac{L\|\theta_0 - \theta_\star\|_2^2}{\epsilon}.$$

Thus $T_{\frac{\epsilon}{10}} = 10 \times T_\epsilon$.
Proof: see exercise sheet 1.

Convergence & optimization complexity: b. smooth and strongly-convex f.

Theorem : Convergence of GD on smooth & strongly convex functions

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

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{L}{2}\Big(1 - \eta\mu\Big)^t \|\theta_0 - \theta_\star\|_2^2.$$

Convergence & optimization complexity: b. smooth and strongly-convex f.

Theorem : Convergence of GD on smooth & strongly convex functions

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

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{L}{2}\Big(1 - \eta\mu\Big)^t \|\theta_0 - \theta_\star\|_2^2.$$

Interpretation:

1. Convergence: $f(\theta_t) \to f(\theta_\star)$ ad $t \to \infty$.
2. Impact of $\eta \to$ choose "maximal" $\eta = 1/L$.
3. Initial condition $\|\theta_0 - \theta_\star\|_2^2$: starting further $\to$ more iterations needed.

} $\rightsquigarrow$ Same as before!

Convergence & optimization complexity: b. smooth and strongly-convex f.

Theorem : Convergence of GD on smooth & strongly convex functions

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

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{L}{2}\Big(1 - \eta\mu\Big)^t \|\theta_0 - \theta_\star\|_2^2.$$

Interpretation:

1. Convergence: $f(\theta_t) \to f(\theta_\star)$ ad $t \to \infty$.
2. Impact of $\eta \to$ choose "maximal" $\eta = 1/L$.
3. Initial condition $\|\theta_0 - \theta_\star\|_2^2$: starting further $\to$ more iterations needed.

$\Big\}$ $\rightsquigarrow$ Same as before!

- Rate $\left(1 - \frac{\mu}{L}\right)^t = \left(1 - \frac{1}{\kappa}\right)^t$: to achieve an error $10\times$ smaller, need to perform a *constant number of extra iterations*. Precisely, to ensure $f(\theta_t) - f(\theta_\star) \leqslant \epsilon$, we need at least T_ϵ iterations with

$$\frac{L}{2}\left(1 - \frac{1}{\kappa}\right)^{T_\epsilon}\|\theta_0 - \theta_\star\|_2^2 \leqslant \epsilon \qquad \Leftrightarrow \qquad T_\epsilon \geqslant \frac{\ln\left(\frac{L\|\theta_0 - \theta_\star\|_2^2}{2\epsilon}\right)}{\ln\left(1 + \frac{1}{\kappa - 1}\right)} \simeq \kappa \ln\left(\frac{L\|\theta_0 - \theta_\star\|_2^2}{2\epsilon}\right)$$

Thus $T_{\frac{\epsilon}{10}} = T_\epsilon + \kappa \ln(10)$.

Impact of the condition number κ

For smooth and strongly convex functions, iteration complexity thus scales as

$$T_\epsilon \simeq \kappa \ln\left(\frac{L\|\theta_0 - \theta_\star\|_2^2}{2\epsilon}\right)$$

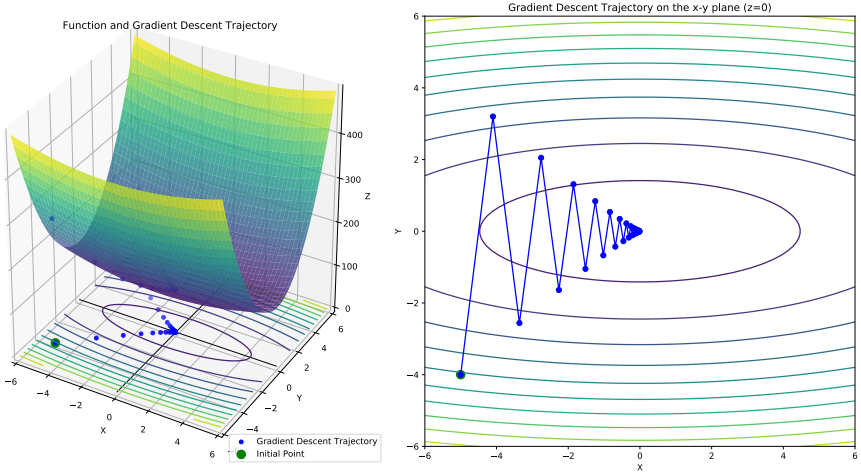
Bad conditioning (κ large) $\rightarrow$ more iterations needed, proportionally to κ.

Impact of the condition number κ

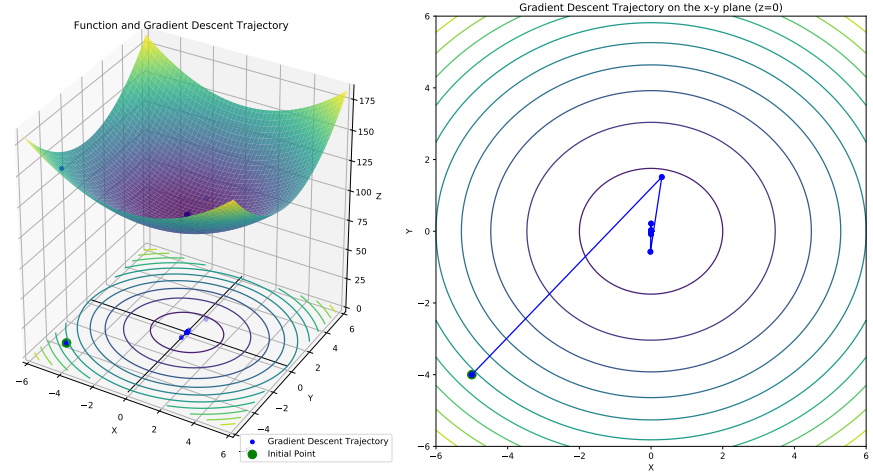
For smooth and strongly convex functions, iteration complexity thus scales as

$$T_\epsilon \simeq \kappa \ln\left(\frac{L\|\theta_0 - \theta_\star\|_2^2}{2\epsilon}\right)$$

Bad conditioning (κ large) $\longrightarrow$ more iterations needed, proportionally to κ.
Why is there such a dependency on κ? $\rightsquigarrow$ see discussion on conditioning before.
Moreover, for a poorly conditioned function, the gradient does not point towards the global optimum.



Large κ (10 here)
harder to optimize



Small κ (1.25 here)
easier to optimize

Remark: in machine learning, κ can be extremely large (typically $\geqslant 10^6$)

Cost per iteration and total cost for complexity ϵ

Cost per iteration C:

- computing $\nabla f(\theta_t) \in \mathbb{R}^d \rightsquigarrow$ cost $O(d)$.
- when f is the empirical risk, i.e., $f(\theta) = \frac{1}{n}\sum_{i=1}^n f_i(\theta) \rightsquigarrow$ cost $O(n \times d)$.

Summary. To optimize with precision ϵ the empirical risk $f(\theta) = \frac{1}{n}\sum_{i=1}^n f_i(\theta)$, the computational cost is

Class	L-smooth convex functions	L-smooth & μ-strongly-convex functions
Number of iter. T_ϵ for ϵ precision	$O\left(\frac{L}{\epsilon}\right)$	$O\left(\kappa \ln\left(\frac{1}{\epsilon}\right)\right)$
Cost per iteration C	$O(nd)$	$O(nd)$
Total cost $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(nd\kappa \ln\left(\frac{1}{\epsilon}\right)\right)$

GD: Summary

First-order method **Deterministic**

Update
$\theta_{t+1} \leftarrow \theta_t - \eta \nabla f(\theta_t)$

Hyperparameters: η, T.
$\rightsquigarrow$ choose η as large as possible without oscillations.

Complexity/iter in ML:
$O(nd)$

Intuition:

- steepest descent direction
- local minimization of first-order approx
- global minimization of QUB for f smooth.

Theory:

- Rate for smooth and (strongly) convex functions


Impact of κ.


Impact of T.


Impact of η.

Let's recap! Wooclap quizz!

Outline

Motivation for Stochastic Gradient Descent.

We consider the ERM problem:

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

GD uses a *full gradients*, i.e. computes at each iteration:

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

which depends on the whole dataset (all observations are used).

For GD, the complexity per iteration on ML examples, $O(d \times n)$, is prohibitive. Think of $d \geqslant 10^6$, $n \geqslant 10^8$...

Motivation for Stochastic Gradient Descent.

We consider the ERM problem:

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

GD uses a *full gradients*, i.e. computes at each iteration:

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

which depends on the whole dataset (all observations are used).

For GD, the complexity per iteration on ML examples, $O(d \times n)$, is prohibitive. Think of $d \geqslant 10^6$, $n \geqslant 10^8$...

Idea: stochastic gradients: use only some (random) elements of the sum!

1. At iteration t, sample of batch $\mathcal{I}_t$ of b observations out of the n available. Define

$$g_t(\theta) = \frac{1}{b} \sum_{i \in \mathcal{I}_t} \nabla f_i(\theta).$$

2. g_t will be used instead of ∇f.
3. g_t is a random quantity.
4. Complexity $O(d \times b)$ at each iteration.

Remark: Goes back to "old" algorithms!

Stochastic Gradient Descent (SGD)

Algorithm : Stochastic gradient descent method (SGD)

Initialization: initial weight vector θ_0,
Hyperparameters: learning rate $(\eta_t)_{t\geqslant 0}$, number of steps T, batch-size b.

For $t \in \{1, \dots, T\}$ do

- Pick at random a subset $\mathcal{I}_t$ of $\{1, \dots, n\}$ with cardinal b.
- Compute

$$\theta_t = \theta_{t-1} - \eta_t \frac{1}{b} \sum_{i \in \mathcal{I}_t} \nabla f_i(\theta_{t-1})$$

Output: Return last θ_T, or the average $\bar{\theta}_T = \frac{1}{T}\sum_{t=1}^{T} w_t$.

Stochastic Gradient Descent (SGD)

Algorithm : Stochastic gradient descent method (SGD)

Initialization: initial weight vector θ_0,
Hyperparameters: learning rate $(\eta_t)_{t \geqslant 0}$, number of steps T, batch-size b.

For $t \in \{1, \dots, T\}$ do

- Pick at random a subset $\mathcal{I}_t$ of $\{1, \dots, n\}$ with cardinal b.
- Compute

$$\theta_t = \theta_{t-1} - \eta_t \frac{1}{b} \sum_{i \in \mathcal{I}_t} \nabla f_i(\theta_{t-1})$$

Output: Return last θ_T, or the average $\bar{\theta}_T = \frac{1}{T} \sum_{t=1}^{T} w_t$.

Choice of b ?

Stochastic Gradient Descent (SGD)

Algorithm : Stochastic gradient descent method (SGD)

Initialization: initial weight vector θ_0,
Hyperparameters: learning rate $(\eta_t)_{t\geqslant 0}$, number of steps T, batch-size b.

For $t \in \{1, \dots, T\}$ do

- Pick at random a subset $\mathcal{I}_t$ of $\{1, \dots, n\}$ with cardinal b.
- Compute

$$\theta_t = \theta_{t-1} - \eta_t \frac{1}{b} \sum_{i \in \mathcal{I}_t} \nabla f_i(\theta_{t-1})$$

Output: Return last θ_T, or the average $\bar{\theta}_T = \frac{1}{T}\sum_{t=1}^{T} w_t$.

Choice of b ?	Name	Gradient *oracle* g_t	Complexity
$b = 1$	*Pure* SGD	$g_t = \nabla f_{i_t}$ with $i_t \sim \mathcal{U}[\![1; n]\!]$	$O(d)$

Stochastic Gradient Descent (SGD)

Algorithm : Stochastic gradient descent method (SGD)

Initialization: initial weight vector θ_0,
Hyperparameters: learning rate $(\eta_t)_{t\geqslant 0}$, number of steps T, batch-size b.

For $t \in \{1, \dots, T\}$ do

- Pick at random a subset $\mathcal{I}_t$ of $\{1, \dots, n\}$ with cardinal b.
- Compute

$$\theta_t = \theta_{t-1} - \eta_t \frac{1}{b} \sum_{i \in \mathcal{I}_t} \nabla f_i(\theta_{t-1})$$

Output: Return last θ_T, or the average $\bar{\theta}_T = \frac{1}{T}\sum_{t=1}^{T} w_t$.

Choice of b ?	Name	Gradient *oracle* g_t	Complexity
$b = 1$	*Pure* SGD	$g_t = \nabla f_{i_t}$ with $i_t \sim \mathcal{U}[\![1; n]\!]$	$O(d)$
$1 < b < n$	*b-minibatch* SGD	$g_t = \frac{1}{b}\sum_{i \in \mathcal{I}_t} \nabla f_i$ with $\#\mathcal{I}_t = b$	$O(d \times b)$

Stochastic Gradient Descent (SGD)

Algorithm : Stochastic gradient descent method (SGD)

Initialization: initial weight vector θ_0,
Hyperparameters: learning rate $(\eta_t)_{t\geqslant 0}$, number of steps T, batch-size b.

For $t \in \{1, \dots, T\}$ do

- Pick at random a subset $\mathcal{I}_t$ of $\{1, \dots, n\}$ with cardinal b.
- Compute

$$\theta_t = \theta_{t-1} - \eta_t \frac{1}{b} \sum_{i \in \mathcal{I}_t} \nabla f_i(\theta_{t-1})$$

Output: Return last θ_T, or the average $\bar{\theta}_T = \frac{1}{T}\sum_{t=1}^{T} w_t$.

Choice of b ?	Name	Gradient *oracle* g_t	Complexity
$b = 1$	*Pure* SGD	$g_t = \nabla f_{i_t}$ with $i_t \sim \mathcal{U}[\![1; n]\!]$	$O(d)$
$1 < b < n$	*b-minibatch* SGD	$g_t = \frac{1}{b}\sum_{i \in \mathcal{I}_t} \nabla f_i$ with $\#\mathcal{I}_t = b$	$O(d \times b)$
$b = n$	*deterministic/full batch* GD	$g_t = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i$	$O(d \times n)$

Stochastic Gradient Descent (SGD)

Algorithm : Stochastic gradient descent method (SGD)

Initialization: initial weight vector θ_0,
Hyperparameters: learning rate $(\eta_t)_{t\geqslant 0}$, number of steps T, batch-size b.

For $t \in \{1, \dots, T\}$ do

- Pick at random a subset $\mathcal{I}_t$ of $\{1, \dots, n\}$ with cardinal b.
- Compute

$$\theta_t = \theta_{t-1} - \eta_t \frac{1}{b} \sum_{i \in \mathcal{I}_t} \nabla f_i(\theta_{t-1})$$

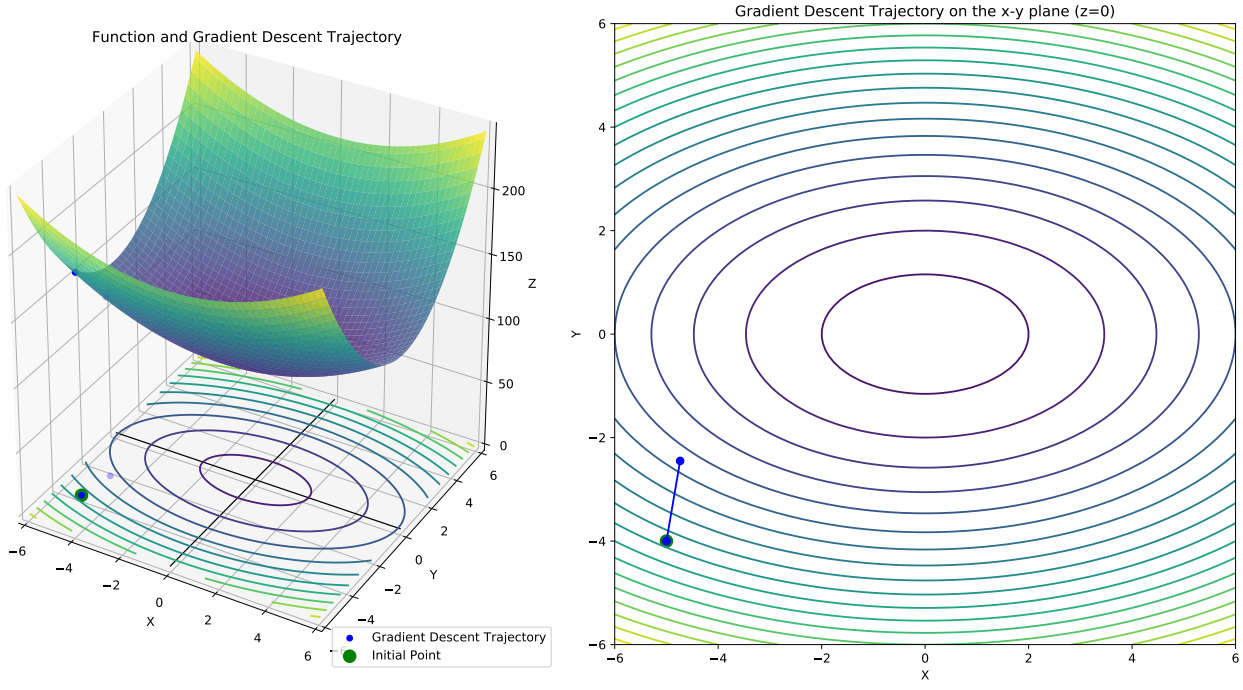
Output: Return last θ_T, or the average $\bar{\theta}_T = \frac{1}{T}\sum_{t=1}^{T} w_t$.

Choice of b ?	Name	Gradient *oracle* g_t	Complexity
$b = 1$	*Pure* SGD	$g_t = \nabla f_{i_t}$ with $i_t \sim \mathcal{U}[\![1; n]\!]$	$O(d)$
$1 < b < n$	*b-minibatch* SGD	$g_t = \frac{1}{b}\sum_{i \in \mathcal{I}_t} \nabla f_i$ with $\#\mathcal{I}_t = b$	$O(d \times b)$
$b = n$	*deterministic/full batch* GD	$g_t = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i$	$O(d \times n)$

- In practice, good choice of b is typically 32, 64, 128, 256
  $\rightsquigarrow$ benefits from parallelization of computation on GPU/CPU.
- For theoretical purposes, focus on $b = 1$, which is the simplest case and sufficient.

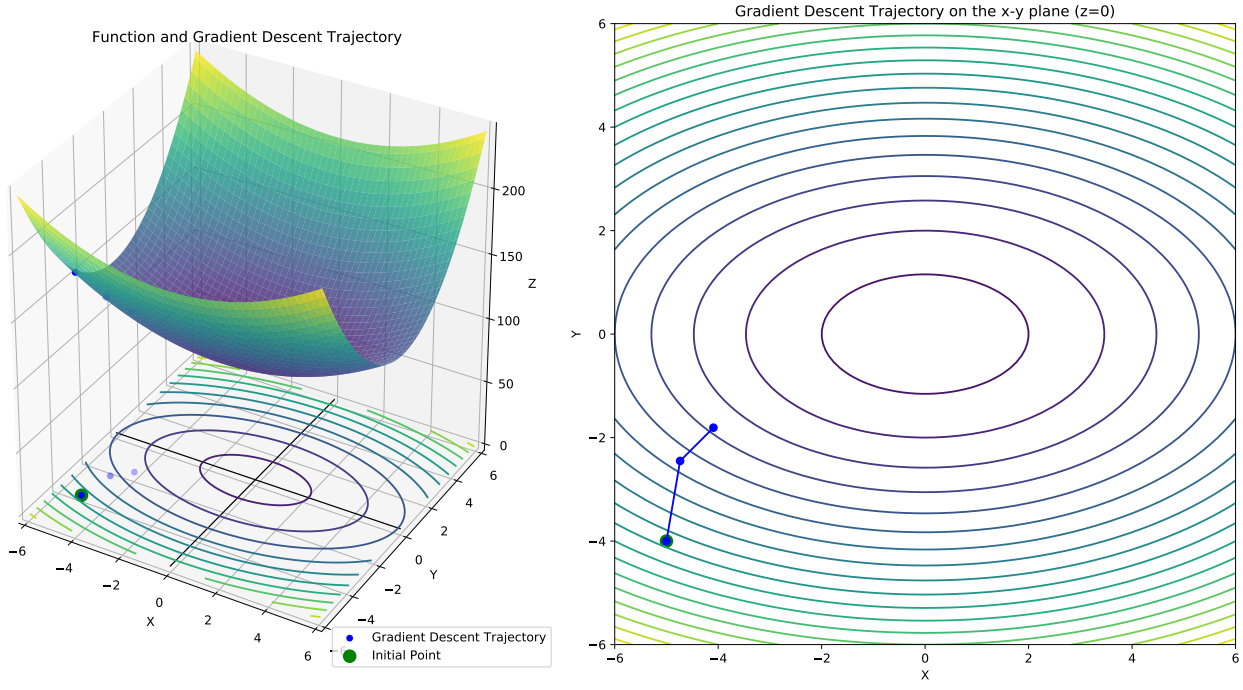
SGD: illustration, typical behavior, impact of the learning rate

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



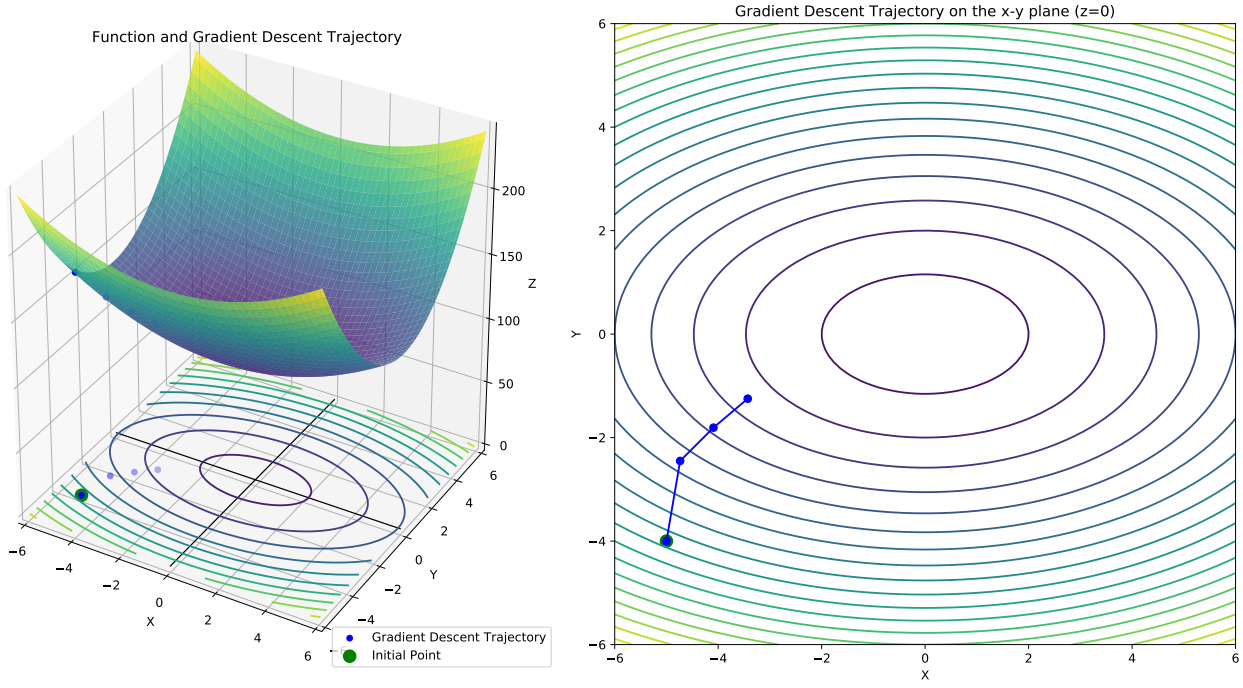
SGD: illustration, typical behavior, impact of the learning rate

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



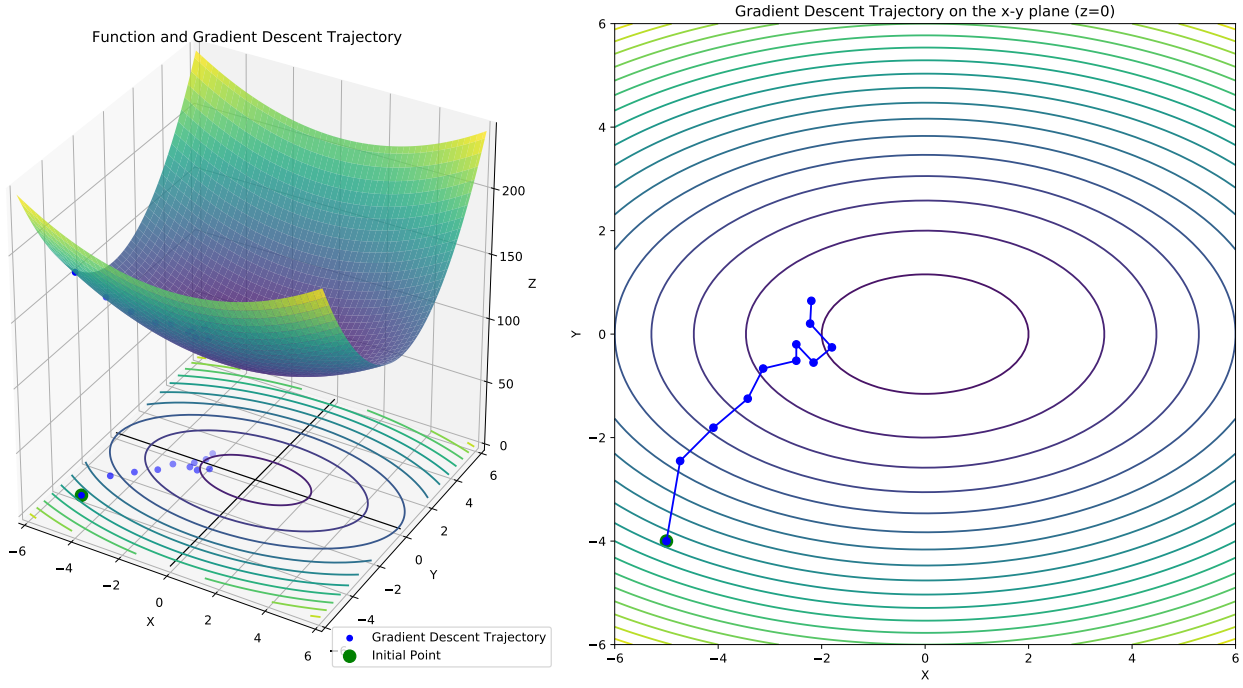
SGD: illustration, typical behavior, impact of the learning rate

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



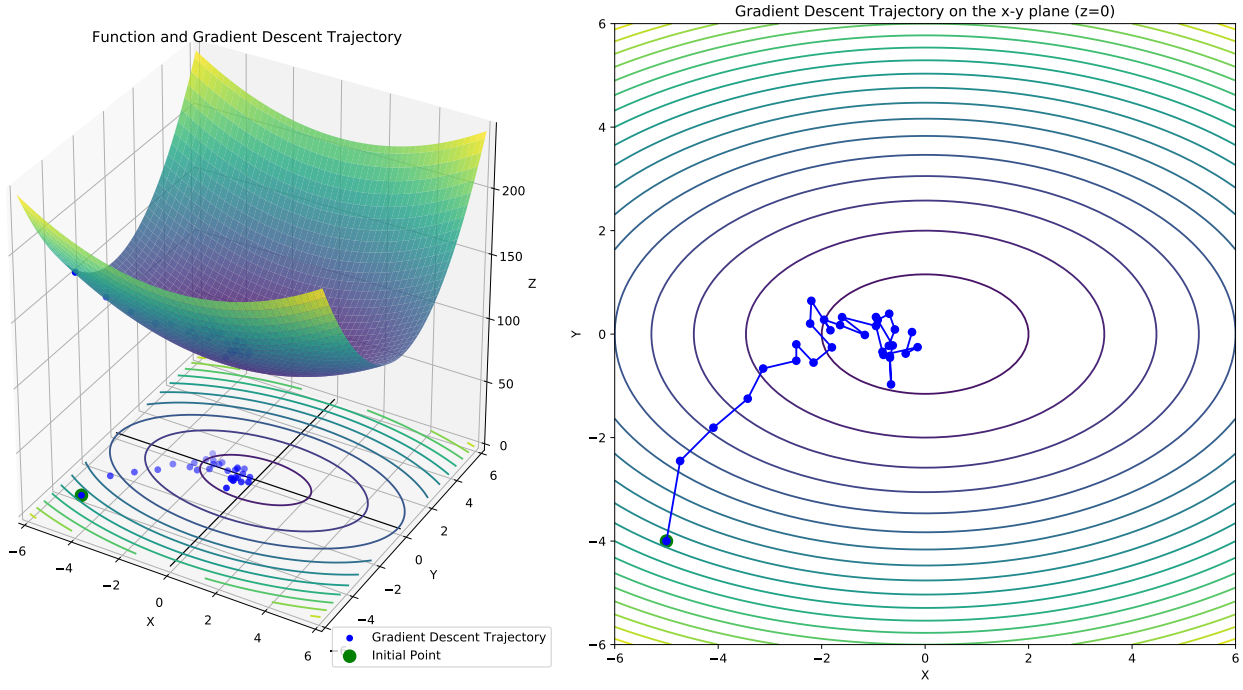
SGD: illustration, typical behavior, impact of the learning rate

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



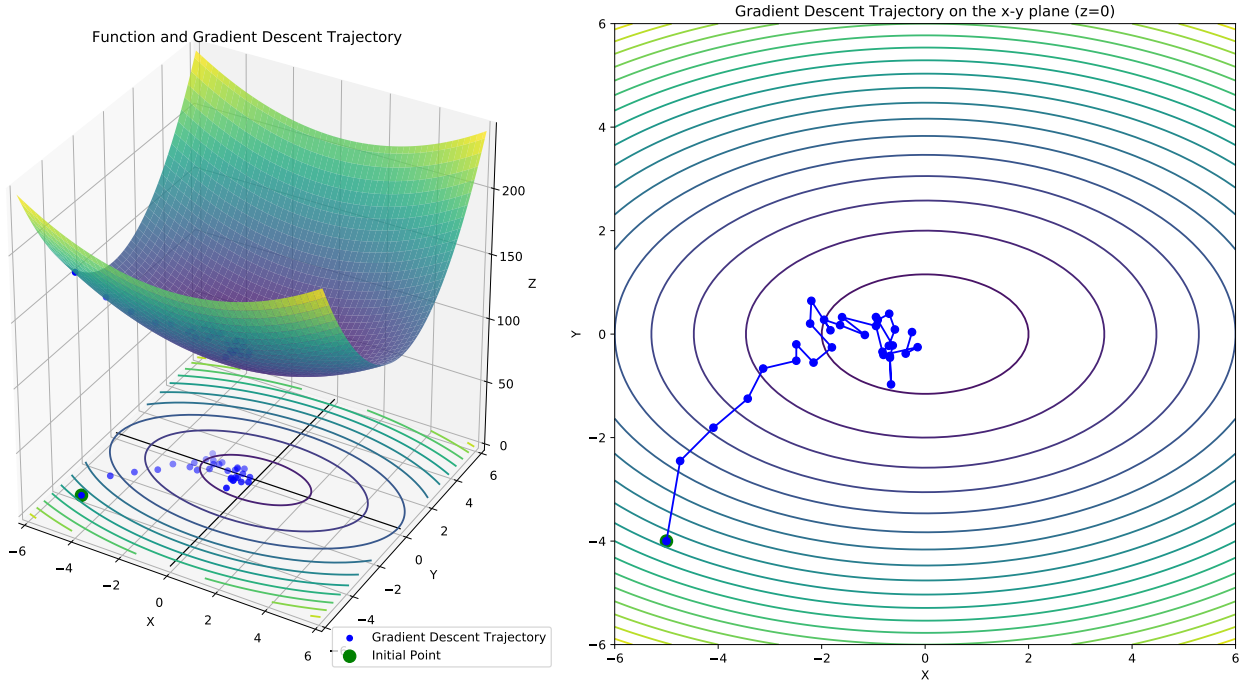
SGD: illustration, typical behavior, impact of the learning rate

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



SGD: illustration, typical behavior, impact of the learning rate

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.

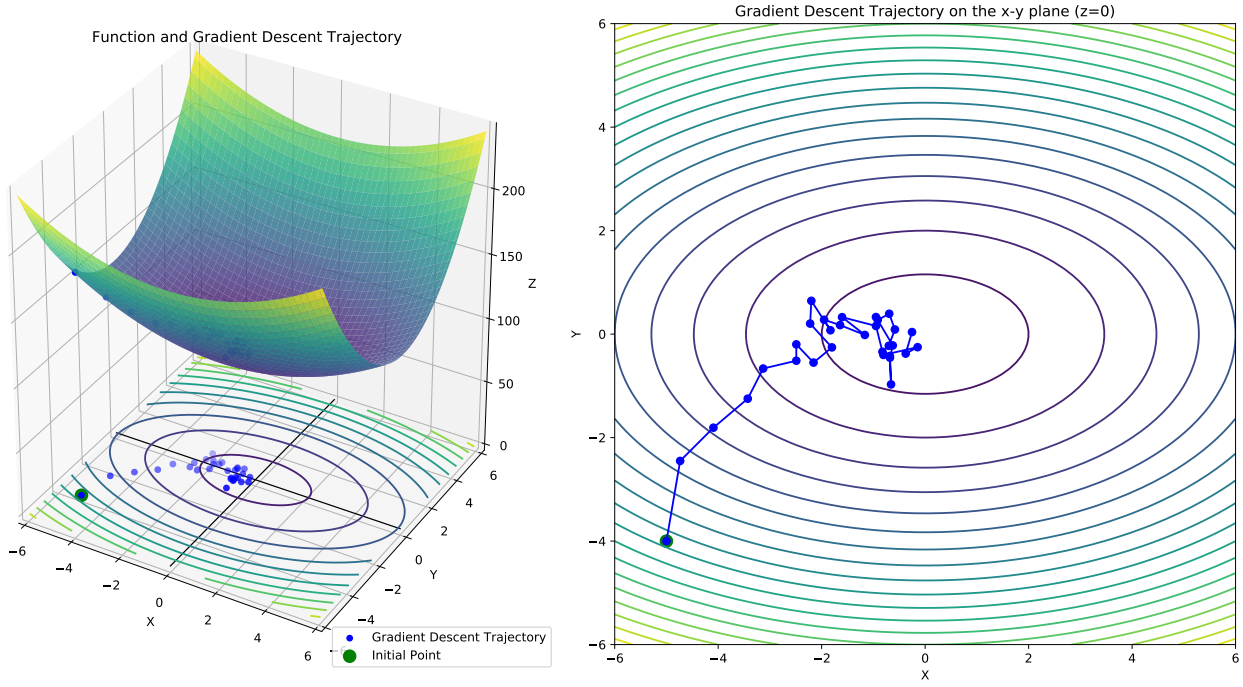


Fundamental tradeoff:

1. Impact of the initial condition (as for GD): using a **larger step** helps to forget faster the initial point.
2. Impact of the noise: using a **smaller step** makes the algorithm less sensitive to the noise.

SGD: illustration, typical behavior, impact of the learning rate

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.

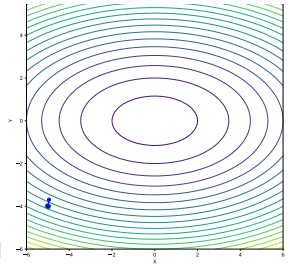
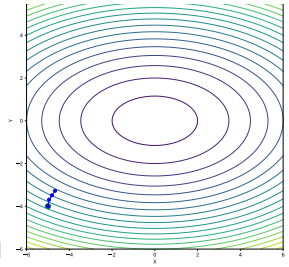
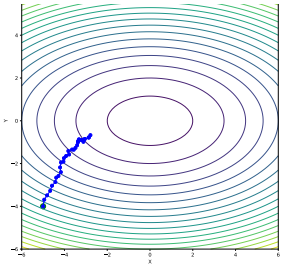


Fundamental tradeoff:

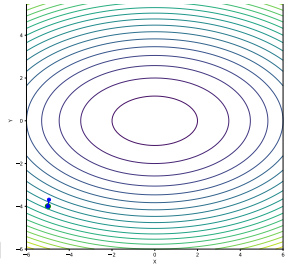
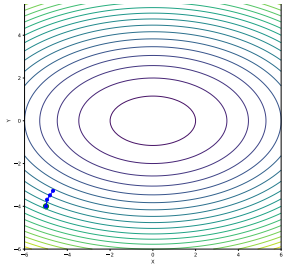
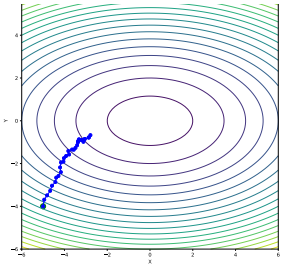
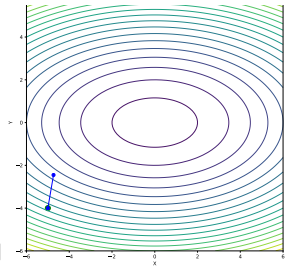
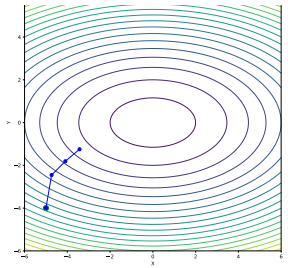
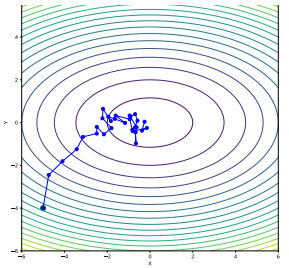
1. Impact of the initial condition (as for GD): using a **larger step** helps to forget faster the initial point.
2. Impact of the noise: using a **smaller step** makes the algorithm less sensitive to the noise.

The ideal step size corresponds to a balance between those two terms.

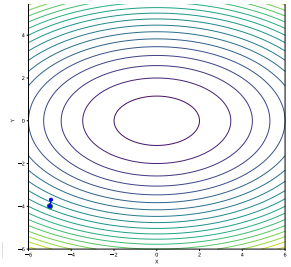
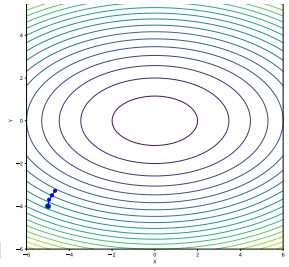
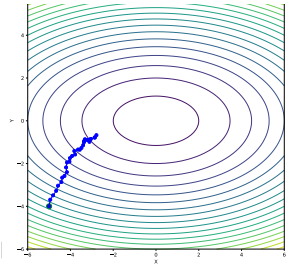
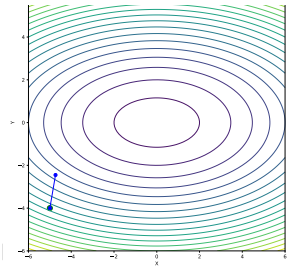
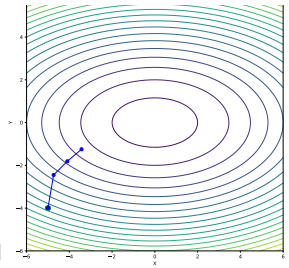
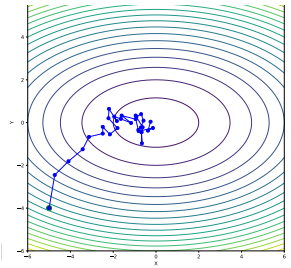
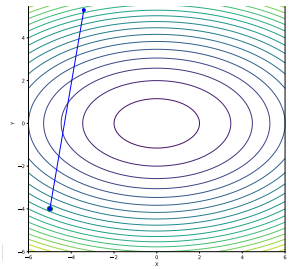
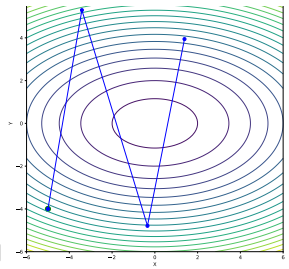
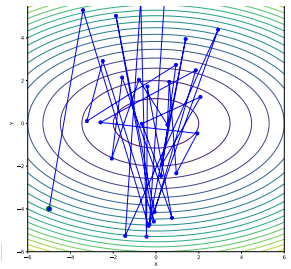
SGD: illustration, typical behavior, impact of the learning rate η

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence, as for GD. We do not forget the initial condition fast enough.

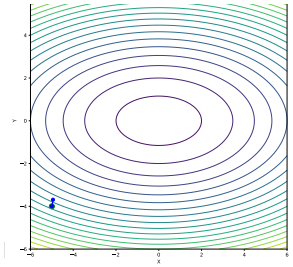
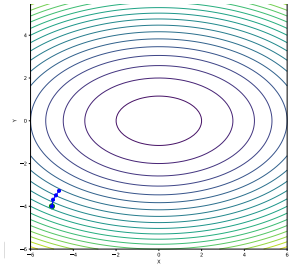
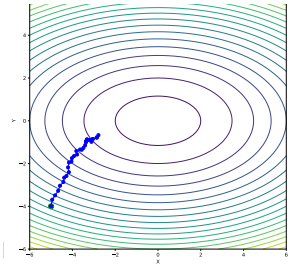
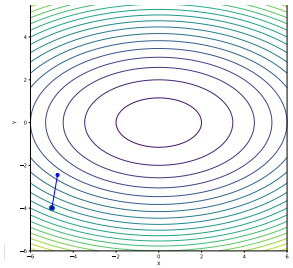
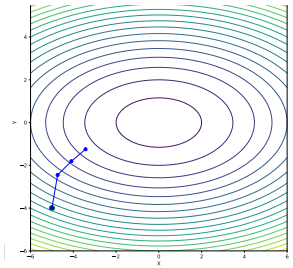
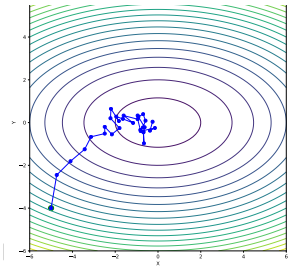
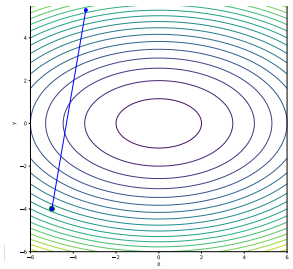
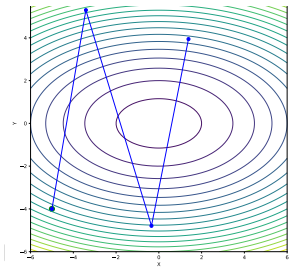
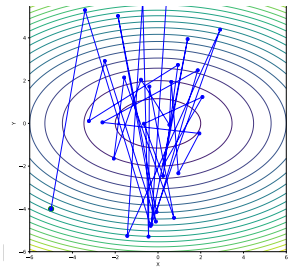
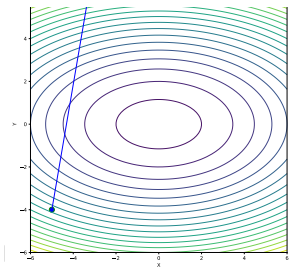
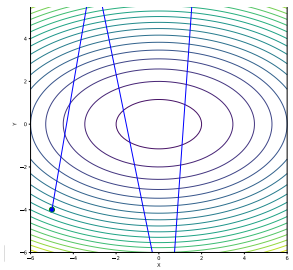
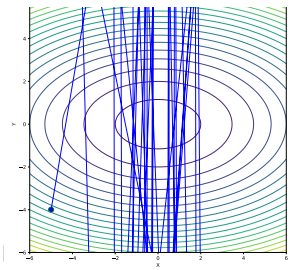
SGD: illustration, typical behavior, impact of the learning rate η

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence, as for GD. We do not forget the initial condition fast enough.
ok				→ reasonable convergence: we balance the noise and initial condition term

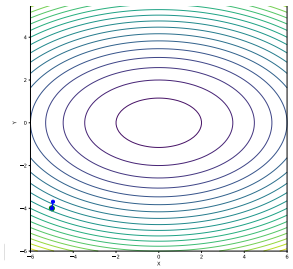
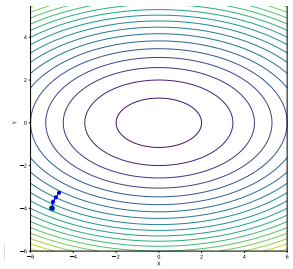
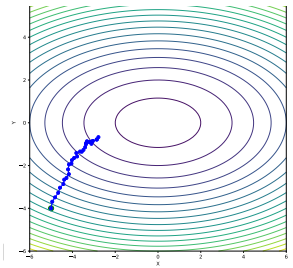
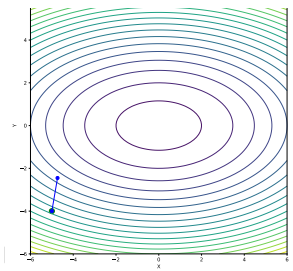
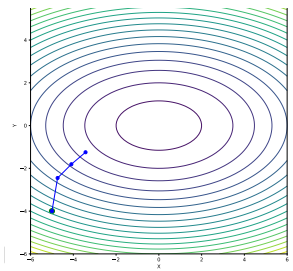
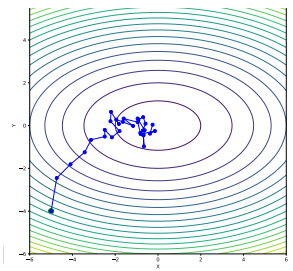
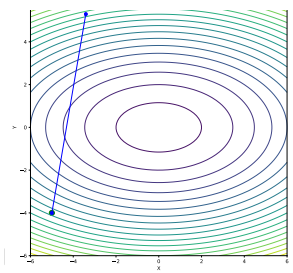
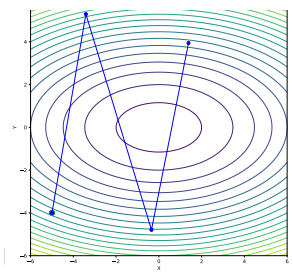
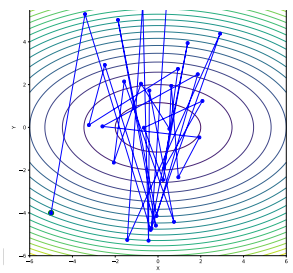
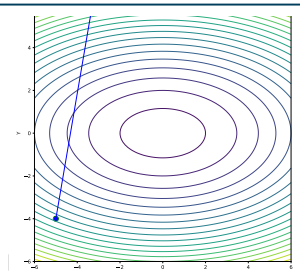
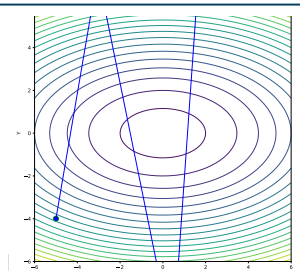
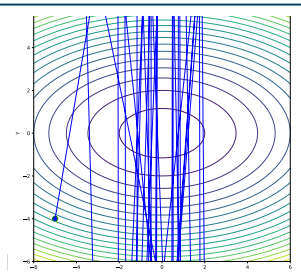
SGD: illustration, typical behavior, impact of the learning rate η

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence, as for GD. We do not forget the initial condition fast enough.
ok				→ reasonable convergence: we balance the noise and initial condition term
too large				→ noise dominates, although the step-size choice would be ok for GD, it is too large for SGD

SGD: illustration, typical behavior, impact of the learning rate η

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence, as for GD. We do not forget the initial condition fast enough.
ok				→ reasonable convergence: we balance the noise and initial condition term
too large				→ noise dominates, although the step-size choice would be ok for GD, it is too large for SGD
far too large				→ diverging oscillations, even for GD

SGD: illustration, typical behavior, impact of the learning rate η

η	$T = 1$	$T = 3$	$T = 30$	
too small				→ very slow convergence, as for GD. We do not forget the initial condition fast enough.
ok				→ reasonable convergence: we balance the noise and initial condition term
too large				→ noise dominates, although the step-size choice would be ok for GD, it is too large for SGD
far too large				→ diverging oscillations, even for GD

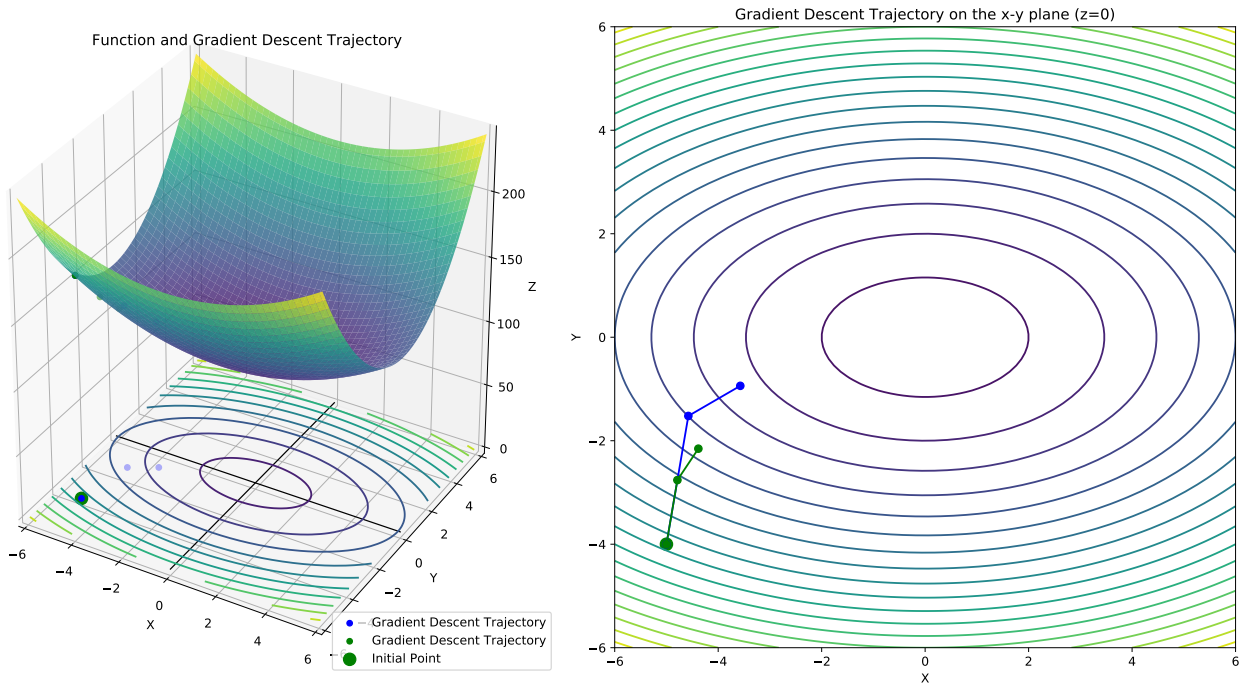
Conclusion: η needs to be much smaller than for GD, to encompass the impact of noise.

→ use a decaying sequence $\eta_t \propto \frac{1}{\sqrt{t}}$ *or*

→ use a step size that decreases with the total number of steps $\eta \propto \frac{1}{\sqrt{T}}$.

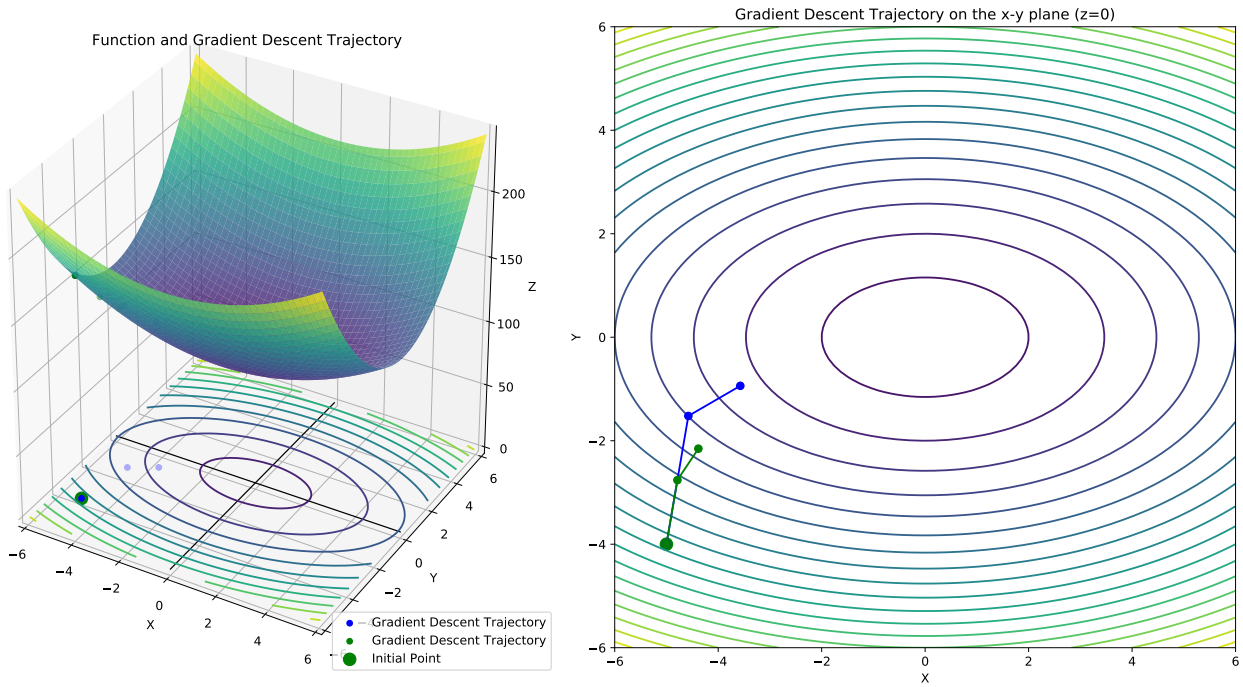
SGD: illustration, typical behavior, impact of averaging

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



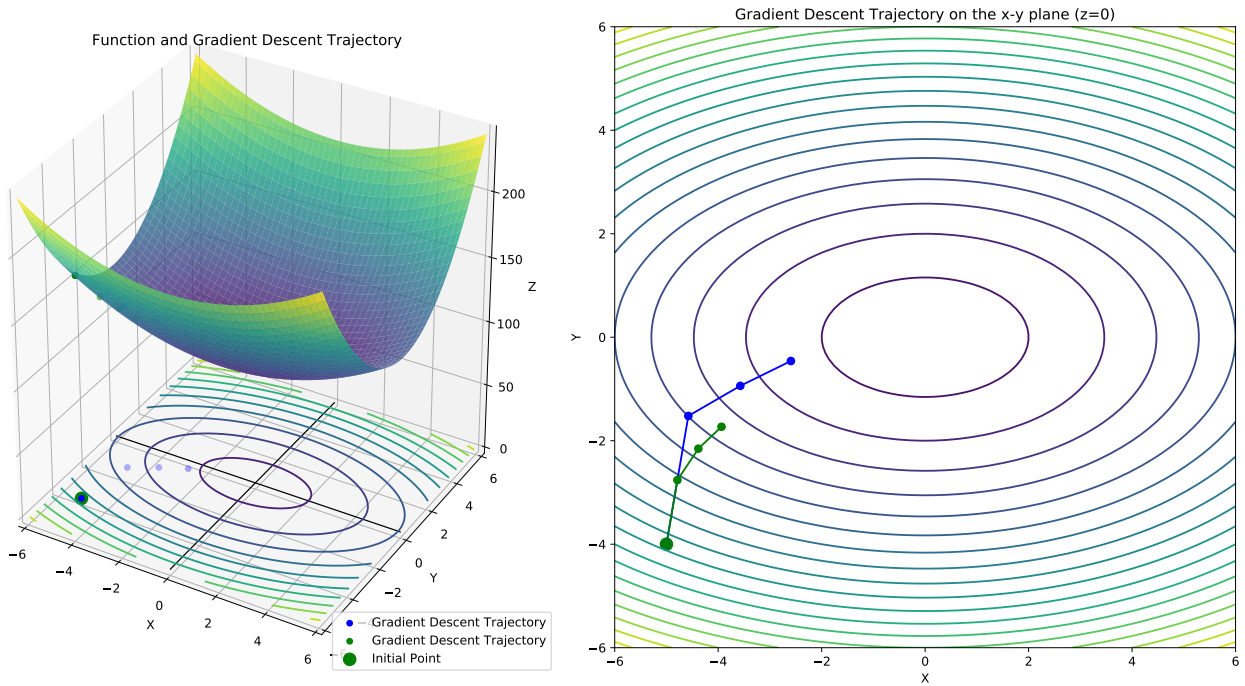
SGD: illustration, typical behavior, impact of averaging

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



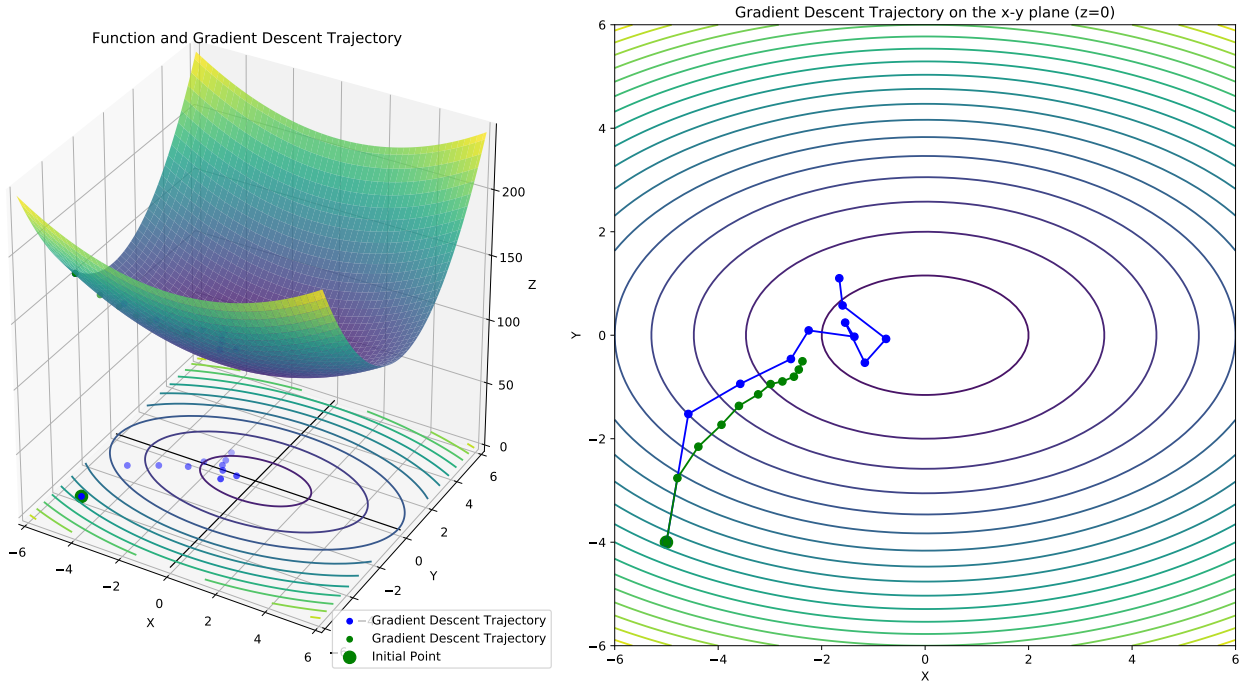
SGD: illustration, typical behavior, impact of averaging

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



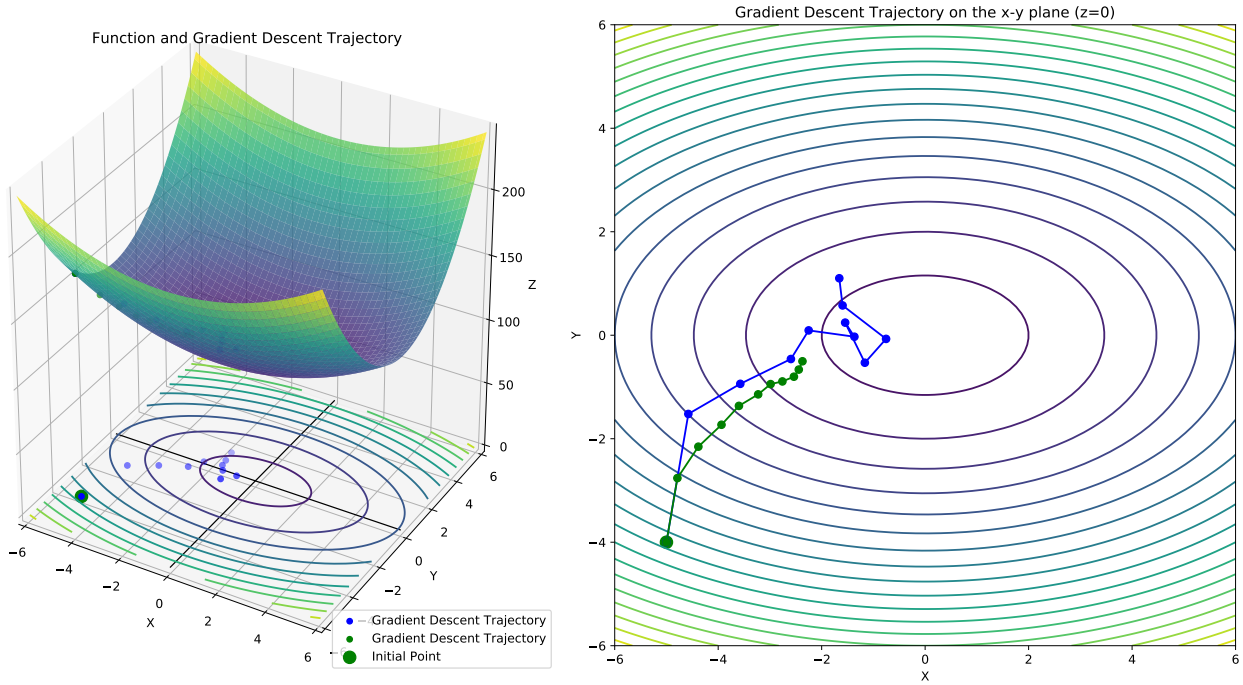
SGD: illustration, typical behavior, impact of averaging

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



SGD: illustration, typical behavior, impact of averaging

Behavior: follows a trajectory somehow similar to GD, but with a perturbed direction at each step.



Impact of averaging:

1. Reduces the impact of noise

Choice of hyperparameters, in practice

Convex case:

- Step-size: theoretically guided, typically decaying sequences.
- Averaging: often helpful from a theoretical standpoint.

In deep learning:

- Step-size: million dollar question!
  Tuning the step size is known to be difficult.
  $\rightarrow$ trial and error, schedulers, etc.
  Most common strategy: *reduce by a factor the step size when no progress is made.*
- Averaging: not used too much.

Question: theoretical guarantees to support intuition?

Theoretical grounds for SGD

If the batch is chosen **uniformly at random** and **independently from the past**, then g_t is, **on average**, ∇f.

Case $b = 1$: if i_t is chosen **uniformly** at random in $\{1, \dots, n\}$, independently form the past iterates, then

$$\begin{aligned}\mathbb{E}[g_t(\theta_{t-1})|\theta_{t-1}] &= \mathbb{E}[\nabla f_{i_t}(\theta_{t-1})|\theta_{t-1}] \\ &= \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\theta_{t-1}) \\ &= \nabla f(\theta_{t-1})\end{aligned}$$

Case $b > 1$: if $\mathcal{I}_t$ is chosen **uniformly** at random in $\{1, \dots, n\}$, with or without resampling points, independently form the past iterates, then

$$\begin{aligned}\mathbb{E}[g_t(\theta_{t-1})|\theta_{t-1}] &= \mathbb{E}[\frac{1}{b}\sum_{i\in\mathcal{I}_t} \nabla f_i(\theta_{t-1})|\theta_{t-1}] \\ &= \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\theta_{t-1}) \\ &= \nabla f(\theta_{t-1})\end{aligned}$$

$g_t(\theta_{t-1})$ is an **unbiased** but **noisy** estimate of the full gradient $\nabla f(\theta_{t-1})$.

Convergence of *pure SGD* algorithm ($b = 1$), to minimize $f = \frac{1}{n}\sum_{i=1}^{n} f_i$

Theorem : Convergence of SGD on smooth functions

If for all $i \in \{1, \dots, n\}$, f_i is convex and $L-$ smooth and there exists $\sigma^2 > 0$:

$$\mathrm{Var}(g_t(\theta)) = \mathbb{E}_{i \sim \mathcal{U}\{1,\dots,n\}}[\|\nabla f_i(\theta) - \nabla f(\theta)\|^2] \leqslant \sigma^2.$$

1. We need to control the *variance* of the *unbiased* stochastic gradient.

Remark: one can use a decreasing sequence $\eta_t = \|\theta_0 - \theta_\star\|/(\sigma\sqrt{t})$ for all $t \in \{1, \dots\}$.

Convergence of *pure SGD* algorithm ($b = 1$), to minimize $f = \frac{1}{n}\sum_{i=1}^{n} f_i$

Theorem : Convergence of SGD on smooth functions

If for all $i \in \{1, \dots, n\}$, f_i is convex and $L-$ smooth and there exists $\sigma^2 > 0$:

$$\mathrm{Var}(g_t(\theta)) = \mathbb{E}_{i\sim\mathcal{U}\{1,\dots,n\}}[\|\nabla f_i(\theta) - \nabla f(\theta)\|^2] \leqslant \sigma^2.$$

Then, with a constant learning rate sequence $\eta_t = \eta \leqslant \frac{1}{L}$ for all $t \leqslant T$:

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|^2}{\eta T} + \eta\sigma^2. \tag{1}$$

1. We need to control the *variance* of the *unbiased* stochastic gradient.

Remark: one can use a decreasing sequence $\eta_t = \|\theta_0 - \theta_\star\|/(\sigma\sqrt{t})$ for all $t \in \{1, \dots\}$.

Convergence of *pure SGD* algorithm ($b = 1$), to minimize $f = \frac{1}{n}\sum_{i=1}^{n} f_i$

Theorem : Convergence of SGD on smooth functions

If for all $i \in \{1, \dots, n\}$, f_i is convex and $L-$ smooth and there exists $\sigma^2 > 0$:

$$\mathrm{Var}(g_t(\theta)) = \mathbb{E}_{i\sim\mathcal{U}\{1,\dots,n\}}[\|\nabla f_i(\theta) - \nabla f(\theta)\|^2] \leqslant \sigma^2.$$

Then, with a constant learning rate sequence $\eta_t = \eta \leqslant \frac{1}{L}$ for all $t \leqslant T$:

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|^2}{\eta T} + \eta\sigma^2. \tag{1}$$

1. We need to control the *variance* of the *unbiased* stochastic gradient.
2. Eq. 1 confirms the existence of an initial condition / noise tradeoff.
   - The initial condition term $\frac{\|\theta_0-\theta_\star\|^2}{\eta T}$ is the *same* as for GD, decreases with η.
   - The extra noise term $\eta\sigma^2$ *increases* with η.
3. The control is for the averaged iterate $\bar{\theta}_T$.

Remark: one can use a decreasing sequence $\eta_t = \|\theta_0 - \theta_\star\|/(\sigma\sqrt{t})$ for all $t \in \{1, \dots\}$.

Convergence of *pure SGD* algorithm ($b = 1$), to minimize $f = \frac{1}{n}\sum_{i=1}^{n} f_i$

Theorem : Convergence of SGD on smooth functions

If for all $i \in \{1, \dots, n\}$, f_i is convex and $L-$ smooth and there exists $\sigma^2 > 0$:

$$\mathrm{Var}(g_t(\theta)) = \mathbb{E}_{i\sim\mathcal{U}\{1,\dots,n\}}[\|\nabla f_i(\theta) - \nabla f(\theta)\|^2] \leqslant \sigma^2.$$

Then, with a constant learning rate sequence $\eta_t = \eta \leqslant \frac{1}{L}$ for all $t \leqslant T$:

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|^2}{\eta T} + \eta\sigma^2. \tag{1}$$

Moreover setting $\eta = \|\theta_0 - \theta_\star\|/(\sigma\sqrt{T})$ to optimize this sum,

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}. \tag{2}$$

1. We need to control the *variance* of the *unbiased* stochastic gradient.
2. Eq. 1 confirms the existence of an initial condition / noise tradeoff.
   - The initial condition term $\frac{\|\theta_0-\theta_\star\|^2}{\eta T}$ is the *same* as for GD, decreases with η.
   - The extra noise term $\eta\sigma^2$ *increases* with η.
3. The control is for the averaged iterate $\bar{\theta}_T$.

Remark: one can use a decreasing sequence $\eta_t = \|\theta_0 - \theta_\star\|/(\sigma\sqrt{t})$ for all $t \in \{1, \dots\}$.

Convergence of *pure SGD* algorithm ($b = 1$), to minimize $f = \frac{1}{n}\sum_{i=1}^{n} f_i$

Theorem : Convergence of SGD on smooth functions

If for all $i \in \{1, \dots, n\}$, f_i is convex and $L-$ smooth and there exists $\sigma^2 > 0$:

$$\mathrm{Var}(g_t(\theta)) = \mathbb{E}_{i\sim\mathcal{U}\{1,\dots,n\}}[\|\nabla f_i(\theta) - \nabla f(\theta)\|^2] \leqslant \sigma^2.$$

Then, with a constant learning rate sequence $\eta_t = \eta \leqslant \frac{1}{L}$ for all $t \leqslant T$:

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{\|\theta_0 - \theta_\star\|^2}{\eta T} + \eta\sigma^2. \tag{1}$$

Moreover setting $\eta = \|\theta_0 - \theta_\star\|/(\sigma\sqrt{T})$ to optimize this sum,

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}. \tag{2}$$

1. We need to control the *variance* of the *unbiased* stochastic gradient.
2. Eq. 1 confirms the existence of an initial condition / noise tradeoff.
   - The initial condition term $\frac{\|\theta_0-\theta_\star\|^2}{\eta T}$ is the *same* as for GD, decreases with η.
   - The extra noise term $\eta\sigma^2$ *increases* with η.
3. The control is for the averaged iterate $\bar{\theta}_T$.
4. Eq. 2 give a good choice of step size and the corresponding rate.
   - The "optimal" step-size to minimize the bound decreases with T, scaling as $1/\sqrt{T}$.
   - The corresponding rate scales as $1/\sqrt{T}$.

Remark: one can use a decreasing sequence $\eta_t = \|\theta_0 - \theta_\star\|/(\sigma\sqrt{t})$ for all $t \in \{1, \dots\}$.

Let's recap! Wooclap quizz!

Outline

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD $(b = 1)$ rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD ($b = 1$) rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

$\rightarrow$ 1. For a given target ϵ

Method	GD	SGD
# of iter. T_ϵ for ϵ precision	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$.
Cost per iteration C	$O(nd)$	$O(d)$
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD ($b = 1$) rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

$\rightarrow$ 1. For a given target ϵ

Method	GD	SGD
# of iter. T_ϵ for ϵ precision	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$.
Cost per iteration C	$O(nd)$	$O(d)$
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

$\rightarrow$ The total complexity does not depend on n for SGD!

$\rightarrow$ For any given ϵ, the total complexity of SGD is smaller than the one of GD if n *is large enough.*

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD ($b = 1$) rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

$\rightarrow$ 1. For a given target ϵ

Method	GD	SGD
# of iter. T_ϵ for ϵ precision	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$.
Cost per iteration C	$O(nd)$	$O(d)$
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

$\rightarrow$ The total complexity does not depend on n for SGD!

$\rightarrow$ For any given ϵ, the total complexity of SGD is smaller than the one of GD if n *is large enough.*

$\rightarrow$ 2. If ϵ is fixed relatively to n?

Method	GD	SGD
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

For $\epsilon = 1/\sqrt{n}$

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD ($b = 1$) rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

$\rightarrow$ 1. For a given target ϵ

Method	GD	SGD
# of iter. T_ϵ for ϵ precision	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$.
Cost per iteration C	$O(nd)$	$O(d)$
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

$\rightarrow$ The total complexity does not depend on n for SGD!

$\rightarrow$ For any given ϵ, the total complexity of SGD is smaller than the one of GD if n *is large enough.*

$\rightarrow$ 2. If ϵ is fixed relatively to n?

Method	GD	SGD
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$
For $\epsilon = 1/\sqrt{n}$	$O\left(n^{3/2}d\right)$	

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD ($b = 1$) rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

$\rightarrow$ 1. For a given target ϵ

Method	GD	SGD
# of iter. T_ϵ for ϵ precision	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$.
Cost per iteration C	$O(nd)$	$O(d)$
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

$\rightarrow$ The total complexity does not depend on n for SGD!

$\rightarrow$ For any given ϵ, the total complexity of SGD is smaller than the one of GD if n *is large enough.*

$\rightarrow$ 2. If ϵ is fixed relatively to n?

Method	GD	SGD
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$
For $\epsilon = 1/\sqrt{n}$	$O\left(n^{3/2}d\right)$	$O(nd)$

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD ($b = 1$) rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

$\rightarrow$ 1. For a given target ϵ

Method	GD	SGD
# of iter. T_ϵ for ϵ precision	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$.
Cost per iteration C	$O(nd)$	$O(d)$
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

$\rightarrow$ The total complexity does not depend on n for SGD!

$\rightarrow$ For any given ϵ, the total complexity of SGD is smaller than the one of GD if n *is large enough.*

$\rightarrow$ 2. If ϵ is fixed relatively to n?

Method	GD	SGD
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$
For $\epsilon = 1/\sqrt{n}$	$O\left(n^{3/2}d\right)$	$O(nd)$
For $\epsilon = 1/n$	$O\left(n^2 d\right)$	$O\left(n^2 d\right)$

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD ($b = 1$) rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

$\rightarrow$ 1. For a given target ϵ

Method	GD	SGD
# of iter. T_ϵ for ϵ precision	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$.
Cost per iteration C	$O(nd)$	$O(d)$
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

$\rightarrow$ The total complexity does not depend on n for SGD!

$\rightarrow$ For any given ϵ, the total complexity of SGD is smaller than the one of GD if n *is large enough.*

$\rightarrow$ 2. If ϵ is fixed relatively to n?

Method	GD	SGD
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$
For $\epsilon = 1/\sqrt{n}$	$O\left(n^{3/2}d\right)$	$O\left(nd\right)$
For $\epsilon = 1/n$	$O\left(n^2d\right)$	$O\left(n^2d\right)$
For $\epsilon = 1/n^2$	$O\left(n^3d\right)$	$O\left(n^4d\right)$

Comparison of GD and SGD - Class of L-smooth convex functions.

SGD ($b = 1$) rate

$$\mathbb{E}\left[f\left(\bar{\theta}_T\right)\right] - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|\sigma}{\sqrt{T}}.$$

$\Rightarrow$ Optimization complexity

$$T_\epsilon = \frac{4\|\theta_0 - \theta_\star\|^2\sigma^2}{\epsilon^2}.$$

$\rightarrow$ 1. For a given target ϵ

Method	GD	SGD
# of iter. T_ϵ for ϵ precision	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon^2}\right)$.
Cost per iteration C	$O(nd)$	$O(d)$
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$

$\rightarrow$ The total complexity does not depend on n for SGD!

$\rightarrow$ For any given ϵ, the total complexity of SGD is smaller than the one of GD if n *is large enough.*

$\rightarrow$ 2. If ϵ is fixed relatively to n?

Method	GD	SGD
Total complexity $= T_\epsilon \times C$	$O\left(\frac{nd}{\epsilon}\right)$	$O\left(\frac{d}{\epsilon^2}\right)$
For $\epsilon = 1/\sqrt{n}$	$O\left(n^{3/2}d\right)$	$O\left(nd\right)$
For $\epsilon = 1/n$	$O\left(n^2d\right)$	$O\left(n^2d\right)$
For $\epsilon = 1/n^2$	$O\left(n^3d\right)$	$O\left(n^4d\right)$

$\rightarrow$ SGD is faster than GD when aiming for a moderate precision w.r.t. n!

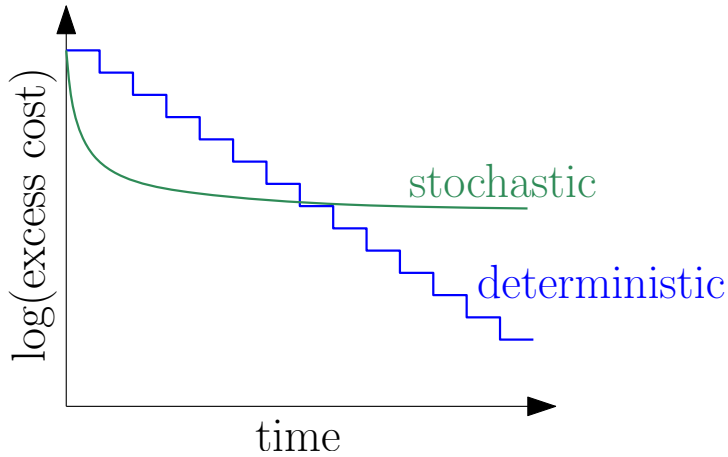
$\rightarrow$ For a high precision w.r.t. n, GD has a lower complexity.

$\rightarrow$ Recall that in ML the typical precision needed is *moderate*.

Comparison of convergence : SGD vs GD

Which one to choose?

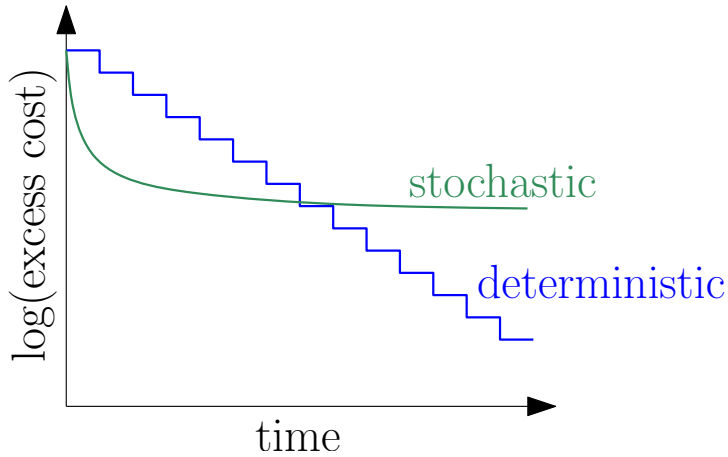
1. Depends on the precision we want.



Comparison of convergence : SGD vs GD

Which one to choose?

1. Depends on the precision we want.



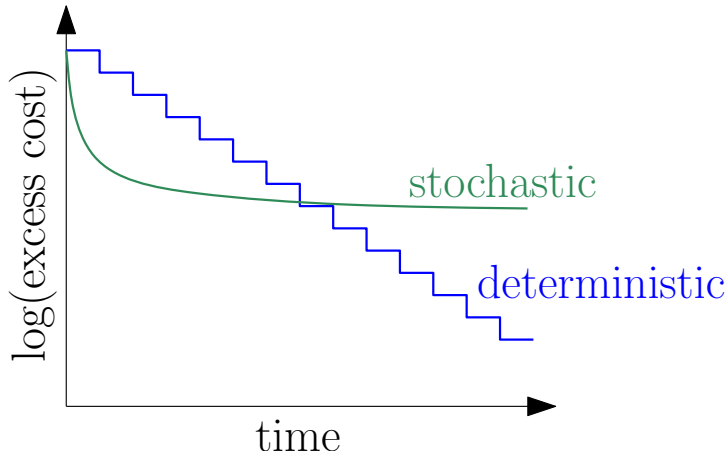
Example: non strongly convex case.

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.



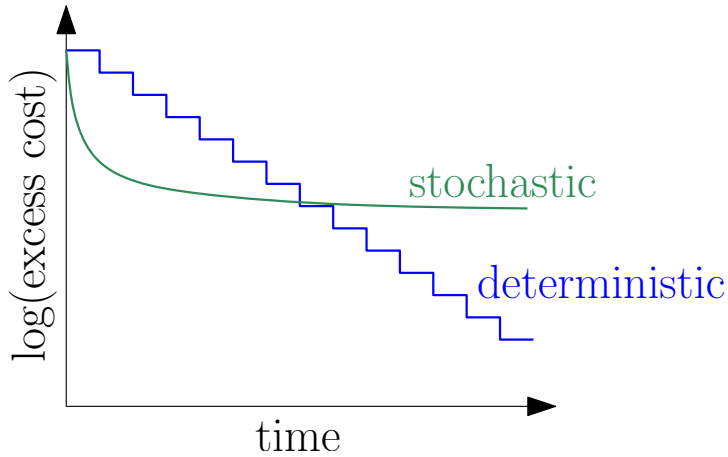
   Example: non strongly convex case.

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).

   Same comparison can be made, with the same conclusions, - on the class of L-smooth μ strongly-convex functions (left as an exercise).

Comparison of GD and SGD .

$\rightarrow$ Overall, SGD has a lower computational complexity when

- n is large
  or
- for ϵ moderate w.r.t. n.

In other words:

- SGD is extremely fast in the early iterations (first few (1-10) passes on the data)
- But it fails to converge accurately to the minimum

$\rightsquigarrow$ GD vs SGD – Which one to choose?

Comparison of GD and SGD .

→ Overall, SGD has a lower computational complexity when

- n is large
  or
- for ϵ moderate w.r.t. n.

In other words:

- SGD is extremely fast in the early iterations (first few (1-10) passes on the data)
- But it fails to converge accurately to the minimum

⇝ GD vs SGD – Which one to choose?

There is no uniformly best algorithm! The choice between SGD and GD depends on the context!

But, as a moderate precision is often enough in ML and n typically very large, SGD is always preferred to GD in ML applications.

SGD: Summary

First-order method **Stochastic**

Update

$$\theta_t \leftarrow \theta_{t-1} - \eta g_t(\theta_{t-1})$$

with g_t unbiased gradient oracle based on b observations

Hyperparameters: $(\eta_t)_t, T, b$.

$\rightsquigarrow$ choose η decreasing of small, to balance noise and optimization.

$\rightsquigarrow$ choose b small to limit complexity, typically 32 to 256.

Complexity/iter in ML:

$$O(b \times d)$$

Intuition & motivation:

Challenge: n large $\rightarrow$ Solution: stochastic alg.

Behavior

- noise - initial condition tradeoff
- averaging helps reduce the noise.

Comparison to GD.

Lower complexity IF we aim for moderate precision or n is large.

Theory:

- Rate for smooth and (strongly) convex functions
- 💡 Impact of T.
- 💡 Impact of (η_t).

$\rightarrow$ SGD is the building block of all ML algorithms

Let's recap! Wooclap quizz!

Outline

1. Gradient descent method, stochastic gradient method

2. Extensions of GD and SGD
   - Momentum and Acceleration
   - Variance reduced methods for ERM
   - Second-order algorithms

Roadmap of Part 2

We are going to study 5 directions into which GD and SGD can be improved.

1. **Variance reduction.**
   Hybrid methods between GD and SGD: computational cost of SGD, convergence of GD!
2. **Accelerated methods.**
   Builds on GD to improve the convergence for poorly conditioned functions (κ very large).
3. **Second order methods.**
   Whenever the dimension is not too large, use also the Hessian matrix information.
4. **Coordinate dependent-step size.**
   To take into account multiple possible scales, we use a different step for each coordinate.
5. **Adaptivity.**
   The size of the step-sizes depends on the observed values of the function, gradients, etc., not a predefined schedule.

Outline

1. Gradient descent method, stochastic gradient method

2. Extensions of GD and SGD
   - Momentum and Acceleration
     - Motivation and algorithms
     - Theoretical results and insights
     - Summary
   - Variance reduced methods for ERM
   - Second-order algorithms

Acceleration / Momentum algorithms

Setup: deterministic (at first, algorithms behave well also in stochastic regimes).

Goal: taking into account the previous updates as *additional velocity*.

→  If I have been moving *consistently* in one direction over the few last updates, I should move *faster* in that direction.

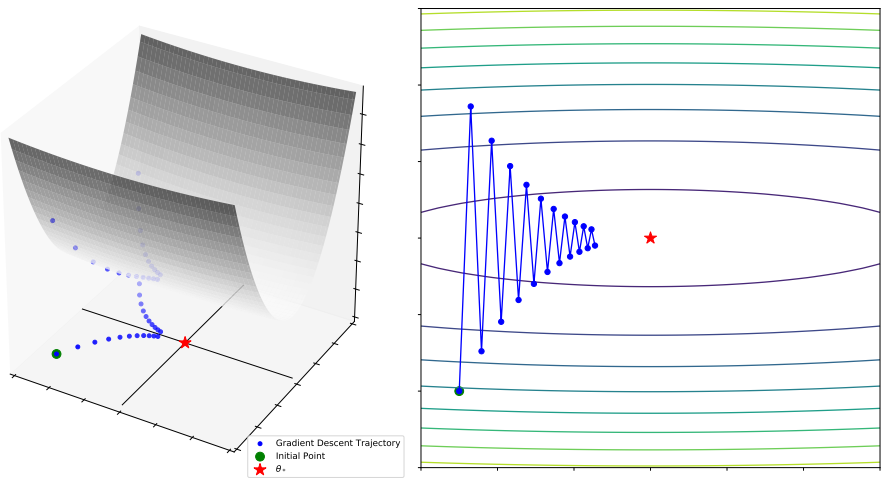
Acceleration / Momentum algorithms

Setup: deterministic (at first, algorithms behave well also in stochastic regimes).

Goal: taking into account the previous updates as *additional velocity*.

→  If I have been moving *consistently* in one direction over the few last updates, I should move *faster* in that direction.

Gradient Descent



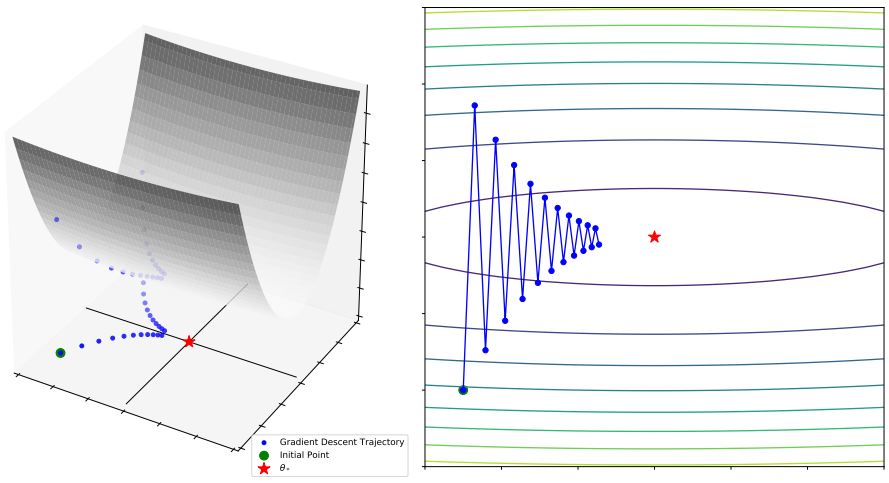
Acceleration / Momentum algorithms

Setup: deterministic (at first, algorithms behave well also in stochastic regimes).

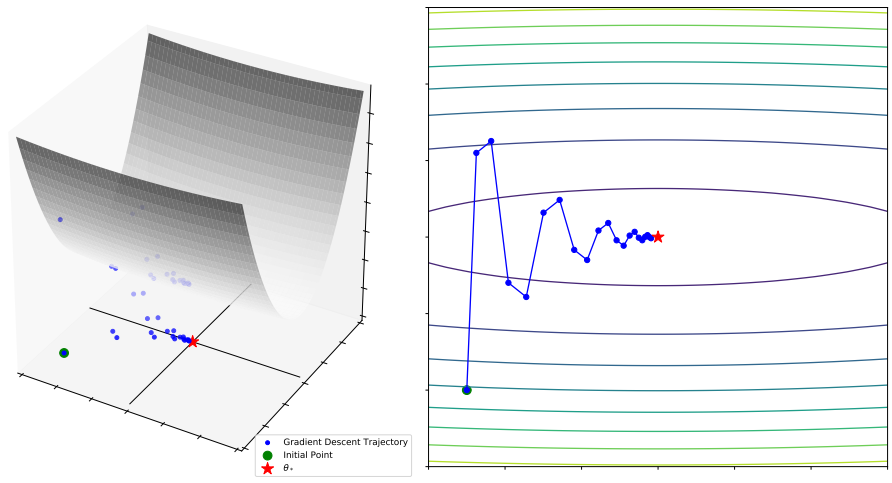
Goal: taking into account the previous updates as *additional velocity*.

→  If I have been moving *consistently* in one direction over the few last updates, I should move *faster* in that direction.

Gradient Descent



Accelerated Gradient Descent



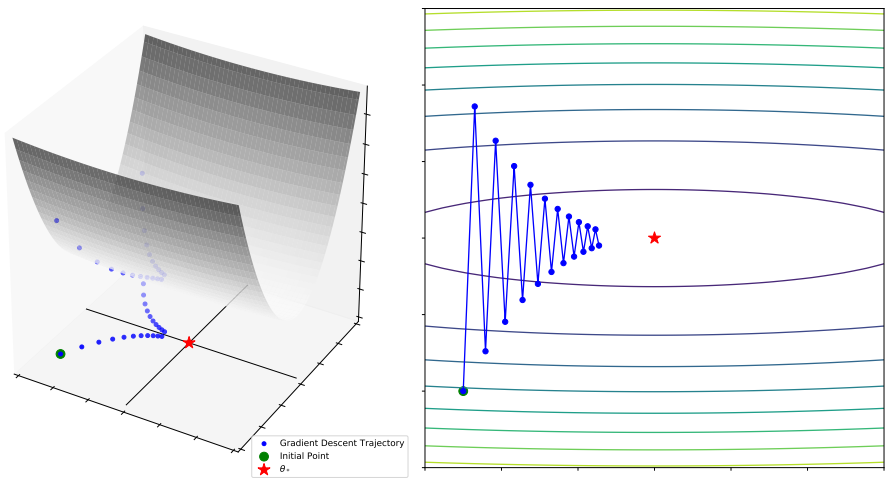
Acceleration / Momentum algorithms

Setup: deterministic (at first, algorithms behave well also in stochastic regimes).

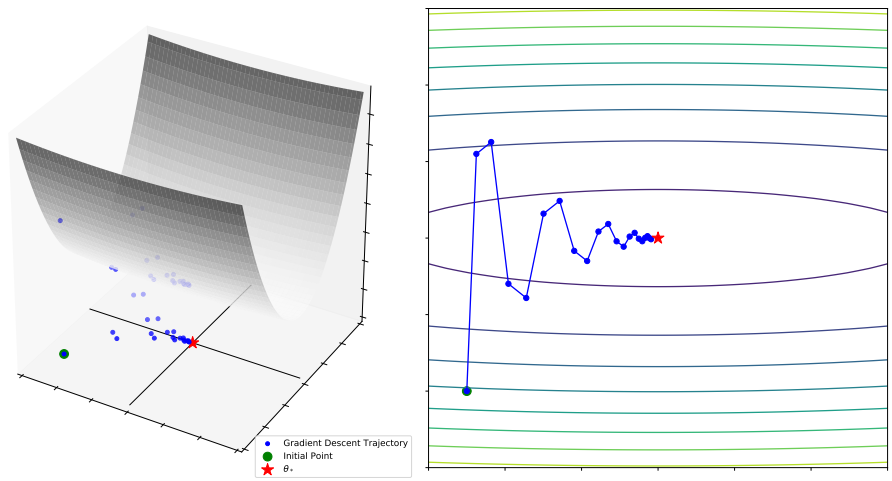
Goal: taking into account the previous updates as *additional velocity*.

→  If I have been moving *consistently* in one direction over the few last updates, I should move *faster* in that direction.

Gradient Descent



Accelerated Gradient Descent



1. Physics inspired approach: the GD update has no "memory": even if we have moved consistently in a direction, we do not *accelerate*.
   → rolling ball analogy.
2. Helpful to improve convergence for *flat areas* (saddle points, bad conditioning), and *local minimum*.

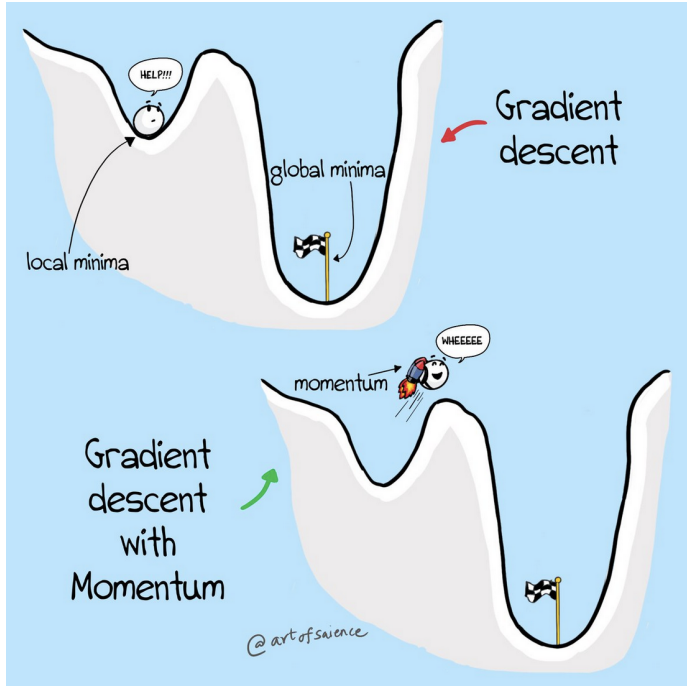


Figure: Classical illustration of HB method for local minima - Source: Twitter @DSaience

Polyak's momentum algorithm (1964) - Heavy ball (HB) method

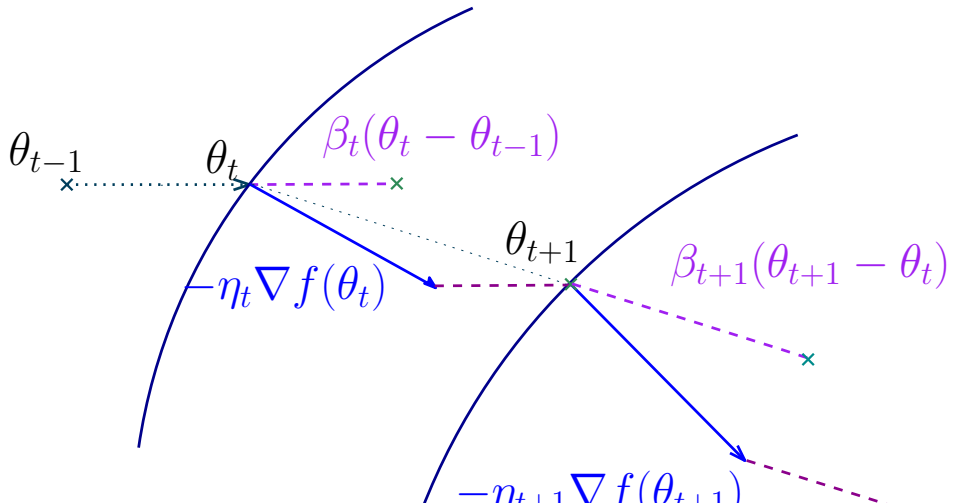
We add to the gradient update a fraction of the difference between the last two iterates, $\beta(\theta_t - \theta_{t-1})$, with $\beta \simeq 1$ typically.

Polyak's momentum algorithm (1964) - Heavy ball (HB) method

We add to the gradient update a fraction of the difference between the last two iterates, $\beta(\theta_t - \theta_{t-1})$, with $\beta \simeq 1$ typically.

Algorithm : Polyak's momentum algorithm - a.k.a. Heavy ball method

Hyperparameters: # steps T, learning rate $(\eta_t)_t > 0$, momentum $(\beta_t)_t \in [0, 1]$



Polyak's momentum algorithm (1964) - Heavy ball (HB) method

We add to the gradient update a fraction of the difference between the last two iterates, $\beta(\theta_t - \theta_{t-1})$, with $\beta \simeq 1$ typically.

Algorithm : Polyak's momentum algorithm - a.k.a. Heavy ball method

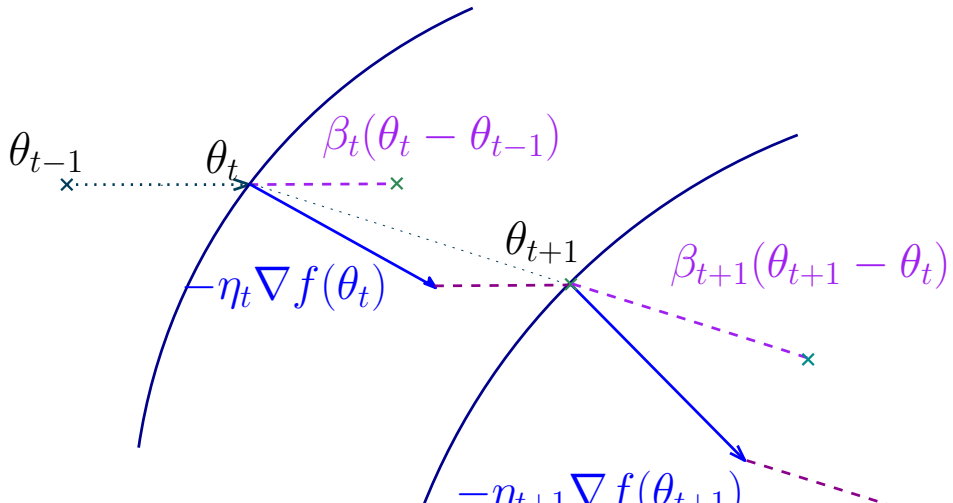
Hyperparameters: # steps T, learning rate $(\eta_t)_t > 0$, momentum $(\beta_t)_t \in [0, 1]$

Initialization: initial weight vector $\theta_1 = \theta_0$

For $t = 1, 2, \ldots, T$ do

$$\theta_{t+1} = \theta_t - \underbrace{\eta_t \nabla f(\theta_t)}_{\text{GD step}} + \underbrace{\beta_t(\theta_t - \theta_{t-1})}_{\text{momentum step / velocity}}$$

Output: Return last θ_T.



Nesterov's Acceleration (NAG, 1983)

Same idea but we first apply the momentum step, *then* take the gradient.

Algorithm : Nesterov accelerated GD

Hyperparameters: # steps T, learning rate $(\eta_t)_t > 0$, momentum $(\beta_t)_t \in [0, 1]$

Initialization: initial weight vector $\theta_1 = \theta_0$.

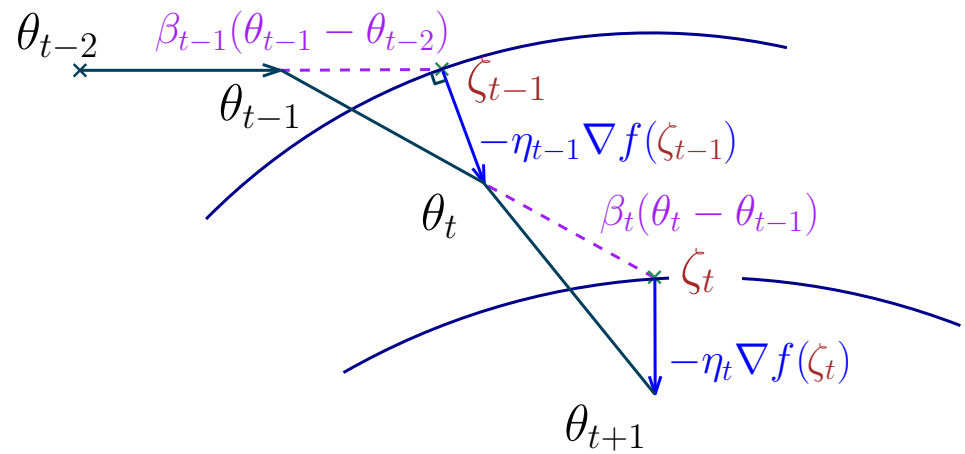
For $t = 1, 2, \dots, T$ do

$$\theta_{t+1} \leftarrow \theta_t + \underbrace{\beta_t(\theta_t - \theta_{t-1})}_{\text{momentum step}} - \underbrace{\eta_t \nabla f(\theta_t + \beta_t(\theta_t - \theta_{t-1}))}_{\text{GD step at } \textit{look-ahead} \text{ point}}$$

Output: Return last θ_T.

Alternative 2-steps description: we introduce the look-ahead point as (ζ_t)

$$\begin{cases} \zeta_t & \leftarrow & \theta_t + \beta_t(\theta_t - \theta_{t-1}) \\ \theta_{t+1} & \leftarrow & \zeta_t - \eta_t \nabla f(\zeta_t) \end{cases}$$

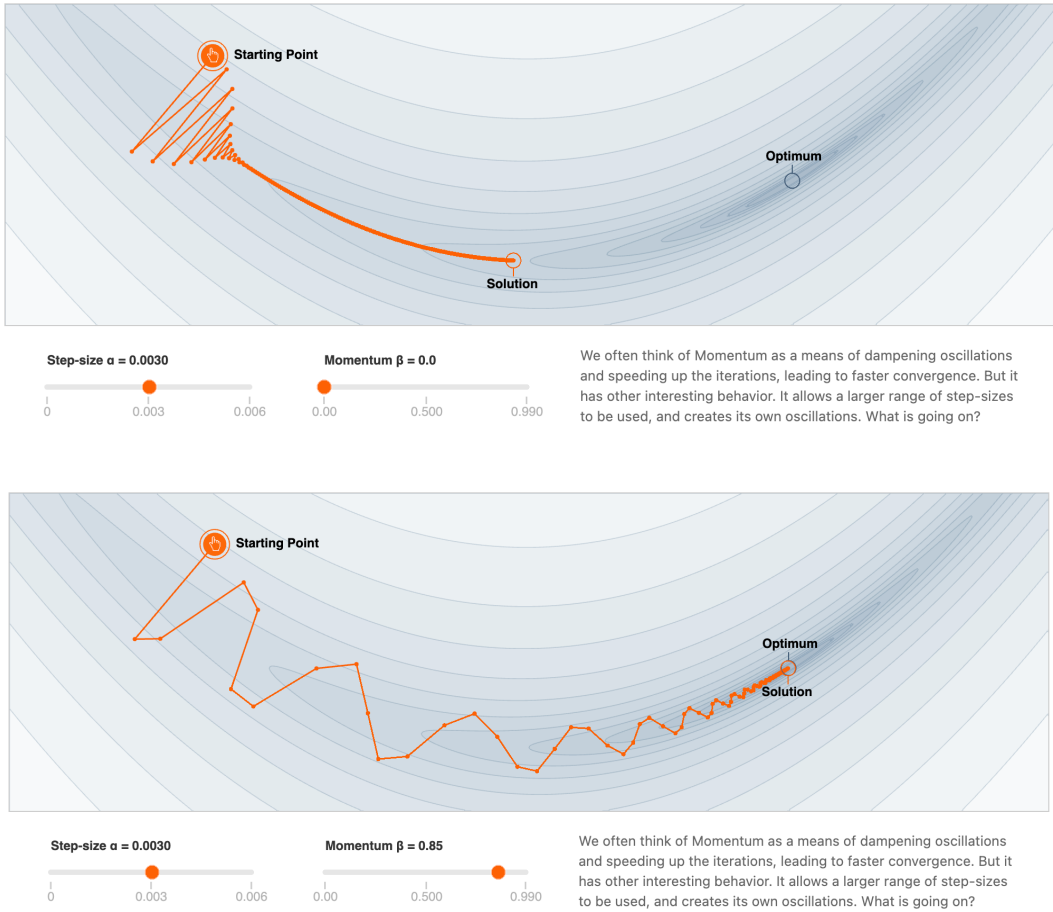


Accelerated methods: more visualization

Polyak Heavy-Ball and Nesterov accelerated gradient descent are both referred to as *accelerated methods* or *momentum methods*.

Interactive visualization with tunable η, β, non convex function,

$\rightarrow$ https://distill.pub/2017/momentum/



Same intuition: directions in which we accelerate consistent progress.

Rate of convergence of Nesterov Accelerated Gradient (NAG) - smooth case

Theorem : NAG rate of convergence - smooth convex functions

If f is L-smooth and convex with a minimum at $\theta_\star$. Then, if for all t, $\beta_{t+1} = \frac{t}{t+3}$, $\eta \leqslant \frac{1}{L}$

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|_2^2}{\eta(t+1)^2}.$$

Rate of convergence of Nesterov Accelerated Gradient (NAG) - smooth case

Theorem : NAG rate of convergence - smooth convex functions

If f is L-smooth and convex with a minimum at $\theta_\star$. Then, if for all t, $\beta_{t+1} = \frac{t}{t+3}$, $\eta \leqslant \frac{1}{L}$

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|_2^2}{\eta(t+1)^2}.$$

Observations:

1. We use a constant step size η, but an increasing momentum parameter $\beta_t \to 1$.

Rate of convergence of Nesterov Accelerated Gradient (NAG) - smooth case

Theorem : NAG rate of convergence - smooth convex functions

If f is L-smooth and convex with a minimum at $\theta_\star$. Then, if for all t, $\beta_{t+1} = \frac{t}{t+3}$, $\eta \leqslant \frac{1}{L}$

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|_2^2}{\eta(t+1)^2}.$$

Observations:

1. We use a constant step size η, but an increasing momentum parameter $\beta_t \to 1$.
2. **Comparison to GD.** With $\eta = \frac{1}{L}$
   - For GD, we had:
   $$f(\theta_t) - f(\theta_\star) \leqslant \frac{L\|\theta_0 - \theta_\star\|_2^2}{t}.$$
   - For NAG, we have:
   $$f(\theta_t) - f(\theta_\star) \leqslant \frac{2L\|\theta_0 - \theta_\star\|_2^2}{(t+1)^2}.$$

Rate of convergence of Nesterov Accelerated Gradient (NAG) - smooth case

Theorem : NAG rate of convergence - smooth convex functions

If f is L-smooth and convex with a minimum at $\theta_\star$. Then, if for all t, $\beta_{t+1} = \frac{t}{t+3}$, $\eta \leqslant \frac{1}{L}$

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{2\|\theta_0 - \theta_\star\|_2^2}{\eta(t+1)^2}.$$

Observations:

1. We use a constant step size η, but an increasing momentum parameter $\beta_t \to 1$.
2. **Comparison to GD.** With $\eta = \frac{1}{L}$
   - For GD, we had:
   $$f(\theta_t) - f(\theta_\star) \leqslant \frac{L\|\theta_0 - \theta_\star\|_2^2}{t}.$$
   - For NAG, we have:
   $$f(\theta_t) - f(\theta_\star) \leqslant \frac{2L\|\theta_0 - \theta_\star\|_2^2}{(t+1)^2}.$$

$\to$ The convergence is much faster!!

Rate of convergence of NAG - smooth strongly-cvx

Theorem : NAG rate of convergence - smooth & strongly-convex functions

If f is L-smooth and μ-strongly- convex with a minimum at $\theta_\star$. Then, if for all t, if

$$\beta_t = \beta = \frac{1-\sqrt{\mu/L}}{1+\sqrt{\mu/L}}, \qquad \eta_t = \eta = \frac{1}{L}$$

we have

$$f(\theta_t) - f(\theta_\star) \leqslant L\left(1-\sqrt{\frac{\mu}{L}}\right)^t \|\theta_0 - \theta_\star\|_2^2 = O\left(\exp\left(-\frac{t}{\sqrt{\kappa}}\right)\right).$$

Rate of convergence of NAG - smooth strongly-cvx

Theorem : NAG rate of convergence - smooth & strongly-convex functions

If f is L-smooth and μ-strongly- convex with a minimum at $\theta_\star$. Then, if for all t, if

$$\beta_t = \beta = \frac{1-\sqrt{\mu/L}}{1+\sqrt{\mu/L}}, \qquad \eta_t = \eta = \frac{1}{L}$$

we have

$$f(\theta_t) - f(\theta_\star) \leqslant L\left(1-\sqrt{\frac{\mu}{L}}\right)^t \|\theta_0 - \theta_\star\|_2^2 = O\left(\exp\left(-\frac{t}{\sqrt{\kappa}}\right)\right).$$

1. **Comparison to GD.** With $\eta = \frac{1}{L}$, we get resp.
   - For GD, we had:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{L}{2}\left(1-\frac{\mu}{L}\right)^t \|\theta_0 - \theta_\star\|_2^2 = O\left(\exp\left(-\frac{t}{\kappa}\right)\right).$$

Rate of convergence of NAG - smooth strongly-cvx

Theorem : NAG rate of convergence - smooth & strongly-convex functions

If f is L-smooth and μ-strongly- convex with a minimum at $\theta_\star$. Then, if for all t, if

$$\beta_t = \beta = \frac{1-\sqrt{\mu/L}}{1+\sqrt{\mu/L}}, \qquad \eta_t = \eta = \frac{1}{L}$$

we have

$$f(\theta_t) - f(\theta_\star) \leqslant L\left(1-\sqrt{\frac{\mu}{L}}\right)^t \|\theta_0 - \theta_\star\|_2^2 = O\left(\exp\left(-\frac{t}{\sqrt{\kappa}}\right)\right).$$

1. **Comparison to GD.** With $\eta = \frac{1}{L}$, we get resp.
   - For GD, we had:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{L}{2}\left(1-\frac{\mu}{L}\right)^t \|\theta_0 - \theta_\star\|_2^2 = O\left(\exp\left(-\frac{t}{\kappa}\right)\right).$$

$\rightarrow$ The convergence is much faster: we need $\sqrt{\kappa}$ times as less iterations to achieve the same upper bound!

2. Recall that in ML, κ can be of the order of 10^6 to 10^9 $\rightarrow$ acceleration can reduce the number of necessary iterations by a factor 1000+.

Rate of convergence of NAG - smooth strongly-cvx

Theorem : NAG rate of convergence - smooth & strongly-convex functions

If f is L-smooth and μ-strongly- convex with a minimum at $\theta_\star$. Then, if for all t, if

$$\beta_t = \beta = \frac{1-\sqrt{\mu/L}}{1+\sqrt{\mu/L}}, \qquad \eta_t = \eta = \frac{1}{L}$$

we have

$$f(\theta_t) - f(\theta_\star) \leqslant L\left(1-\sqrt{\frac{\mu}{L}}\right)^t \|\theta_0 - \theta_\star\|_2^2 = O\left(\exp\left(-\frac{t}{\sqrt{\kappa}}\right)\right).$$

1. **Comparison to GD.** With $\eta = \frac{1}{L}$, we get resp.
   - For GD, we had:

$$f(\theta_t) - f(\theta_\star) \leqslant \frac{L}{2}\left(1-\frac{\mu}{L}\right)^t \|\theta_0 - \theta_\star\|_2^2 = O\left(\exp\left(-\frac{t}{\kappa}\right)\right).$$

$\rightarrow$ The convergence is much faster: we need $\sqrt{\kappa}$ times as less iterations to achieve the same upper bound!

2. Recall that in ML, κ can be of the order of 10^6 to 10^9 $\rightarrow$ acceleration can reduce the number of necessary iterations by a factor 1000+.

Proposition (informal): Actually, NAG achieves the *optimal* rate for first order methods on smooth and (strongly)-convex functions.

Meaning that no other method using only first-order information can perform significantly faster (more than a constant times) on all functions of the class under consideration.

Theoretical results, Heavy ball method

For HB method, the results are more complicated.

Positive results:

1. It achieves acceleration (similar to NAG) if we consider only *quadratic* functions.
2. It is then even *twice* faster than NAG.

Theoretical results, Heavy ball method

For HB method, the results are more complicated.

Positive results:

1. It achieves acceleration (similar to NAG) if we consider only *quadratic* functions.
2. It is then even *twice* faster than NAG.

Negative results:

1. It is known to have pathological behaviors on non-quadratic functions.
2. Especially, it fails to systematically *accelerate* on this class.

Theoretical results, Heavy ball method

For HB method, the results are more complicated.

Positive results:

1. It achieves acceleration (similar to NAG) if we consider only *quadratic* functions.
2. It is then even *twice* faster than NAG.

Negative results:

1. It is known to have pathological behaviors on non-quadratic functions.
2. Especially, it fails to systematically *accelerate* on this class.

Nevertheless, HB is more used in practice, especially in deep-learning contexts, where it is used by default, and using NAG requires to change the default setting.

Tuning of β

The momentum parameter $\beta \in [0; 1]$ defines the strength of the acceleration.

1. $\beta = 0$: no acceleration, we get GD.
2. $\beta = 1$: divergence (no friction).

The good value is thus in between, typically close to 1.

- The two theorems before give a theoretically grounded tuning for particular class of functions.
- In practice, or if μ, L are unknown, difficult question. Often use first a default value of 0.9, or 0.99.

Summary: Accelerated methods - NAG and HB.

First-order method

Deterministic automatic adaptation for stochastic.

Accelerated: use a *momentum term*.

Intuition & motivation:
Challenge: GD suffers from bad conditioning, local minima
→ Solution: use *acceleration*
If we make consistent progress in one direction, accelerate in that direction.

Update:

$$\theta_t \overset{\text{NAG}}{\leftarrow} \theta_{t-1} - \eta \nabla f(\theta_{t-1}) + \beta(\theta_{t-1} - \theta_{t-2}).$$

$$\theta_t \overset{\text{HB}}{\leftarrow} \theta_{t-1} - \eta \nabla f(\zeta_{t-1}) + \beta(\theta_{t-1} - \theta_{t-2}).$$

Two strategies: only diff. is the point at which the gradient is taken

- HB: used more often in practice, optimal theoretically on quadratics
- NAG: optimal theoretically on smooth and (strongly) convex functions.

Hyperparameters: η, T, β,
$\rightsquigarrow \eta$: same tradeoffs as GD.
$\rightsquigarrow \beta$ theory or 0.9, 0.99

Complexity/iter for ERM

- nd if deterministic
- d if stochastic.

Theory:

- Rate for smooth and strongly convex functions for NAG
- NAG is "optimal" among first order method
- Impact of bad conditioning (κ large) reduced.

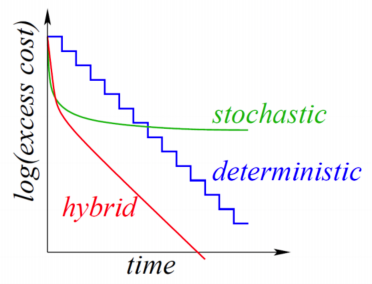
Take-aways:
→ Acceleration/momentum is fundamental to improve convergence, for poorly condition functions, on flat regions, saddle points, and to escape local minima.
→ It is systematically used in DL.

Let's recap! Wooclap quizz!

Outline

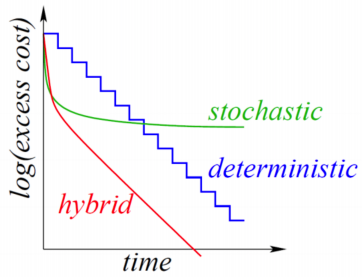
Motivation

1. Can we get the best of both worlds between SGD and GD?



Motivation

1. Can we get the best of both worlds between SGD and GD?



→ Objective of Variance reduced method.

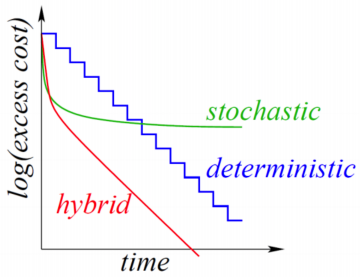
2. How to improve on SGD?

No memory / reusing of information from the past in SGD.

→ Key idea of Variance reduced method.

Motivation

1. Can we get the best of both worlds between SGD and GD?



→ Objective of Variance reduced method.

2. How to improve on SGD?

No memory / reusing of information from the past in SGD.

→ Key idea of Variance reduced method.

Variance reduction methods build on stochastic methods. They can thus ONLY be applied if the objective is a sum, such as

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

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

GD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5

✓ $\leftrightarrow \nabla f_i(\theta_t)$ = one gradient at the current iterate θ_t.

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

GD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t = 1$	✓	✓	✓	✓	✓

✓ $\leftrightarrow \nabla f_i(\theta_t)$ = one gradient at the current iterate θ_t.

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

GD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t = 1$	✓	✓	✓	✓	✓
$t = 2$	✓	✓	✓	✓	✓
$t = 3$	✓	✓	✓	✓	✓
$t = 4$	✓	✓	✓	✓	✓
$t = 5$	✓	✓	✓	✓	✓
$t = 6$	✓	✓	✓	✓	✓

✓ $\leftrightarrow \nabla f_i(\theta_t)$ = one gradient at the current iterate θ_t.

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

GD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t = 1$	✓	✓	✓	✓	✓
$t = 2$	✓	✓	✓	✓	✓
$t = 3$	✓	✓	✓	✓	✓
$t = 4$	✓	✓	✓	✓	✓
$t = 5$	✓	✓	✓	✓	✓
$t = 6$	✓	✓	✓	✓	✓

SGD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t = 1$: $i_1 = 3$			✓		
$t = 2$: $i_2 = 1$	✓				

✓ $\leftrightarrow \nabla f_i(\theta_t)$ = one gradient at the current iterate θ_t.

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

GD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t = 1$	✓	✓	✓	✓	✓
$t = 2$	✓	✓	✓	✓	✓
$t = 3$	✓	✓	✓	✓	✓
$t = 4$	✓	✓	✓	✓	✓
$t = 5$	✓	✓	✓	✓	✓
$t = 6$	✓	✓	✓	✓	✓

SGD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t = 1$: $i_1 = 3$			✓		
$t = 2$: $i_2 = 1$	✓				
$t = 3$: $i_3 = 4$				✓	
$t = 4$: $i_4 = 1$	✓				
$t = 5$: $i_5 = 5$					✓
$t = 6$: $i_6 = 2$		✓			

✓ $\leftrightarrow \nabla f_i(\theta_t)$ = one gradient at the current iterate θ_t.

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

GD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=1$	✓	✓	✓	✓	✓
$t=2$	✓	✓	✓	✓	✓
$t=3$	✓	✓	✓	✓	✓
$t=4$	✓	✓	✓	✓	✓
$t=5$	✓	✓	✓	✓	✓
$t=6$	✓	✓	✓	✓	✓

SGD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=1$: $i_1=3$			✓		
$t=2$: $i_2=1$	✓				
$t=3$: $i_3=4$				✓	
$t=4$: $i_4=1$	✓				
$t=5$: $i_5=5$					✓
$t=6$: $i_6=2$		✓			

SAG/SVRG

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=0$	✓	✓	✓	✓	✓
$t=1$: $i_1=3$	∼	∼	✓	∼	∼

✓ $\leftrightarrow \nabla f_i(\theta_t)$ = one gradient at the current iterate θ_t.

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

GD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=1$	✓	✓	✓	✓	✓
$t=2$	✓	✓	✓	✓	✓
$t=3$	✓	✓	✓	✓	✓
$t=4$	✓	✓	✓	✓	✓
$t=5$	✓	✓	✓	✓	✓
$t=6$	✓	✓	✓	✓	✓

SGD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=1$: $i_1=3$			✓		
$t=2$: $i_2=1$	✓				
$t=3$: $i_3=4$				✓	
$t=4$: $i_4=1$	✓				
$t=5$: $i_5=5$					✓
$t=6$: $i_6=2$		✓			

SAG/SVRG

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=0$	✓	✓	✓	✓	✓
$t=1$: $i_1=3$	∼	∼	✓	∼	∼
$t=2$: $i_2=1$	✓	∼	∼	∼	∼
$t=3$: $i_3=4$	∼	∼	∼	✓	∼
$t=4$: $i_4=1$	✓	∼	∼	∼	∼
$t=5$: $i_5=5$	∼	∼	∼	∼	✓
$t=6$: $i_6=2$	∼	✓	∼	∼	∼

✓ $\leftrightarrow \nabla f_i(\theta_t)$ = one gradient at the current iterate θ_t.

Variance reduced methods for finite sum minimization: SAG/SVRG

Objective: $\min \left\{ f(\theta) = \frac{1}{n} \sum_{i=0}^{n} f_i(\theta) \right\}$.

Update: $\theta_{t+1} = \theta_t - \eta_t \underbrace{g_t}_{??}$

GD: at step $t+1$

1. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_t)$

SGD: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$
2. use $g_t = \nabla f_{i_t}(\theta_t)$

SAG/SVRG: at step $t+1$,

1. sample $i_t \sim \mathcal{U}[1; n]$,
2. use $g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t_i})$
3. with $\theta_{t_i} \in \{\theta_1, \dots \theta_t\}$ "one iterate in the past",
4. and $\theta_{t_{i_t}}$ updated to θ_t.

Example: $n = 5$:

$$\frac{1}{5} \sum_{i=1}^{5} f_i(\theta)$$

GD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=1$	✓	✓	✓	✓	✓
$t=2$	✓	✓	✓	✓	✓
$t=3$	✓	✓	✓	✓	✓
$t=4$	✓	✓	✓	✓	✓
$t=5$	✓	✓	✓	✓	✓
$t=6$	✓	✓	✓	✓	✓

SGD:

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=1$: $i_1=3$			✓		
$t=2$: $i_2=1$	✓				
$t=3$: $i_3=4$				✓	
$t=4$: $i_4=1$	✓				
$t=5$: $i_5=5$					✓
$t=6$: $i_6=2$		✓			

SAG/SVRG

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5
$t=0$	✓	✓	✓	✓	✓
$t=1$: $i_1=3$	∼	∼	✓	∼	∼
$t=2$: $i_2=1$	✓	∼	∼	∼	∼
$t=3$: $i_3=4$	∼	∼	∼	✓	∼
$t=4$: $i_4=1$	✓	∼	∼	∼	∼
$t=5$: $i_5=5$	∼	∼	∼	∼	✓
$t=6$: $i_6=2$	∼	✓	∼	∼	∼

✓ $\leftrightarrow \nabla f_i(\theta_t)$ = one gradient at the current iterate θ_t.

∼ $\leftrightarrow \nabla f_i(\theta_{t_i})$ = one gradient an old iterate θ_{t_i} ($\rightsquigarrow$ difference between SVRG and SAG.)

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t = 0$	✓	✓	✓	✓	✓	θ_0	$\theta_1 \leftarrow \theta_0 - \eta/n \sum$

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t = 0$	✓	✓	✓	✓	✓	θ_0	
$t = 1$: $i_1 = 3$	$\sim$	$\sim$	✓	$\sim$	$\sim$	θ_1	$\theta_2 \leftarrow \theta_1 - \eta/n \sum$ □

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t = 0$	✓	✓	✓	✓	✓	θ_0	
$t = 1$: $i_1 = 3$	∼	∼	✓	∼	∼	θ_1	
$t = 2$: $i_2 = 1$	✓	∼	∼	∼	∼	θ_2	$\theta_3 \leftarrow \theta_2 - \eta/n \sum$ □

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t = 0$	✓	✓	✓	✓	✓	θ_0	
$t = 1$: $i_1 = 3$	$\sim$	$\sim$	✓	$\sim$	$\sim$	θ_1	
$t = 2$: $i_2 = 1$	✓	$\sim$	$\sim$	$\sim$	$\sim$	θ_2	
$t = 3$: $i_3 = 4$	$\sim$	$\sim$	$\sim$	✓	$\sim$	θ_3	
$t = 4$: $i_4 = 1$	✓	$\sim$	$\sim$	$\sim$	$\sim$	θ_4	$\theta_5 \leftarrow \theta_4 - \eta/n \sum$ □

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t = 0$	✓	✓	✓	✓	✓	θ_0	
$t = 1$: $i_1 = 3$	$\sim$	$\sim$	✓	$\sim$	$\sim$	θ_1	
$t = 2$: $i_2 = 1$	✓	$\sim$	$\sim$	$\sim$	$\sim$	θ_2	
$t = 3$: $i_3 = 4$	$\sim$	$\sim$	$\sim$	✓	$\sim$	θ_3	
$t = 4$: $i_4 = 1$	✓	$\sim$	$\sim$	$\sim$	$\sim$	θ_4	
$t = 5$: $i_5 = 5$	$\sim$	$\sim$	$\sim$	$\sim$	✓	θ_5	$\theta_6 \leftarrow \theta_5 - \eta/n \sum$ □

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t = 0$	✓	✓	✓	✓	✓	θ_0	
$t = 1$: $i_1 = 3$	$\sim$	$\sim$	✓	$\sim$	$\sim$	θ_1	
$t = 2$: $i_2 = 1$	✓	$\sim$	$\sim$	$\sim$	$\sim$	θ_2	
$t = 3$: $i_3 = 4$	$\sim$	$\sim$	$\sim$	✓	$\sim$	θ_3	
$t = 4$: $i_4 = 1$	✓	$\sim$	$\sim$	$\sim$	$\sim$	θ_4	
$t = 5$: $i_5 = 5$	$\sim$	$\sim$	$\sim$	$\sim$	✓	θ_5	$\theta_6 \leftarrow \theta_5 - \eta/n \sum$ □

In other words, at each step t:

1. For each function f_i, $i = 1, \dots, n$, keep in memory a gradient (represented as □) $g_{i,t} = \nabla f_i(\theta_{t_i})$ of f_i at the last point it was computed.

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t=0$	✓	✓	✓	✓	✓	θ_0	
$t=1$: $i_1=3$	∼	∼	✓	∼	∼	θ_1	
$t=2$: $i_2=1$	✓	∼	∼	∼	∼	θ_2	
$t=3$: $i_3=4$	∼	∼	∼	✓	∼	θ_3	
$t=4$: $i_4=1$	✓	∼	∼	∼	∼	θ_4	
$t=5$: $i_5=5$	∼	∼	∼	∼	✓	θ_5	$\theta_6 \leftarrow \theta_5 - \eta/n \sum$ □

In other words, at each step t:

1. For each function f_i, $i = 1, \dots, n$, keep in memory a gradient (represented as □) $g_{i,t} = \nabla f_i(\theta_{t_i})$ of f_i at the last point it was computed.
2. Sample $i_t \in \{1, \dots, n\}$ with replacement, and update: $g_{i_t,t} = \nabla f_{i_t}(\theta_t)$.

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t=0$	✓	✓	✓	✓	✓	θ_0	
$t=1$: $i_1=3$	∼	∼	✓	∼	∼	θ_1	
$t=2$: $i_2=1$	✓	∼	∼	∼	∼	θ_2	
$t=3$: $i_3=4$	∼	∼	∼	✓	∼	θ_3	
$t=4$: $i_4=1$	✓	∼	∼	∼	∼	θ_4	
$t=5$: $i_5=5$	∼	∼	∼	∼	✓	θ_5	$\theta_6 \leftarrow \theta_5 - \eta/n \sum$ □

In other words, at each step t:

1. For each function f_i, $i = 1, \dots, n$, keep in memory a gradient (represented as □) $g_{i,t} = \nabla f_i(\theta_{t_i})$ of f_i at the last point it was computed.
2. Sample $i_t \in \{1, \dots, n\}$ with replacement, and update: $g_{i_t,t} = \nabla f_{i_t}(\theta_t)$.
3. For $i \neq i_t$, propagate the previous value: $g_{i,t} = g_{i,t-1}$

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t=0$	✓	✓	✓	✓	✓	θ_0	
$t=1$: $i_1=3$	∼	∼	✓	∼	∼	θ_1	
$t=2$: $i_2=1$	✓	∼	∼	∼	∼	θ_2	
$t=3$: $i_3=4$	∼	∼	∼	✓	∼	θ_3	
$t=4$: $i_4=1$	✓	∼	∼	∼	∼	θ_4	
$t=5$: $i_5=5$	∼	∼	∼	∼	✓	θ_5	$\theta_6 \leftarrow \theta_5 - \eta/n \sum$ □

In other words, at each step t:

1. For each function f_i, $i = 1, \dots, n$, keep in memory a gradient (represented as □) $g_{i,t} = \nabla f_i(\theta_{t_i})$ of f_i at the last point it was computed.
2. Sample $i_t \in \{1, \dots, n\}$ with replacement, and update: $g_{i_t,t} = \nabla f_{i_t}(\theta_t)$.
3. For $i \neq i_t$, propagate the previous value: $g_{i,t} = g_{i,t-1}$
4. Iteration: $\theta_{t+1} = \theta_t - \dfrac{\eta}{n} \displaystyle\sum_{i=1}^{n} g_{i,t}$.

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t=0$	✓	✓	✓	✓	✓	θ_0	
$t=1$: $i_1=3$	∼	∼	✓	∼	∼	θ_1	
$t=2$: $i_2=1$	✓	∼	∼	∼	∼	θ_2	
$t=3$: $i_3=4$	∼	∼	∼	✓	∼	θ_3	
$t=4$: $i_4=1$	✓	∼	∼	∼	∼	θ_4	
$t=5$: $i_5=5$	∼	∼	∼	∼	✓	θ_5	$\theta_6 \leftarrow \theta_5 - \eta/n \sum$ □

In other words, at each step t:

1. For each function f_i, $i = 1, \dots, n$, keep in memory a gradient (represented as □) $g_{i,t} = \nabla f_i(\theta_{t_i})$ of f_i at the last point it was computed.
2. Sample $i_t \in \{1, \dots, n\}$ with replacement, and update: $g_{i_t,t} = \nabla f_{i_t}(\theta_t)$.
3. For $i \neq i_t$, propagate the previous value: $g_{i,t} = g_{i,t-1}$
4. Iteration: $\theta_{t+1} = \theta_t - \dfrac{\eta}{n} \displaystyle\sum_{i=1}^{n} g_{i,t}$.

Equivalently: $\sum_{i=1}^n g_{i,t} = \left(\sum_{i=1}^n \nabla f_i(\theta_{t_i}) - \nabla f_{i_t}(\theta_{t_{i_t}}) + \nabla f_{i_t}(\theta_t)\right)$.

SAG: precise definition

In SAG, we use as the *old iterate* the last time the corresponding function was picked.

Step \ Obs. used	f_1	f_2	f_3	f_4	f_5	"Current iterate"	Update
$t=0$	✓	✓	✓	✓	✓	θ_0	
$t=1$: $i_1=3$	∼	∼	✓	∼	∼	θ_1	
$t=2$: $i_2=1$	✓	∼	∼	∼	∼	θ_2	
$t=3$: $i_3=4$	∼	∼	∼	✓	∼	θ_3	
$t=4$: $i_4=1$	✓	∼	∼	∼	∼	θ_4	
$t=5$: $i_5=5$	∼	∼	∼	∼	✓	θ_5	$\theta_6 \leftarrow \theta_5 - \eta/n \sum$ □

In other words, at each step t:

1. For each function f_i, $i=1,\dots,n$, keep in memory a gradient (represented as □) $g_{i,t} = \nabla f_i(\theta_{t_i})$ of f_i at the last point it was computed.
2. Sample $i_t \in \{1,\dots,n\}$ with replacement, and update: $g_{i_t,t} = \nabla f_{i_t}(\theta_t)$.
3. For $i \neq i_t$, propagate the previous value: $g_{i,t} = g_{i,t-1}$
4. Iteration: $\theta_{t+1} = \theta_t - \dfrac{\eta}{n} \displaystyle\sum_{i=1}^{n} g_{i,t}$.

Equivalently: $\sum_{i=1}^n g_{i,t} = \left(\sum_{i=1}^n \nabla f_i(\theta_{t_i}) - \nabla f_{i_t}(\theta_{t_{i_t}}) + \nabla f_{i_t}(\theta_t)\right)$.

↬ $\oplus$ update costs the same as SGD → complexity per iteration $O(d)$.
↬ $\ominus$ needs to store all gradients $\nabla f_i(\theta_{t_i})$ at "points in the past" → storage cost $O(nd)$.

Variance reduced algorithms: SAG

Algorithm : Stochastic Average Gradient (SAG)

Hyperparameters: learning rate $\eta > 0$, number of steps T.

Initialization: initial weight vector θ_0.

Iteration 0. For all $i = 1, \dots, n$, compute $g_{i,0} \leftarrow \nabla f_i(\theta_{(0)})$. Update $\theta_1 = \theta_0 - \eta \sum_{i=1}^n g_{i,0}$.

For $t = 1, 2, \dots, T$ do

- Pick uniformly at random i_t in $\{1, \dots, n\}$
- Update $g_{i_t,t} \leftarrow \nabla f_{i_t}(\theta_{t-1})$
- Propagate for all $i \neq i_t$, $g_{i,t} \leftarrow g_{i,t-1}$.
- Apply

$$\theta_t \leftarrow \theta_{t-1} - \frac{\eta}{n} \sum_{i=1}^n g_{i,t}$$

Output: Return last θ_T

Variant of SAG: SAGA $\rightarrow$ similar but with more weight on the current gradient / current point.

Solving the memory problem: SVRG

SAG method is promising, but its implementation is limited to "moderate n" regime:

- n large enough for stochastic methods to be relevant.
- the memory cost $n \times d$ needs not to be too large.

To solve the memory issue, SVRG was introduced. SVRG enjoys:

- $\oplus$ A negligible memory cost: $O(d)$
- $\ominus$ A computational cost twice as large as SGD ($2d$ per iteration).
- $\ominus$ Has an extra hyperparameter m, not always easy to tune.

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1, \dots, T \\ k \text{ inner loop index, in } 1, \dots, m \end{cases}$

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1, \dots, T \\ k \text{ inner loop index, in } 1, \dots, m \end{cases}$

$\rightarrow$ Outer loop: reset every m iter. We compute and store $\nabla f(\theta_{t+1,0}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t+1,0})$

Outer loop – Inner loop / Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$	Update	Iter. complexity
$t = 1$	$k = 0$	✓	✓	✓	✓	✓	$\theta_{1,0}$	$\theta_{1,1} \leftarrow \theta_{1,0} - \frac{\eta}{n} \sum \square$.	$O(nd)$

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1, \dots, T \\ k \text{ inner loop index, in } 1, \dots, m \end{cases}$

$\rightarrow$ Outer loop: reset every m iter. We compute and store $\nabla f(\theta_{t+1,0}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t+1,0})$

$\rightarrow$ Inner loop update: $\theta_{t,k+1} \leftarrow \theta_{t,k} - \frac{\eta}{n} \sum \square$.

Outer loop – Inner loop \ Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$	Update	Iter. complexity
$t=1$	$k=0$	✓	✓	✓	✓	✓	$\theta_{1,0}$		$O(nd)$
	$k=1$: $i_1=3$	∼	∼	✓	∼	∼	$\theta_{1,1}$	$\theta_{1,2} \leftarrow \theta_{1,1} - \frac{\eta}{n} \sum \square$.	$O(d)$

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1, \dots, T \\ k \text{ inner loop index, in } 1, \dots, m \end{cases}$

$\rightarrow$ Outer loop: reset every m iter. We compute and store $\nabla f(\theta_{t+1,0}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t+1,0})$

$\rightarrow$ Inner loop update: $\theta_{t,k+1} \leftarrow \theta_{t,k} - \frac{\eta}{n} \sum \square$.

Outer loop – Inner loop \ Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$	Update	Iter. complexity
$t=1$	$k=0$	✓	✓	✓	✓	✓	$\theta_{1,0}$		$O(nd)$
	$k=1$: $i_1=3$	~	~	✓	~	~	$\theta_{1,1}$		$O(d)$
	$k=2$: $i_2=1$	✓	~	~	~	~	$\theta_{1,2}$	$\theta_{1,3} \leftarrow \theta_{1,2} - \frac{\eta}{n} \sum \square$.	$O(d)$

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1,\dots,T \\ k \text{ inner loop index, in } 1,\dots,m \end{cases}$

$\rightarrow$ Outer loop: reset every m iter. We compute and store $\nabla f(\theta_{t+1,0}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\theta_{t+1,0})$

$\rightarrow$ Inner loop update: $\theta_{t,k+1} \leftarrow \theta_{t,k} - \frac{\eta}{n}\sum \square$.

Outer loop – Inner loop \ Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$	Update	Iter. complexity
$t=1$	$k=0$	✓	✓	✓	✓	✓	$\theta_{1,0}$		$O(nd)$
	$k=1$: $i_1=3$	~	~	✓	~	~	$\theta_{1,1}$		$O(d)$
	$k=2$: $i_2=1$	✓	~	~	~	~	$\theta_{1,2}$		$O(d)$
	$\vdots$						$\vdots$		
	$k=m$: $i_m=4$	~	~	~	✓	~	$\theta_{1,m}$	$\theta_{1,m+1} \leftarrow \theta_{1,m} - \frac{\eta}{n}\sum \square$.	$O(d)$

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1, \dots, T \\ k \text{ inner loop index, in } 1, \dots, m \end{cases}$

$\rightarrow$ Outer loop: reset every m iter. We compute and store $\nabla f(\theta_{t+1,0}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta_{t+1,0})$

$\rightarrow$ Inner loop update: $\theta_{t,k+1} \leftarrow \theta_{t,k} - \frac{\eta}{n} \sum \square$.

Outer loop – Inner loop / Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$	Update	Iter. complexity
$t=1$	$k=0$	✓	✓	✓	✓	✓	$\theta_{1,0}$		$O(nd)$
	$k=1$: $i_1=3$	∼	∼	✓	∼	∼	$\theta_{1,1}$		$O(d)$
	$k=2$: $i_2=1$	✓	∼	∼	∼	∼	$\theta_{1,2}$		$O(d)$
	⋮						⋮		
	$k=m$: $i_m=4$	∼	∼	∼	✓	∼	$\theta_{1,m}$		$O(d)$
$t=2$	$k=0$:	✓	✓	✓	✓	✓	$\theta_{2,0} \leftarrow \theta_{1,m}$	$\theta_{2,1} \leftarrow \theta_{2,0} - \frac{\eta}{n} \sum \square$	$O(nd)$

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1,\dots,T \\ k \text{ inner loop index, in } 1,\dots,m \end{cases}$

$\rightarrow$ Outer loop: reset every m iter. We compute and store $\nabla f(\theta_{t+1,0}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\theta_{t+1,0})$

$\rightarrow$ Inner loop update: $\theta_{t,k+1} \leftarrow \theta_{t,k} - \frac{\eta}{n}\sum \square$.

Outer loop – Inner loop / Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$	Update	Iter. complexity
$t=1$	$k=0$	✓	✓	✓	✓	✓	$\theta_{1,0}$		$O(nd)$
	$k=1$: $i_1=3$	∼	∼	✓	∼	∼	$\theta_{1,1}$		$O(d)$
	$k=2$: $i_2=1$	✓	∼	∼	∼	∼	$\theta_{1,2}$		$O(d)$
	⋮						⋮		
	$k=m$: $i_m=4$	∼	∼	∼	✓	∼	$\theta_{1,m}$		$O(d)$
$t=2$	$k=0$:	✓	✓	✓	✓	✓	$\theta_{2,0} \leftarrow \theta_{1,m}$		$O(nd)$
	$k=1$: $i_1=5$	∼	∼	∼	∼	✓	$\theta_{2,1}$	$\theta_{2,2} \leftarrow \theta_{2,1} - \frac{\eta}{n}\sum \square$.	$O(d)$

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1,\dots,T \\ k \text{ inner loop index, in } 1,\dots,m \end{cases}$

$\rightarrow$ Outer loop: reset every m iter. We compute and store $\nabla f(\theta_{t+1,0}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\theta_{t+1,0})$

$\rightarrow$ Inner loop update: $\theta_{t,k+1} \leftarrow \theta_{t,k} - \frac{\eta}{n}\sum \square$.

Outer loop – Inner loop \ Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$	Update	Iter. complexity
$t=1$	$k=0$	✓	✓	✓	✓	✓	$\theta_{1,0}$		$O(nd)$
	$k=1$: $i_1=3$	∼	∼	✓	∼	∼	$\theta_{1,1}$		$O(d)$
	$k=2$: $i_2=1$	✓	∼	∼	∼	∼	$\theta_{1,2}$		$O(d)$
	⋮						⋮		
	$k=m$: $i_m=4$	∼	∼	∼	✓	∼	$\theta_{1,m}$		$O(d)$
$t=2$	$k=0$:	✓	✓	✓	✓	✓	$\theta_{2,0} \leftarrow \theta_{1,m}$		$O(nd)$
	$k=1$: $i_1=5$	∼	∼	∼	∼	✓	$\theta_{2,1}$		$O(d)$
	$k=2$: $i_2=2$	∼	✓	∼	∼	∼	$\theta_{2,2}$	$\theta_{2,3} \leftarrow \theta_{2,2} - \frac{\eta}{n}\sum \square$	$O(d)$
	⋮					⋮			

Solving the memory problem: SVRG

How SVRG works: two loops structure: Iterate $\theta_{t,k}$: $\begin{cases} t \text{ outer loop index, in } 1,\dots,T \\ k \text{ inner loop index, in } 1,\dots,m \end{cases}$

$\rightarrow$ Outer loop: reset every m iter. We compute and store $\nabla f(\theta_{t+1,0}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\theta_{t+1,0})$

$\rightarrow$ Inner loop update: $\theta_{t,k+1} \leftarrow \theta_{t,k} - \frac{\eta}{n}\sum \square$.

Outer loop – Inner loop \ Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$	Update	Iter. complexity
$t=1$	$k=0$	✓	✓	✓	✓	✓	$\theta_{1,0}$		$O(nd)$
	$k=1$: $i_1=3$	~	~	✓	~	~	$\theta_{1,1}$		$O(d)$
	$k=2$: $i_2=1$	✓	~	~	~	~	$\theta_{1,2}$		$O(d)$
	$\vdots$						$\vdots$		
	$k=m$: $i_m=4$	~	~	~	✓	~	$\theta_{1,m}$		$O(d)$
$t=2$	$k=0$:	✓	✓	✓	✓	✓	$\theta_{2,0} \leftarrow \theta_{1,m}$		$O(nd)$
	$k=1$: $i_1=5$	~	~	~	~	✓	$\theta_{2,1}$		$O(d)$
	$k=2$: $i_2=2$	~	✓	~	~	~	$\theta_{2,2}$	$\theta_{2,3} \leftarrow \theta_{2,2} - \frac{\eta}{n}\sum \square$	$O(d)$
	$\vdots$					$\vdots$			

The "old iterate" is always the one at the beginning of the loop $\theta_{t,0}$. That way, we only have to store *one* gradient $\nabla f(\theta_{t,0})$. Indeed:

$$\sum \square = \left(\underbrace{\nabla f(\theta_{t,0})}_{\text{stored}} + \underbrace{\nabla f_{i_k}(\theta_{t,k})}_{\text{computed} \rightarrow O(d)} - \underbrace{\nabla f_{i_k}(\theta_{t,0})}_{\text{computed} \rightarrow O(d)} \right)$$

Solving the memory problem: SVRG

Algorithm : Stochastic Variance Reduced Gradient (SVRG)

Hyperparameters: step $\eta > 0$, # of outer loops T, # of inner loop iterations m.
Initialization: initial weight vector θ_0
Outer loop: for $t = 1, 2, \ldots, T$ do

- Compute $\nabla f(\theta_{t,0})$
- **Inner loop:** for $k = 1, \ldots, m$
  - Sample uniformly at random i_k in $\{1, \ldots, n\}$
  - Apply the step

$$\theta_{t,k+1} \leftarrow \theta_{t,k} - \eta(\nabla f_{i_k}(\theta_{t,k}) - \nabla f_{i_k}(\theta_{t,0}) + \nabla f(\theta_{t,0}))$$

- Set

$$\theta_{t+1,0} \leftarrow \theta_{t,m+1} \qquad \left(\text{or } \frac{1}{m} \sum_{k=1}^{m} \theta_{t,k} \right)$$

Output: Return $\theta_{T+1,0}$.

Outer loop – Inner loop \ Obs. used		f_1	f_2	f_3	f_4	f_5	"Current iterate" $\theta_{t,k}$
$t = 1$	$k = 0$	✓	✓	✓	✓	✓	$\theta_{1,0}$
	$k = 1$: $i_1 = 3$	∼	∼	✓	∼	∼	$\theta_{1,1}$
	$k = 2$: $i_2 = 1$	✓	∼	∼	∼	∼	$\theta_{1,2}$
	⋮						⋮
	$k = m$: $i_m = 4$	∼	∼	∼	✓	∼	$\theta_{1,m}$
$t = 1$	$k = 0$:	✓	✓	✓	✓	✓	$\theta_{2,0} = \theta_{1,m}$
	$k = 1$: $i_1 = 5$	∼	∼	∼	∼	✓	$\theta_{2,1}$
	⋮					⋮	

Remark: the choice of m for SVRG is not always easy. By default, use $m = n$.

Theoretical results for variance reduced methods

Theorem : Convergence of SAGA

Assume that *all* functions f_i are L-smooth and f is μ-strongly convex. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then with $\eta = \frac{1}{4L}$, after T iterations, SAGA satisfies:

$$\mathbb{E}\left[f(\theta_T) - f(\theta_\star)\right] \leqslant \frac{L}{2}\left(1 - \min\left\{\frac{1}{3n}, \frac{3\mu}{16L}\right\}\right)^T \left(1 + \frac{n}{4}\right)\|\theta_0 - \theta_\star\|_2^2.$$

Theoretical results for variance reduced methods

Theorem : Convergence of SAGA

Assume that *all* functions f_i are L-smooth and f is μ-strongly convex. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then with $\eta = \frac{1}{4L}$, after T iterations, SAGA satisfies:

$$\mathbb{E}\left[f(\theta_T) - f(\theta_\star)\right] \leqslant \frac{L}{2}\left(1 - \min\left\{\frac{1}{3n}, \frac{3\mu}{16L}\right\}\right)^T \left(1 + \frac{n}{4}\right)\|\theta_0 - \theta_\star\|_2^2.$$

Similar contraction to GD (recall Theorem on GD under same assumptions, $\eta = \frac{1}{L}$):

$$f(\theta_T) - f(\theta_\star) \leqslant \frac{L}{2}\left(1 - \frac{\mu}{L}\right)^T \|\theta_0 - \theta_\star\|_2^2. \qquad \text{(Reminder)}$$

Theoretical results for variance reduced methods

Theorem : Convergence of SAGA

Assume that *all* functions f_i are L-smooth and f is μ-strongly convex. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then with $\eta = \frac{1}{4L}$, after T iterations, SAGA satisfies:

$$\mathbb{E}\left[f(\theta_T) - f(\theta_\star)\right] \leqslant \frac{L}{2}\left(1 - \min\left\{\frac{1}{3n}, \frac{3\mu}{16L}\right\}\right)^T \left(1 + \frac{n}{4}\right)\|\theta_0 - \theta_\star\|_2^2.$$

Similar contraction to GD (recall Theorem on GD under same assumptions, $\eta = \frac{1}{L}$):

$$f(\theta_T) - f(\theta_\star) \leqslant \frac{L}{2}\left(1 - \frac{\mu}{L}\right)^T \|\theta_0 - \theta_\star\|_2^2. \qquad \text{(Reminder)}$$

- Choice of hyperparameter η:
  - $\rightarrow$ similar to GD: we use η constant.
  - $\rightarrow$ here $\eta = \frac{1}{4L}$ is suggested by theory.
- no noise/initial condition tradeoff

$\rightarrow$ the dynamic thus compares to the one of GD.

Theoretical results for variance reduced methods

Theorem : Convergence of SAGA

Assume that *all* functions f_i are L-smooth and f is μ-strongly convex. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then with $\eta = \frac{1}{4L}$, after T iterations, SAGA satisfies:

$$\mathbb{E}\left[f(\theta_T) - f(\theta_\star)\right] \leqslant \frac{L}{2}\left(1 - \min\left\{\frac{1}{3n}, \frac{3\mu}{16L}\right\}\right)^T \left(1 + \frac{n}{4}\right)\|\theta_0 - \theta_\star\|_2^2.$$

Similar contraction to GD (recall Theorem on GD under same assumptions, $\eta = \frac{1}{L}$):

$$f(\theta_T) - f(\theta_\star) \leqslant \frac{L}{2}\left(1 - \frac{\mu}{L}\right)^T \|\theta_0 - \theta_\star\|_2^2. \qquad \text{(Reminder)}$$

- Choice of hyperparameter η:
  - $\rightarrow$ similar to GD: we use η constant.
  - $\rightarrow$ here $\eta = \frac{1}{4L}$ is suggested by theory.
- no noise/initial condition tradeoff

$\rightarrow$ the dynamic thus compares to the one of GD.

(Advanced): proof is of interest, see e.g. Francis Bach's book, Th. 5.6.

- *Idea of the proof:* the gradient oracle has a variance that goes to 0 as $t \to \infty$.
- In words, although the method is stochastic, the amount of noise reduces to 0 as we advance along the iterations.

Theoretical results for variance reduced methods

Theorem : Convergence of SAGA

Assume that *all* functions f_i are L-smooth and f is μ-strongly convex. Let $\theta_\star$ be the minimum of f on $\mathbb{R}^d$. Then with $\eta = \frac{1}{4L}$, after T iterations, SAGA satisfies:

$$\mathbb{E}\left[f(\theta_T) - f(\theta_\star)\right] \leqslant \frac{L}{2}\left(1 - \min\left\{\frac{1}{3n}, \frac{3\mu}{16L}\right\}\right)^T \left(1 + \frac{n}{4}\right)\|\theta_0 - \theta_\star\|_2^2.$$

Similar contraction to GD (recall Theorem on GD under same assumptions, $\eta = \frac{1}{L}$):

$$f(\theta_T) - f(\theta_\star) \leqslant \frac{L}{2}\left(1 - \frac{\mu}{L}\right)^T \|\theta_0 - \theta_\star\|_2^2. \qquad \text{(Reminder)}$$

- Choice of hyperparameter η:
  - $\rightarrow$ similar to GD: we use η constant.
  - $\rightarrow$ here $\eta = \frac{1}{4L}$ is suggested by theory.
- no noise/initial condition tradeoff

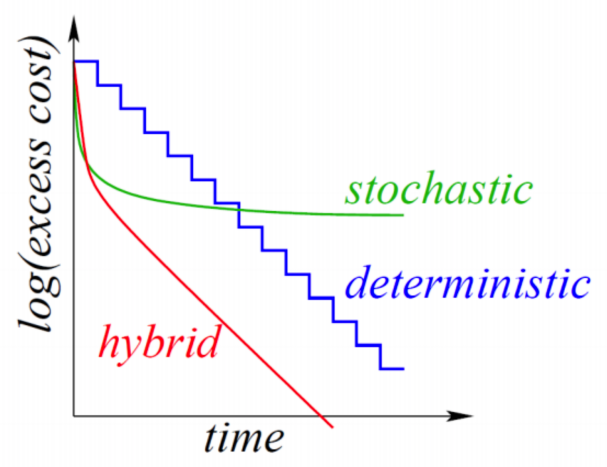
$\rightarrow$ the dynamic thus compares to the one of GD.

(Advanced): proof is of interest, see e.g. Francis Bach's book, Th. 5.6.

- *Idea of the proof:* the gradient oracle has a variance that goes to 0 as $t \to \infty$.
- In words, although the method is stochastic, the amount of noise reduces to 0 as we advance along the iterations.

Variance reduction enables to achieve the best of both worlds.

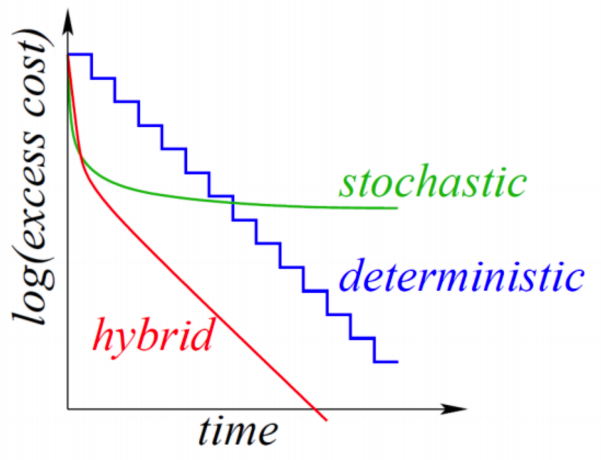
	SGD	GD	SAG/SVRG/SAGA
Complexity per iteration	$O(d)$	$O(n \times d)$	$O(d)$
Rate for L-smooth & μ-strongly-convex	$O\left(\frac{\kappa}{t}\right)$	$O\left(\exp\left(-\frac{t}{\kappa}\right)\right)$	$O\left(\exp\left(-t(\frac{1}{\kappa} \wedge \frac{1}{n})\right)\right)$



(Fig. from Schmidt, Le Roux, Bach 2023)

Variance reduction enables to achieve the best of both worlds.

	SGD	GD	SAG/SVRG/SAGA
Complexity per iteration	$O(d)$	$O(n \times d)$	$O(d)$
Rate for L-smooth & μ-strongly-convex	$O\left(\frac{\kappa}{t}\right)$	$O\left(\exp\left(-\frac{t}{\kappa}\right)\right)$	$O\left(\exp\left(-t(\frac{1}{\kappa} \wedge \frac{1}{n})\right)\right)$



(Fig. from Schmidt, Le Roux, Bach 2023)

Extensions

1. **Other schemes:** The study of variance reduced methods is still an active field of research. Many novel methods are regularly presented.
   $\rightsquigarrow$ SDCA, Spider, etc.
2. **Mini-batches:** it is straightforwardly possible to extend SAG, SVRG, etc. to b-minibatches at each step.

Summary: variance reduction (VR) methods.

First-order method **Stochastic**

⚠ Only applicable to finite sums problems

Update

$$\theta_t \leftarrow \theta_{t-1} - \eta g_t(\theta_{t-1})$$

with g_t gradient oracle based on one *fresh* gradient and gradients *from the past*.

Hyperparameters: η, T, ($+$ m for SVRG.)
$\rightsquigarrow$ η: same tradeoffs as GD.
$\rightsquigarrow$ m not easy to choose

Complexity/iter in ML:

d for SAG, SAGA

$2d$ for SVRG

Intuition & motivation:
Challenge: SGD suffers from the noise
$\rightarrow$ Solution: use *previously observed gradients* to reduce the noise

Theory:

- Rate for smooth and strongly convex functions
- 💡 VR achieves the best of both worlds between SGD and GD.
- 💡 VR achieves high precision.

Several strategies:

- SAG: high mem. cost $O(nd)$, single loop
- SVRG: normal mem. cost $O(d)$, two loops.
- SAGA, Spider, etc.

$\rightarrow$ VR is the one of the best candidates for n large, convex and smooth problems, where we want to achieve a high precision.

Variance reduction: take-aways and limits

Variance reduced methods are often the best ones in large n, convex, high precision, scenarios.

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

Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver, you might want to consider the following aspects:

- 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.

Figure: sklearn documentation

See also:

1. Benchmarking of multiple strategies - *benchopt*.
2. Implementation of SAG/SAGA/SVRG in *lightning*.

Stochastic averaged gradient (SAG and SAGA)

`classification.SAGClassifier`, `classification.SAGAClassifier`, `regression.SAGRegressor`, `regression.SAGARegressor`

- Main idea: instead of using the full gradient (average of sample-wise gradients), compute gradient for a randomly selected sample and use out-dated gradients for other samples
- Non-smooth losses: Yes (`classification.SAGAClassifier` and `regression.SAGARegressor`)
- Penalties: L1, L2, Elastic-net
- Learning rate: Yes (not very sensitive)
- Multiclass: one-vs-rest

Stochastic variance-reduced gradient (SVRG)

`classification.SVRGClassifier`, `regression.SVRGRegressor`

- Main idea: compute full gradient periodically and use it to center the gradient estimate (this can be shown to reduce the variance)
- Non-smooth losses: No
- Penalties: L2
- Learning rate: Yes (not very sensitive)
- Multiclass: one-vs-rest

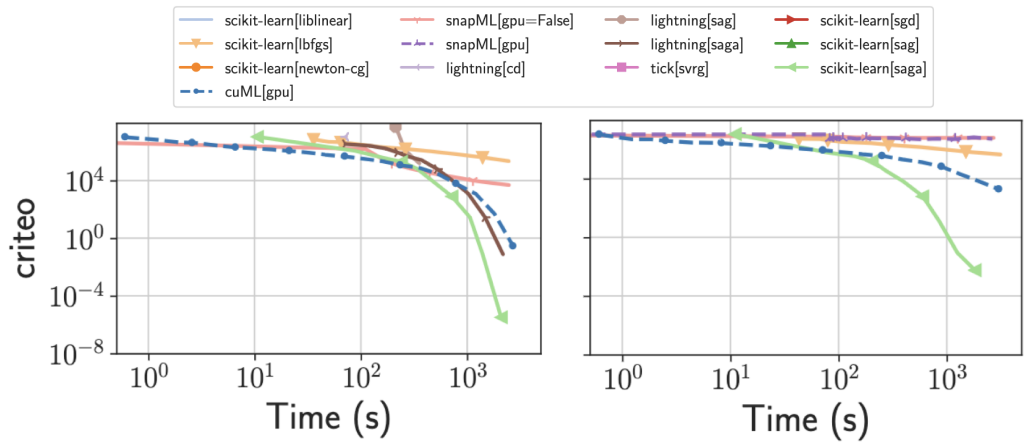


Figure: lighting documentation - *Benchopt* results - link

Variance reduction: take-aways and limits

Variance reduced methods are often the best ones in large n, convex, high precision, scenarios.

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

Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver, you might want to consider the following aspects:

- 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.

Figure: sklearn documentation

See also:

1. Benchmarking of multiple strategies - *benchopt*.
2. Implementation of SAG/SAGA/SVRG in *lightning*.

However:

1. High precision is not always useful
2. Typically not used in deep learning.
   - Those methods require some regularity of the functions f_i (for the "past gradients" to be good estimators of the gradients at the current point).
   - Empirically, not successful $\rightsquigarrow$ hypothesis: convergence to "bad minima" $\rightarrow$ bad generalization (advanced).

$\rightarrow$ variance reduced methods are not used in deep learning contexts.

$\rightarrow$ not even implemented in keras, pytorch, tensorflow...

Let's recap! Wooclap quizz!

Outline

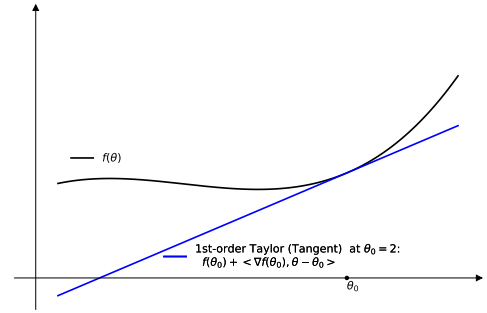
1. Gradient descent method, stochastic gradient method

2. Extensions of GD and SGD
   - Momentum and Acceleration
   - Variance reduced methods for ERM
   - Second-order algorithms
     - Motivation and algorithms
     - Theoretical results
     - Quasi-Newton methods - Summary

Intuition - Second order methods - Newton method

Reminder - Proposition

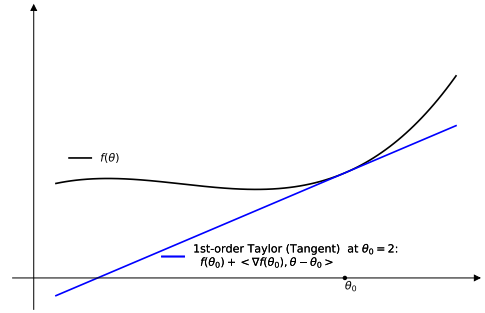
$$\underset{\theta \in B(\theta_0, R)}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle}_{\text{First-order approximation}} = \underbrace{\theta_0 - \frac{R}{\|\nabla f(\theta_0)\|} \nabla f(\theta_0)}_{\text{GD!}}$$



Intuition - Second order methods - Newton method

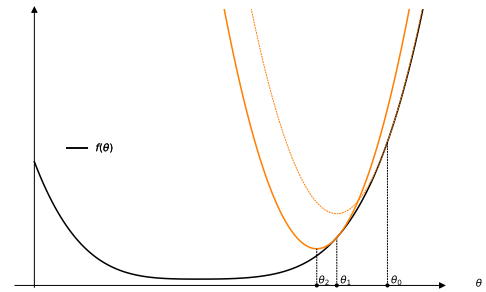
Reminder - Proposition

$$\underset{\theta \in B(\theta_0, R)}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle}_{\text{First-order approximation}} = \underbrace{\theta_0 - \frac{R}{\|\nabla f(\theta_0)\|} \nabla f(\theta_0)}_{\text{GD!}}$$



Reminder - Proposition

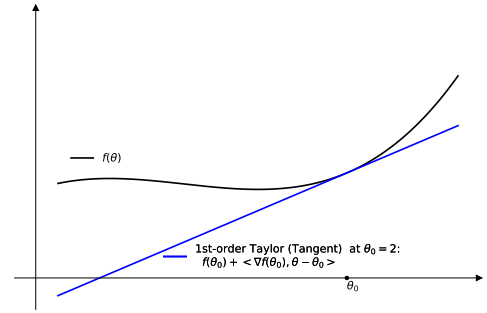
$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{L}{2} \|\theta - \theta_0\|^2}_{\text{Quadratic upper bound}} = \underbrace{\theta_0 - \frac{1}{L} \nabla f(\theta_0)}_{\text{GD!}}$$



Intuition - Second order methods - Newton method

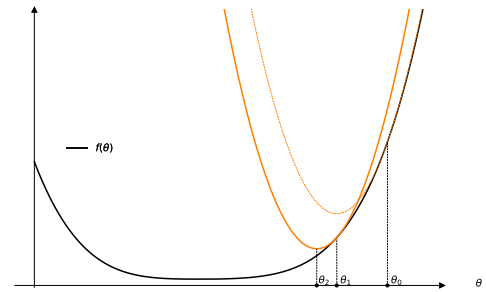
Reminder - Proposition

$$\underset{\theta \in B(\theta_0, R)}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle}_{\text{First-order approximation}} = \underbrace{\theta_0 - \frac{R}{\|\nabla f(\theta_0)\|} \nabla f(\theta_0)}_{\text{GD!}}$$



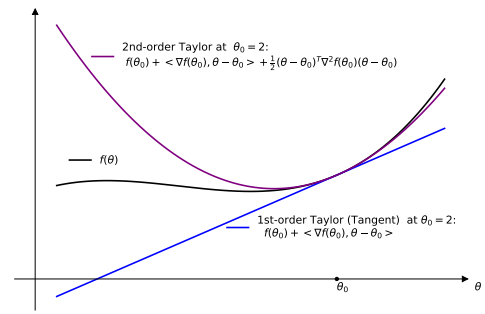
Reminder - Proposition

$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{L}{2} \|\theta - \theta_0\|^2}_{\text{Quadratic upper bound}} = \underbrace{\theta_0 - \frac{1}{L} \nabla f(\theta_0)}_{\text{GD!}}$$



💡 **Newton**: global minimization Second-order Taylor approximation

$$\underbrace{\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2} (\theta - \theta_0)^\top \nabla^2 f(\theta_0) (\theta - \theta_0)}_{\text{Second-order approximation}}}_{\textbf{Newton update}}$$



→ **Second order method:** requires to know the Hessian matrix $\nabla^2 f(\theta_0)$

Newton algorithm - closed form solution

- The Newton's direction minimizes the best locally quadratic approximation of f.
- For a point θ_0, such that $\nabla^2 f(\theta_0) \succ 0$

$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation} =: h_{\theta_0}(\theta)} = \underbrace{\theta_0 - [\nabla^2 f(\theta_0)]^{-1} \nabla f(\theta_0)}_{\textbf{Newton update}}$$

Newton algorithm - closed form solution

- The Newton's direction minimizes the best locally quadratic approximation of f.
- For a point θ_0, such that $\nabla^2 f(\theta_0) \succ 0$

$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation} =: h_{\theta_0}(\theta)} = \underbrace{\theta_0 - [\nabla^2 f(\theta_0)]^{-1} \nabla f(\theta_0)}_{\textbf{Newton update}}$$

Proof:

$$\nabla h_{\theta_0}(\theta) = \nabla f(\theta_0) + \nabla^2 f(\theta_0)\ (\theta - \theta_0)$$

$$\nabla h_{\theta_0}(\theta) = 0 \overset{\nabla^2 f(\theta_0) \text{invertible}}{\Leftrightarrow} \theta - \theta_0 = -(\nabla^2 f(\theta_0))^{-1} \nabla f(\theta_0)$$

$$\Leftrightarrow \theta = \theta_0 - (\nabla^2 f(\theta_0))^{-1} \nabla f(\theta_0)$$

Newton algorithm - closed form solution

- The Newton's direction minimizes the best locally quadratic approximation of f.
- For a point θ_0, such that $\nabla^2 f(\theta_0) \succ 0$

$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation} =: h_{\theta_0}(\theta)} = \underbrace{\theta_0 - [\nabla^2 f(\theta_0)]^{-1} \nabla f(\theta_0)}_{\textbf{Newton update}}$$

Proof:

$$\nabla h_{\theta_0}(\theta) = \nabla f(\theta_0) + \nabla^2 f(\theta_0)\ (\theta - \theta_0)$$

$$\nabla h_{\theta_0}(\theta) = 0 \overset{\nabla^2 f(\theta_0) \text{invertible}}{\Leftrightarrow} \theta - \theta_0 = -(\nabla^2 f(\theta_0))^{-1} \nabla f(\theta_0)$$

$$\Leftrightarrow \theta = \theta_0 - (\nabla^2 f(\theta_0))^{-1} \nabla f(\theta_0)$$

Critical point and $\forall \theta$, $\nabla^2 h_{\theta_0}(\theta) = \nabla^2 f(\theta_0) \succ 0 \Rightarrow$ global minimum

Newton algorithm - closed form solution

- The Newton's direction minimizes the best locally quadratic approximation of f.
- For a point θ_0, such that $\nabla^2 f(\theta_0) \succ 0$

$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation} =: h_{\theta_0}(\theta)} = \underbrace{\theta_0 - [\nabla^2 f(\theta_0)]^{-1} \nabla f(\theta_0)}_{\textbf{Newton update}}$$

Proof:

$$\nabla h_{\theta_0}(\theta) = \nabla f(\theta_0) + \nabla^2 f(\theta_0)\ (\theta - \theta_0)$$

$$\nabla h_{\theta_0}(\theta) = 0 \overset{\nabla^2 f(\theta_0) \text{invertible}}{\Leftrightarrow} \theta - \theta_0 = -(\nabla^2 f(\theta_0))^{-1} \nabla f(\theta_0)$$

$$\Leftrightarrow \theta = \theta_0 - (\nabla^2 f(\theta_0))^{-1} \nabla f(\theta_0)$$

Critical point and $\forall \theta$, $\nabla^2 h_{\theta_0}(\theta) = \nabla^2 f(\theta_0) \succ 0 \Rightarrow$ global minimum

Algorithm : Newton Descent.

Input: Twice differentiable function f to minimize.
Initialization: initial weight vector θ_0
Hyper-parameters: number of iterations T.
For $t \in \{1, \dots, T\}$:

- $\theta_{t+1} \leftarrow \theta_t - (\nabla^2 f(\theta_t))^{-1} \nabla f(\theta_t)$

Output: θ_T.

Visualization

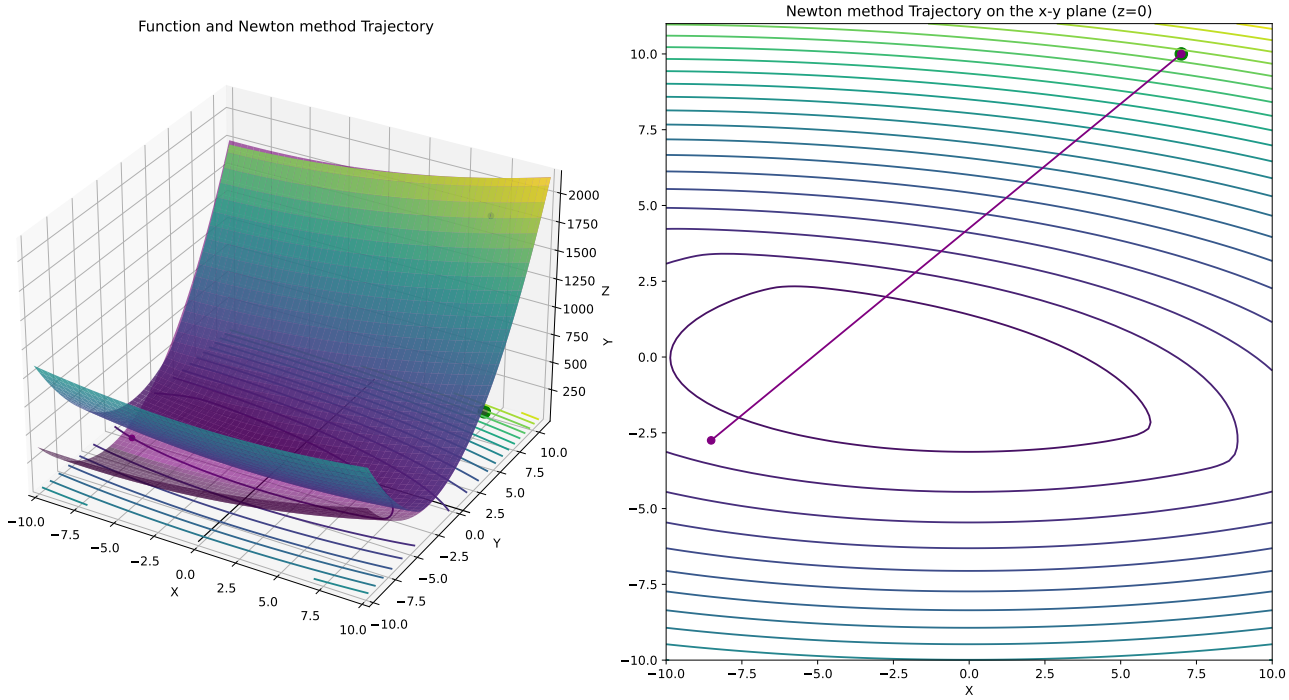


Figure: Newton method, step $T = 1$

Visualization

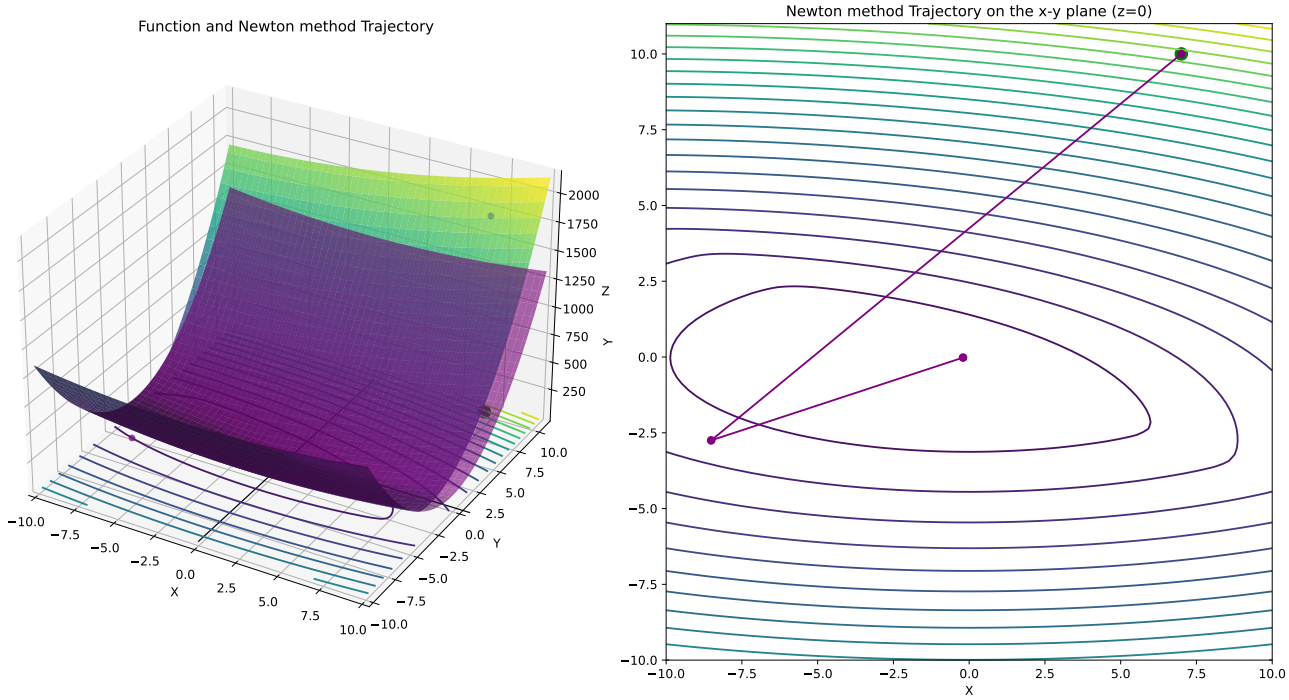


Figure: Newton method, step $T = 2$

Visualization

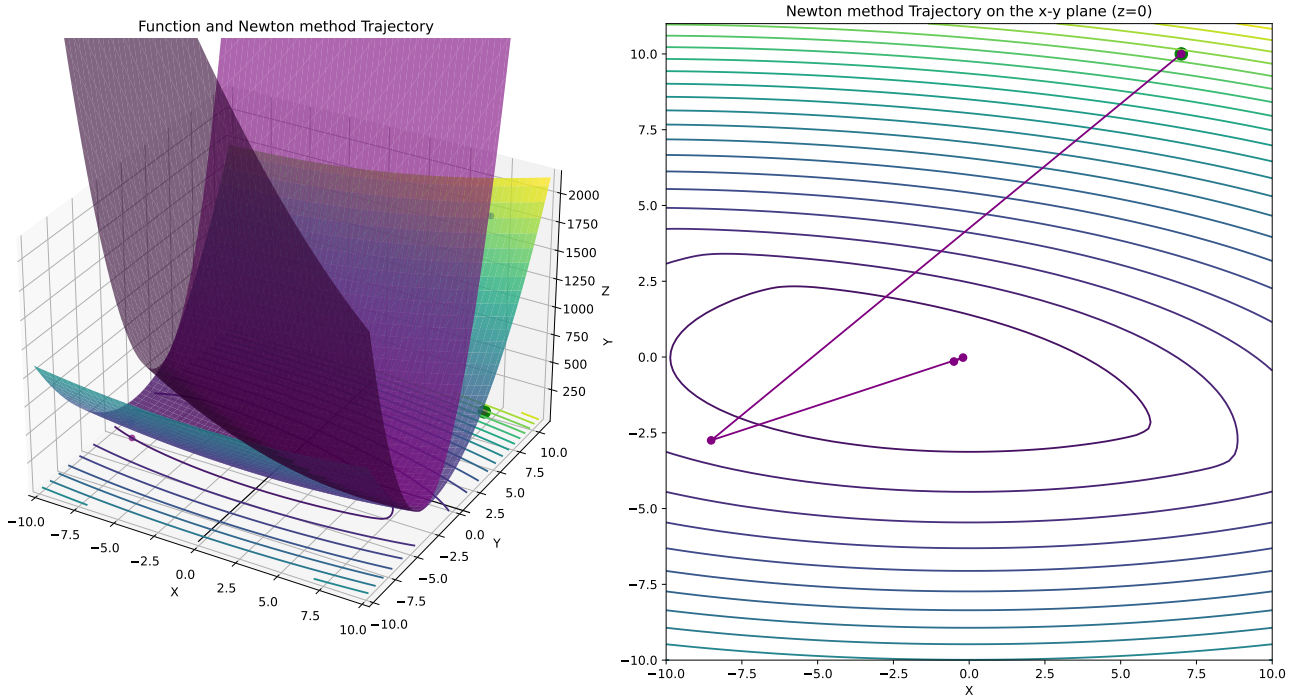


Figure: Newton method, step $T = 3$

Visualization

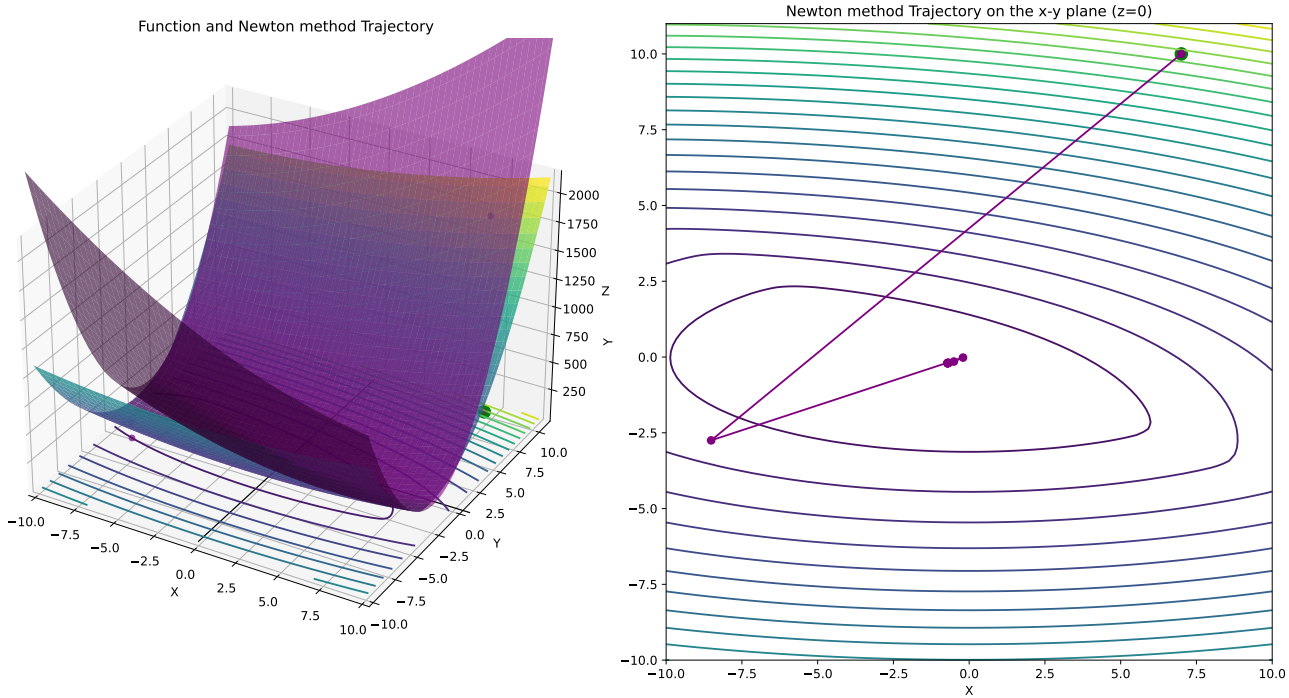
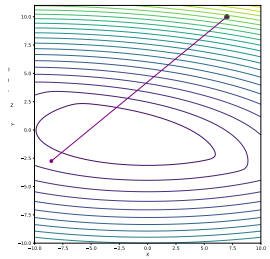
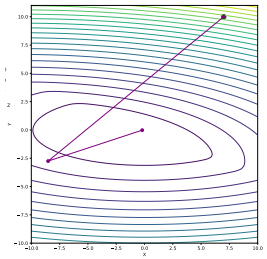
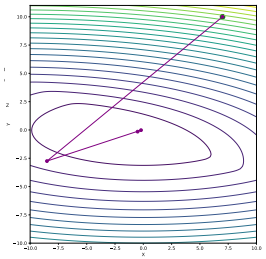
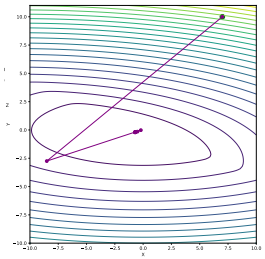
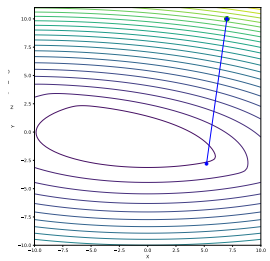
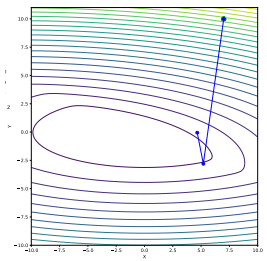
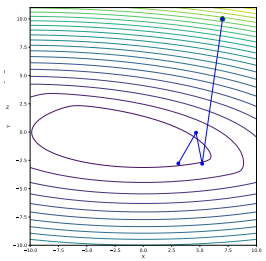
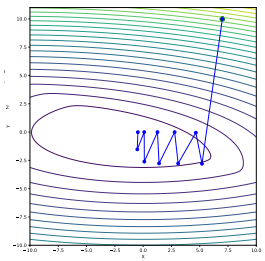
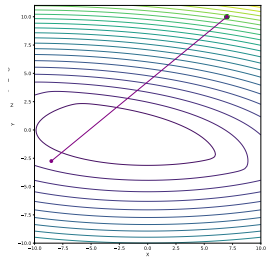
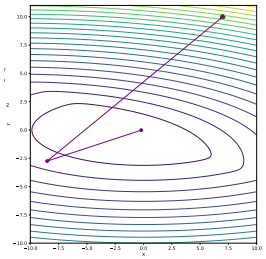
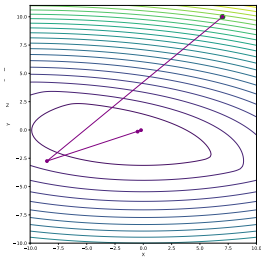
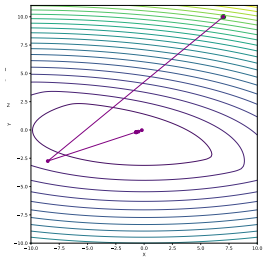
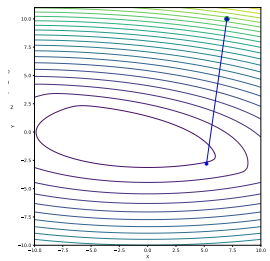
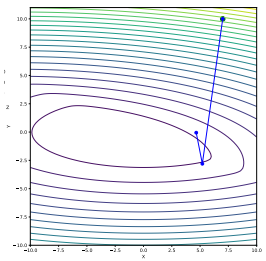
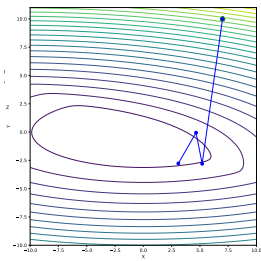
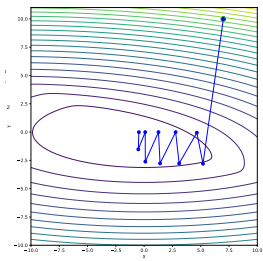


Figure: Newton method, step $T = 10$

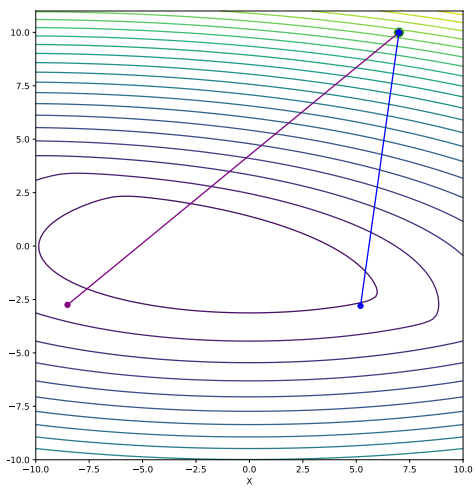
Comparison to GD

	$T = 1$	$T = 2$	$T = 3$	$T = 10$	
Newton					→ very fast convergence
GD					→ much slower

Comparison to GD

	$T = 1$	$T = 2$	$T = 3$	$T = 10$	
Newton					→ very fast convergence
GD					→ much slower

First step:



1. Gradient is orthogonal to the level set:

$$\theta_{t+1} \leftarrow \theta_t - \underbrace{\frac{1}{L}}_{\text{step}} \nabla f(\theta_t)$$

2. Newton step takes the curvature into account: "sees also how the gradient varies locally".

$$\theta_{t+1} \leftarrow \theta_t - \underbrace{\left(\nabla^2 f(\theta_t)\right)^{-1}}_{\text{pseudo-step}} \nabla f(\theta_t)$$

Theoretical result - Newton on quadratics

Case of quadratic functions.

Reminder: if there exists a matrix $A \in \mathbb{R}^{d \times d}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}$ such that

$$f : \theta \mapsto \frac{1}{2}\theta^T A\theta + b^\top \theta + c$$

then, for all $\theta \in \mathbb{R}^d$, then the second-order Taylor approximation is **exact** at all points: for all θ, θ_0

$$f(\theta) = \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation}}.$$

Theoretical result - Newton on quadratics

Case of quadratic functions.

Reminder: if there exists a matrix $A \in \mathbb{R}^{d \times d}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}$ such that

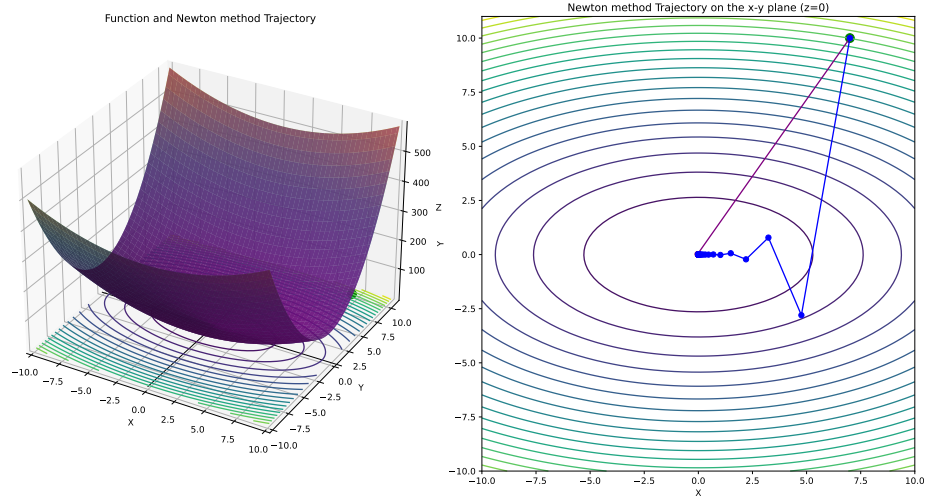
$$f : \theta \mapsto \frac{1}{2}\theta^T A\theta + b^\top \theta + c$$

then, for all $\theta \in \mathbb{R}^d$, then the second-order Taylor approximation is **exact** at all points: for all θ, θ_0

$$f(\theta) = \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation}}.$$

💡 **Newton method converges after one step:** for any starting point θ_0,

$$\theta_1 = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation}} = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} f(\theta) = \theta_\star$$



Theoretical result - Newton on quadratics

Case of quadratic functions.

Reminder: if there exists a matrix $A \in \mathbb{R}^{d \times d}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}$ such that

$$f : \theta \mapsto \frac{1}{2}\theta^T A\theta + b^\top \theta + c$$

then, for all $\theta \in \mathbb{R}^d$, then the second-order Taylor approximation is **exact** at all points: for all θ, θ_0

$$f(\theta) = \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation}}.$$

Newton method converges after one step: for any starting point θ_0,

$$\theta_1 = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation}} = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} f(\theta) = \theta_\star$$

$$= \theta_0 - \underbrace{A^{-1}}_{(\nabla^2 f(\theta_0))^{-1}} \underbrace{A(\theta_0 - \theta_\star)}_{\nabla f(\theta_0)} = \theta_\star$$

Theoretical result - Newton on quadratics

Case of quadratic functions.

Reminder: if there exists a matrix $A \in \mathbb{R}^{d \times d}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}$ such that

$$f : \theta \mapsto \frac{1}{2}\theta^T A \theta + b^\top \theta + c$$

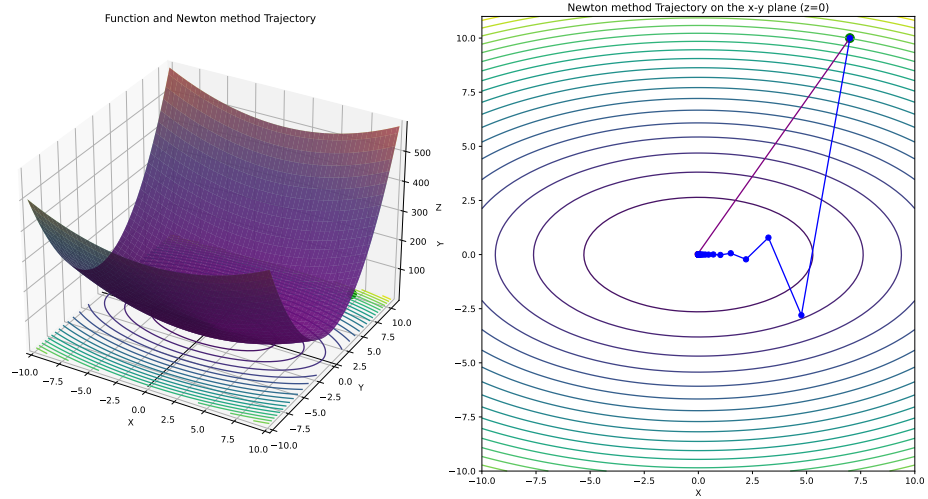
then, for all $\theta \in \mathbb{R}^d$, then the second-order Taylor approximation is **exact** at all points: for all θ, θ_0

$$f(\theta) = \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation}}.$$

Newton method converges after one step: for any starting point θ_0,

$$\theta_1 = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} \underbrace{f(\theta_0) + \langle \nabla f(\theta_0), \theta - \theta_0 \rangle + \frac{1}{2}(\theta - \theta_0)^\top \nabla^2 f(\theta_0)\ (\theta - \theta_0)}_{\text{Second-order approximation}} = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} f(\theta) = \theta_\star$$

$$= \theta_0 - \underbrace{A^{-1}}_{(\nabla^2 f(\theta_0))^{-1}} \underbrace{A(\theta_0 - \theta_\star)}_{\nabla f(\theta_0)} = \theta_\star$$



Theoretical result - Newton on smooth-convex functions

1. **Conditioning:** Newtons method is not affected by a problems conditioning κ .

2. **Converges extremely fast once a neighborhood has been reached**:

Proposition : Newton method – local convergence rate

If θ_0 is "close enough" to $\theta_\star$ (in a certain sense), and f is L-smooth, μ-strongly convex and $\nabla^2 f$ is M Lipschitz, then,

$$f(\theta_t) - f_\star \leqslant C_{\mu,L,M} \left(\frac{1}{2}\right)^{2^t} .$$

Theoretical result - Newton on smooth-convex functions

1. **Conditioning:** Newtons method is not affected by a problems conditioning κ .

2. **Converges extremely fast once a neighborhood has been reached**:

Proposition : Newton method – local convergence rate

If θ_0 is "close enough" to $\theta_\star$ (in a certain sense), and f is L-smooth, μ-strongly convex and $\nabla^2 f$ is M Lipschitz, then,

$$f(\theta_t) - f_\star \leqslant C_{\mu,L,M} \left(\frac{1}{2}\right)^{2^t} .$$

- $\rightsquigarrow$ error is **squared** at each step!
  Once a neighborhood of the solution has been reached, the excess risk decays at a quadratic rate: typically

Theoretical result - Newton on smooth-convex functions

1. **Conditioning:** Newtons method is not affected by a problems conditioning κ .

2. **Converges extremely fast once a neighborhood has been reached**:

Proposition : Newton method – local convergence rate

If θ_0 is "close enough" to $\theta_\star$ (in a certain sense), and f is L-smooth, μ-strongly convex and $\nabla^2 f$ is M Lipschitz, then,

$$f(\theta_t) - f_\star \leqslant C_{\mu,L,M} \left(\frac{1}{2}\right)^{2^t} .$$

- $\rightsquigarrow$ error is **squared** at each step!
  Once a neighborhood of the solution has been reached, the excess risk decays at a quadratic rate: typically

iteration	1	2	3	4	5	...
error	10^{-1}	10^{-2}	10^{-4}	10^{-8}	10^{-16}	...

- Machine precision is then obtained in a few steps.

Newton: drawbacks

1. ⚠ Complexity: each iteration requires $O(d^3)$ operations (solving a $d \times d$-linear system).
2. ⚠ Memory: each iteration requires $O(d^2)$ (Hessian size) storage.
   $\rightsquigarrow$ may simply be prohibitive in ML setups.

Newton: drawbacks

1. ⚠ Complexity: each iteration requires $O(d^3)$ operations (solving a $d \times d$-linear system).
2. ⚠ Memory: each iteration requires $O(d^2)$ (Hessian size) storage.
   $\rightsquigarrow$ may simply be prohibitive in ML setups.

3. Requires the function to be twice differentiable.
4. Requires the Hessian to be invertible (numerical problems may arise otherwise)

5. Requires convexity:
   $\rightsquigarrow$ ⚠ The sequence of iterates is the same if we change f into $-f$.
   $\rightsquigarrow$ ⚠ Newton method cannot distinguish minima and maxima!

Newton: drawbacks

1. ⚠ Complexity: each iteration requires $O(d^3)$ operations (solving a $d \times d$-linear system).
2. ⚠ Memory: each iteration requires $O(d^2)$ (Hessian size) storage.
   $\rightsquigarrow$ may simply be prohibitive in ML setups.

3. Requires the function to be twice differentiable.
4. Requires the Hessian to be invertible (numerical problems may arise otherwise)

5. Requires convexity:
   $\rightsquigarrow$ ⚠ The sequence of iterates is the same if we change f into $-f$.
   $\rightsquigarrow$ ⚠ Newton method cannot distinguish minima and maxima!

6. May diverge: no global; convergence is ensured.
   $\rightsquigarrow$ solutions exist damped-Newton, cubic-regularization, etc.

Newton: drawbacks

1. ⚠ Complexity: each iteration requires $O(d^3)$ operations (solving a $d \times d$-linear system).
2. ⚠ Memory: each iteration requires $O(d^2)$ (Hessian size) storage.
   ⇝ may simply be prohibitive in ML setups.

3. Requires the function to be twice differentiable.
4. Requires the Hessian to be invertible (numerical problems may arise otherwise)

5. Requires convexity:
   ⇝ ⚠ The sequence of iterates is the same if we change f into $-f$.
   ⇝ ⚠ Newton method cannot distinguish minima and maxima!

6. May diverge: no global; convergence is ensured.
   ⇝ solutions exist damped-Newton, cubic-regularization, etc.

💡 Extremely high precision, at a very high cost per iteration, only for convex smooth problems.
→ only useful in specific ML cases.

Quasi-Newton methods

Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - A_{t-1}^{-1} \nabla f(\theta_{t-1})$,
2. with $A_{t-1} = \nabla^2 f(\theta_{t-1})$
   - Complexity: $O(d^3)$ per iteration.

Quasi-Newton methods

Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - A_{t-1}^{-1} \nabla f(\theta_{t-1})$,
2. with $A_{t-1} = \nabla^2 f(\theta_{t-1})$
   - Complexity: $O(d^3)$ per iteration.

Quasi-Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$

Quasi-Newton methods

Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - A_{t-1}^{-1} \nabla f(\theta_{t-1})$,
2. with $A_{t-1} = \nabla^2 f(\theta_{t-1})$
   - Complexity: $O(d^3)$ per iteration.

Quasi-Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in R$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.

Quasi-Newton methods

Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - A_{t-1}^{-1} \nabla f(\theta_{t-1})$,
2. with $A_{t-1} = \nabla^2 f(\theta_{t-1})$
   - Complexity: $O(d^3)$ per iteration.

Quasi-Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in R$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t,

Quasi-Newton methods

Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - A_{t-1}^{-1} \nabla f(\theta_{t-1})$,
2. with $A_{t-1} = \nabla^2 f(\theta_{t-1})$
   - Complexity: $O(d^3)$ per iteration.
   - Remark: $\forall \zeta$ in a neighborhood of θ_t:

$$A_t(\zeta - \theta_t) = \nabla^2 f(\theta_t)(\zeta - \theta_t) \simeq \nabla f(\zeta) - \nabla f(\theta_t)$$

$$\text{in particular} \quad A_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} \simeq \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Quasi-Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in R$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t,

Quasi-Newton methods

Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - A_{t-1}^{-1} \nabla f(\theta_{t-1})$,
2. with $A_{t-1} = \nabla^2 f(\theta_{t-1})$
   - Complexity: $O(d^3)$ per iteration.
   - Remark: $\forall \zeta$ in a neighborhood of θ_t:

$$A_t(\zeta - \theta_t) = \nabla^2 f(\theta_t)(\zeta - \theta_t) \simeq \nabla f(\zeta) - \nabla f(\theta_t)$$

$$\text{in particular} \quad A_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} \simeq \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Quasi-Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in R$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Quasi-Newton methods

Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - A_{t-1}^{-1} \nabla f(\theta_{t-1})$,
2. with $A_{t-1} = \nabla^2 f(\theta_{t-1})$
   - Complexity: $O(d^3)$ per iteration.
   - Remark: $\forall \zeta$ in a neighborhood of θ_t:

$$A_t(\zeta - \theta_t) = \nabla^2 f(\theta_t)(\zeta - \theta_t) \simeq \nabla f(\zeta) - \nabla f(\theta_t)$$

$$\text{in particular} \quad A_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} \simeq \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Quasi-Newton method:

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in R$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

💡 In quasi-Newton's methods, the Newton direction is approximated by using only first order information (gradient)

$\rightarrow$ ⚠ **QN methods are first-order methods, (but with omplexity d^2)**

💡 Successive iterates and gradients yield second order information

Quasi-Newton's methods: (BFGS) algorithm

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in \mathbb{R}$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring "key property"

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction}(\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference}(\Delta G)_t}$$

Quasi-Newton's methods: (BFGS) algorithm

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in \mathbb{R}$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring "key property"

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Example: Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

$$B_t = B_{t-1} - \underbrace{\frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}}{(\Delta\theta)_t^T B_{t-1} (\Delta\theta)_t}} + \underbrace{\frac{(\Delta G)_t (\Delta G)_t^T}{(\Delta G)_t^T (\Delta\theta)_t}}.$$

Quasi-Newton's methods: (BFGS) algorithm

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in \mathbb{R}$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring "key property"

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Example: Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

$$B_t = B_{t-1} - \underbrace{\frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}}{(\Delta\theta)_t^T B_{t-1} (\Delta\theta)_t}}_{\text{rank-1 matrix}} + \underbrace{\frac{(\Delta G)_t (\Delta G)_t^T}{(\Delta G)_t^T (\Delta\theta)_t}}_{\text{rank-1 matrix}}.$$

✓ B_t is a rank-2 update.

Quasi-Newton's methods: (BFGS) algorithm

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in \mathbb{R}$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring "key property"

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Example: Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

$$B_t = B_{t-1} - \underbrace{\frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}}{(\Delta\theta)_t^T B_{t-1} (\Delta\theta)_t}}_{\text{rank-1 matrix}} + \underbrace{\frac{(\Delta G)_t (\Delta G)_t^T}{(\Delta G)_t^T (\Delta\theta)_t}}_{\text{rank-1 matrix}}.$$

- ✓ B_t is a rank-2 update.
- ✓ We have the "key property":

$$B_t(\Delta\theta)_t = B_{t-1}(\Delta\theta)_t - \frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}(\Delta\theta)_t}{(\Delta\theta)_t^T B_{t-1} (\Delta\theta)_t} + \frac{(\Delta G)_t (\Delta G)_t^T (\Delta\theta)_t}{(\Delta G)_t^T (\Delta\theta)_t}$$

Quasi-Newton's methods: (BFGS) algorithm

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in \mathbb{R}$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring "key property"

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Example: Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

$$B_t = B_{t-1} - \underbrace{\frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}}{(\Delta\theta)_t^T B_{t-1} (\Delta\theta)_t}}_{\text{rank-1 matrix}} + \underbrace{\frac{(\Delta G)_t (\Delta G)_t^T}{(\Delta G)_t^T (\Delta\theta)_t}}_{\text{rank-1 matrix}}.$$

- ✓ B_t is a rank-2 update.
- ✓ We have the "key property":

$$\begin{aligned} B_t(\Delta\theta)_t &= B_{t-1}(\Delta\theta)_t - \frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}(\Delta\theta)_t}{(\Delta\theta)_t^T B_{t-1}(\Delta\theta)_t} + \frac{(\Delta G)_t (\Delta G)_t^T (\Delta\theta)_t}{(\Delta G)_t^T (\Delta\theta)_t} \\ &= B_{t-1}(\Delta\theta)_t - B_{t-1}(\Delta\theta)_t + (\Delta G)_t \end{aligned}$$

Quasi-Newton's methods: (BFGS) algorithm

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in \mathbb{R}$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring "key property"

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction } (\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference } (\Delta G)_t}$$

Example: Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

$$B_t = B_{t-1} - \underbrace{\frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}}{(\Delta\theta)_t^T B_{t-1} (\Delta\theta)_t}}_{\text{rank-1 matrix}} + \underbrace{\frac{(\Delta G)_t (\Delta G)_t^T}{(\Delta G)_t^T (\Delta\theta)_t}}_{\text{rank-1 matrix}}.$$

- ✓ B_t is a rank-2 update.
- ✓ We have the "key property":

$$\begin{aligned} B_t(\Delta\theta)_t &= B_{t-1}(\Delta\theta)_t - \frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}(\Delta\theta)_t}{(\Delta\theta)_t^T B_{t-1}(\Delta\theta)_t} + \frac{(\Delta G)_t (\Delta G)_t^T (\Delta\theta)_t}{(\Delta G)_t^T (\Delta\theta)_t} \\ &= B_{t-1}(\Delta\theta)_t - B_{t-1}(\Delta\theta)_t + (\Delta G)_t \\ &= (\Delta G)_t \end{aligned}$$

Quasi-Newton's methods: (BFGS) algorithm

1. $\theta_t \leftarrow \theta_{t-1} - \eta_{t-1} B_{t-1}^{-1} \nabla f(\theta_{t-1})$
2. $\eta_{t-1} \in \mathbb{R}$ (line-search), B_{t-1} to be chosen such that
   - B_t is a rank-1/rank-2 update w.r.t. $B_{t-1} \rightarrow$ complexity: $O(d^2)$ per iteration.
   - B_t "mimics" A_t, by ensuring "key property"

$$B_t \underbrace{(\theta_t - \theta_{t-1})}_{\text{update direction}(\Delta\theta)_t} = \underbrace{\nabla f(\theta_t) - \nabla f(\theta_{t-1})}_{\text{gradient difference}(\Delta G)_t}$$

Example: Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

$$B_t = B_{t-1} - \underbrace{\frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}}{(\Delta\theta)_t^T B_{t-1} (\Delta\theta)_t}}_{\text{rank-1 matrix}} + \underbrace{\frac{(\Delta G)_t (\Delta G)_t^T}{(\Delta G)_t^T (\Delta\theta)_t}}_{\text{rank-1 matrix}}.$$

- ✓ B_t is a rank-2 update.
- ✓ We have the "key property":

$$\begin{aligned} B_t(\Delta\theta)_t &= B_{t-1}(\Delta\theta)_t - \frac{B_{t-1}(\Delta\theta)_t (\Delta\theta)_t^T B_{t-1}(\Delta\theta)_t}{(\Delta\theta)_t^T B_{t-1}(\Delta\theta)_t} + \frac{(\Delta G)_t (\Delta G)_t^T (\Delta\theta)_t}{(\Delta G)_t^T (\Delta\theta)_t} \\ &= B_{t-1}(\Delta\theta)_t - B_{t-1}(\Delta\theta)_t + (\Delta G)_t \\ &= (\Delta G)_t \end{aligned}$$

Considered as the state-of-the-art quasi-Newton's algorithm!
Extremely efficient in practice for smooth convex optimization! Often the default method!
Extension: **L-BFGS** reduces per-iteration complexity to Md, M extra hyper-parameter.

Newton and L-BFGS in practical applications? Ideal scenario

⚠ Methods above are deterministic.

⚠ ⚠ In supervised ML, computing the Hessian or gradient has a cost proportional to n, typically $O(nd^2 + d^3)$ for Newton.*

*extensions to stochastic case exist, see e.g., Stochastic-L-BFGS

Newton and L-BFGS in practical applications? Ideal scenario

⚠ Methods above are deterministic.
⚠ ⚠ In supervised ML, computing the Hessian or gradient has a cost proportional to n, typically $O(nd^2 + d^3)$ for Newton.*

- Ideal scenario ? Need convex and smooth problem, n not too large (deterministic method), d not too large for Newton

→ Logistic regression

*extensions to stochastic case exist, see e.g., Stochastic-L-BFGS

Newton and L-BFGS in practical applications? Ideal scenario

⚠ Methods above are deterministic.
⚠ ⚠ In supervised ML, computing the Hessian or gradient has a cost proportional to n, typically $O(nd^2 + d^3)$ for Newton.*

- Ideal scenario ? Need convex and smooth problem, n not too large (deterministic method), d not too large for Newton

→ Logistic regression

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

Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver, you might want to consider the following aspects:

- 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
- 'newton-cholesky' is a good choice for `n_samples` >> `n_features`, especially with one-hot encoded categorical features with rare categories. Note that it is limited to binary classification and the one-versus-rest reduction for multiclass classification. Be aware that the memory usage of this solver has a quadratic dependency on `n_features` because it explicitly computes the Hessian matrix.

Warning: The choice of the algorithm depends on the penalty chosen. Supported penalties by solver:

- 'lbfgs' - ['l2', None]
- 'liblinear' - ['l1', 'l2']
- 'newton-cg' - ['l2', None]
- 'newton-cholesky' - ['l2', None]
- 'sag' - ['l2', None]
- 'saga' - ['elasticnet', 'l1', 'l2', None]

Figure: Logistic-regression solvers - sklearn documentation - link

Newton and L-BFGS in practical applications? Ideal scenario

⚠ Methods above are deterministic.
⚠ ⚠ In supervised ML, computing the Hessian or gradient has a cost proportional to n, typically $O(nd^2 + d^3)$ for Newton.*

- Ideal scenario ? Need convex and smooth problem, n not too large (deterministic method), d not too large for Newton

→ Logistic regression

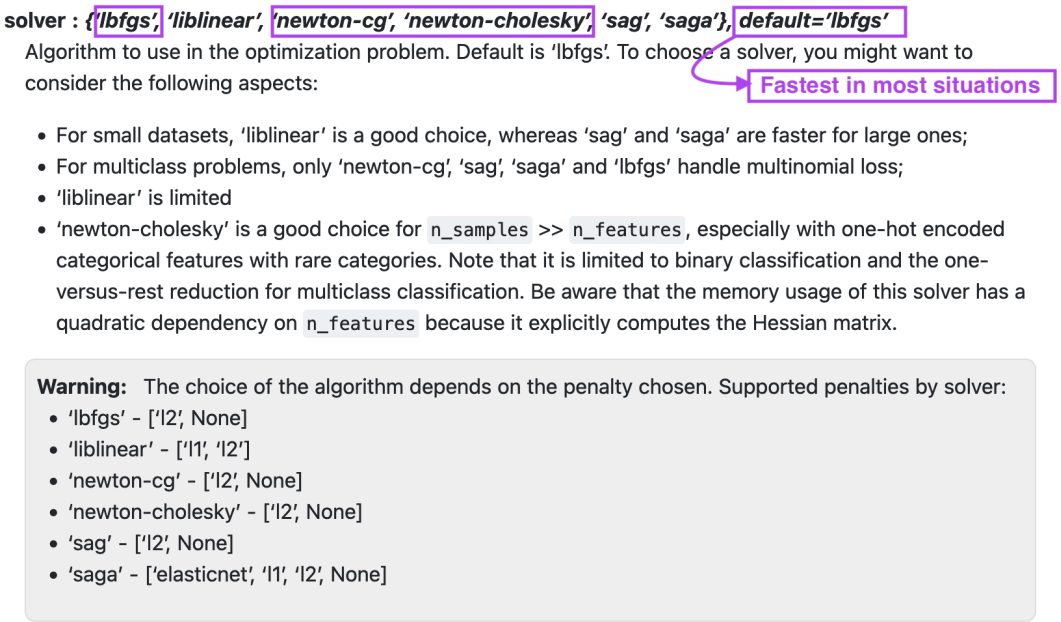


Figure: Logistic-regression solvers - sklearn documentation - link

Newton and L-BFGS in practical applications? Ideal scenario

⚠ Methods above are deterministic.

⚠ ⚠ In supervised ML, computing the Hessian or gradient has a cost proportional to n, typically $O(nd^2 + d^3)$ for Newton.*

- Ideal scenario ? Need convex and smooth problem, n not too large (deterministic method), d not too large for Newton

→ Logistic regression

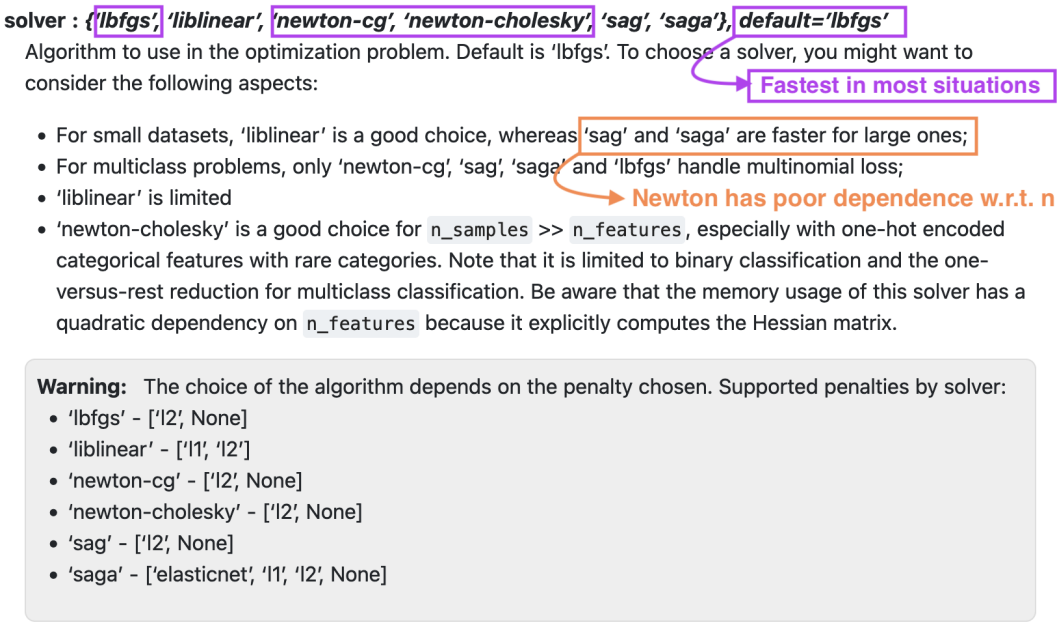


Figure: Logistic-regression solvers - sklearn documentation - link

Newton and L-BFGS in practical applications? Ideal scenario

⚠ Methods above are deterministic.

⚠ ⚠ In supervised ML, computing the Hessian or gradient has a cost proportional to n, typically $O(nd^2 + d^3)$ for Newton.*

- Ideal scenario ? Need convex and smooth problem, n not too large (deterministic method), d not too large for Newton

→ Logistic regression

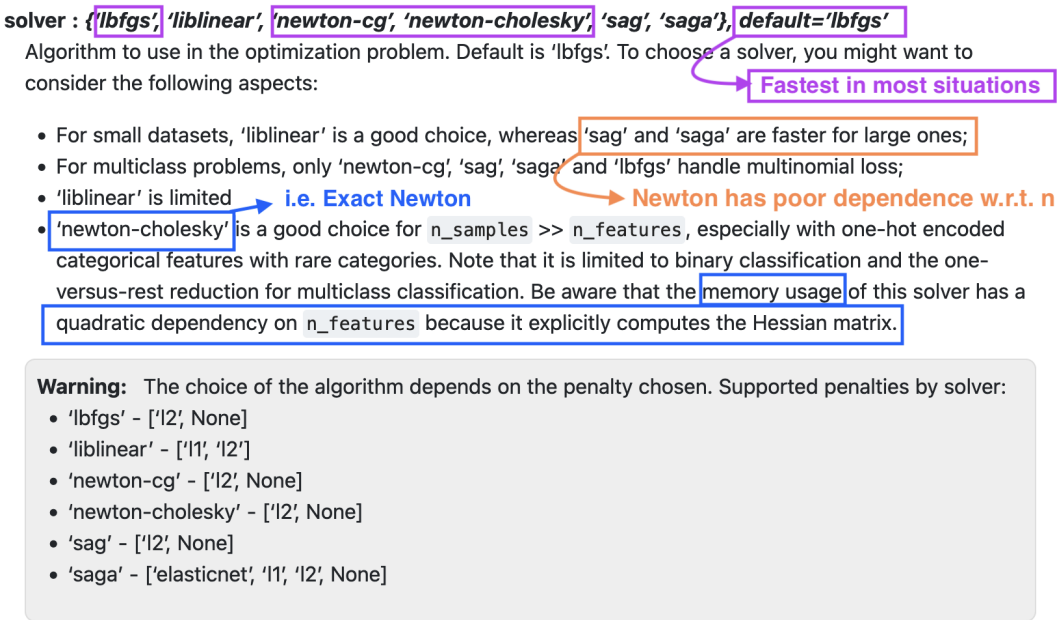


Figure: Logistic-regression solvers - sklearn documentation - link

Newton and L-BFGS in practical applications? Ideal scenario

⚠ Methods above are deterministic.

⚠ ⚠ In supervised ML, computing the Hessian or gradient has a cost proportional to n, typically $O(nd^2 + d^3)$ for Newton.*

- Ideal scenario ? Need convex and smooth problem, n not too large (deterministic method), d not too large for Newton

→ Logistic regression

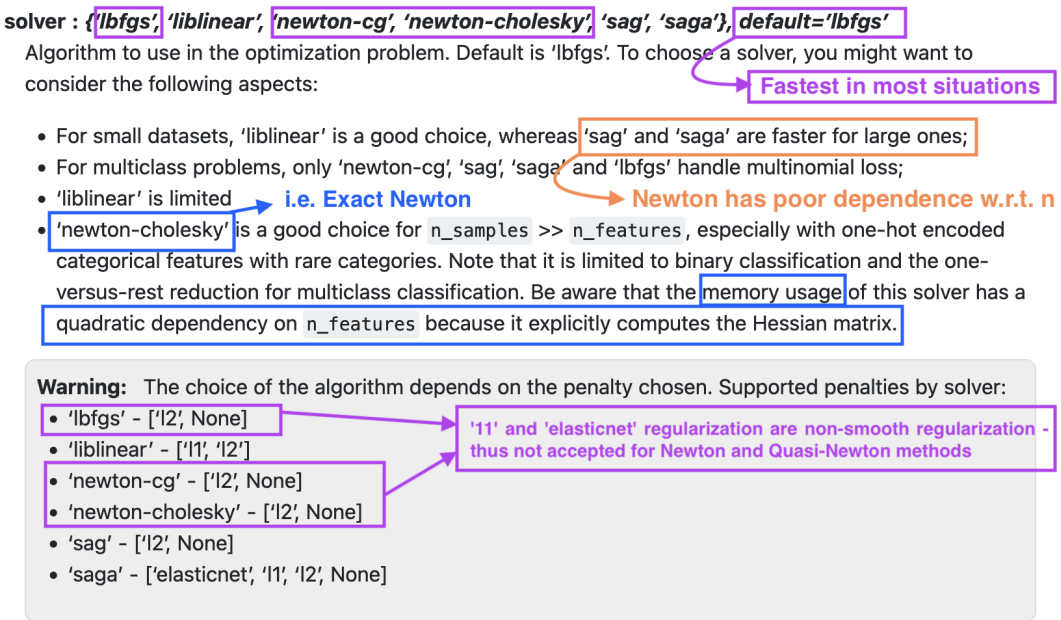


Figure: Logistic-regression solvers - sklearn documentation - link

*extensions to stochastic case exist, see e.g., Stochastic-L-BFGS

Second order methods: Summary

Newton: second-order method
Quasi-Newton: first-order method

Deterministic
(Stochastic extensions)

~~Variance reduction~~
~~Acceleration~~

Newton update
$\theta_{t+1} \leftarrow \theta_t - (\nabla^2 f(\theta_t))^{-1} \nabla f(\theta_t)$

BFGS update
$\theta_{t+1} \leftarrow \theta_t - \eta (B_t)^{-1} \nabla f(\theta_t)$

Hyperparameters:
T
T, η
$\rightsquigarrow \eta$ line search

Complexity/iter in ML:
$O(nd^2 + d^3)$
$O(nd + d^2)$

Intuition & motivation:
Challenge:

- gradient doesn't point in the best direction

→ Solution:

- Use Hessian information
- Minimize 2nd order approx.

Theory:

- Super fast rate for smooth and (strongly) convex functions, **locally**
- One-step convergence for quadratics

Take away:

→ (L)-BFGS is SOTA method for convex smooth optimization if

1. n, d are not too large
2. high precision is needed