

COMP 545: Advanced topics in optimization

From simple to complex ML systems

Lecture 3

Overview

- In the last lecture, we:
 - Talked about a bit of smooth non-convex and convex optimization
 - Worked in practice and theory with gradient descent
 - Discussed the limits and convergence rates of gradient descent

Overview

- In the last lecture, we:
 - Talked about a bit of smooth non-convex and convex optimization
 - Worked in practice and theory with gradient descent
 - Discussed the limits and convergence rates of gradient descent
- Often, gradient descent is not sufficient in practice. In this lecture, we will:
 - Discuss **alternatives to gradient descent**
 - Discuss cases where the above methods are problematic
 - Discuss gradient descent versions that somehow **accelerate convergence**

From previous lecture: lower bounds

- For objectives with Lipschitz continuous gradients:

$$f(x_t) - f(x^*) \geq \frac{3L\|x_0 - x^*\|_2^2}{32(t+1)^2}$$

(Under these assumptions, and using only gradients, we cannot achieve better than $O\left(\frac{1}{t^2}\right)$)

- In addition, for objectives that are strongly convex:

$$\|x_t - x^*\|_2^2 \geq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^{2t} \|x_0 - x^*\|_2^2$$

$$\kappa := \frac{L}{\mu}$$

(The case we described has near optimal exponent, but does not involve the square root of κ)

From previous lecture: lower bounds

- For objectives with Lipschitz continuous gradients:

$$f(x_t) - f(x^*) \geq \frac{3L\|x_0 - x^*\|_2^2}{32(t + 1)^2}$$

(Under these assumptions, and using only gradients, we cannot achieve better than $O\left(\frac{1}{t^2}\right)$)

- In addition, for objectives that are strongly convex:

$$\|x_t - x^*\|_2^2 \geq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^{2t} \|x_0 - x^*\|_2^2 \quad \kappa := \frac{L}{\mu}$$

(The case we described has near optimal exponent, but does not involve the square root of κ)

Can we do better if we use more information?

The notorious Newton's method

- Remember the second-order Taylor expansion:

$$f(x + \Delta x) \approx f(x) + \langle \nabla f(x), \Delta x \rangle + \frac{1}{2} \langle \nabla^2 f(x) \Delta x, \Delta x \rangle$$

The notorious Newton's method

- Remember the second-order Taylor expansion:

$$f(x + \Delta x) \approx f(x) + \langle \nabla f(x), \Delta x \rangle + \frac{1}{2} \langle \nabla^2 f(x) \Delta x, \Delta x \rangle$$

- Taking the derivative and setting to zero:

$$\nabla_{\Delta x} f(x + \Delta x) = 0 \quad \Rightarrow \quad \nabla f(x) + \nabla^2 f(x) \Delta x = 0 \quad \Rightarrow \quad \Delta x = -(\nabla^2 f(x))^{-1} \nabla f(x)$$

The notorious Newton's method

- Remember the second-order Taylor expansion:

$$f(x + \Delta x) \approx f(x) + \langle \nabla f(x), \Delta x \rangle + \frac{1}{2} \langle \nabla^2 f(x) \Delta x, \Delta x \rangle$$

- Taking the derivative and setting to zero:

$$\nabla_{\Delta x} f(x + \Delta x) = 0 \quad \Rightarrow \quad \nabla f(x) + \nabla^2 f(x) \Delta x = 0 \quad \Rightarrow \quad \Delta x = -(\nabla^2 f(x))^{-1} \nabla f(x)$$

- **Newton's iteration:**

$$x_{t+1} = x_t - \eta H_t^{-1} \nabla f(x_t), \quad H_t := \nabla^2 f(x_t)$$

The notorious Newton's method

- Remember the second-order Taylor expansion:

$$f(x + \Delta x) \approx f(x) + \langle \nabla f(x), \Delta x \rangle + \frac{1}{2} \langle \nabla^2 f(x) \Delta x, \Delta x \rangle$$

- Taking the derivative and setting to zero:

$$\nabla_{\Delta x} f(x + \Delta x) = 0 \quad \Rightarrow \quad \nabla f(x) + \nabla^2 f(x) \Delta x = 0 \quad \Rightarrow \quad \Delta x = -(\nabla^2 f(x))^{-1} \nabla f(x)$$

- **Newton's iteration:**

$$x_{t+1} = x_t - \eta H_t^{-1} \nabla f(x_t), \quad H_t := \nabla^2 f(x_t)$$

- Theory dictates even $\eta = 1$; often this is too optimistic, we use $\eta < 1$

(Damped Newton's method)

The notorious Newton's method

Demo

Guarantees of Newton's method

$$\min_{x \in \mathbb{R}^p} f(x)$$

“Assume the objective is has Lipschitz continuous Hessians. Also, assume that the initial point is close enough to the optimal point:

$$\|x_0 - x^*\|_2 < \frac{2\mu}{3M} \quad \text{where} \quad \nabla^2 f(x^*) \succeq \mu I \quad \text{and} \quad \|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq M \|x - y\|_2$$

$$x_{t+1} = x_t - (\nabla^2 f(x_t))^{-1} \nabla f(x_t)$$

converges quadratically according to:

$$\|x_{t+1} - x^*\|_2 \leq \frac{M \|x_t - x^*\|_2^2}{2(\mu - M \|x_t - x^*\|_2)} \quad “$$

Guarantees of Newton's method

Local convergence guarantees
Assumes no convexity – but assumes good initialization

$$\min_{x \in \mathbb{R}^p} f(x)$$

“Assume the objective is has Lipschitz continuous Hessians. Also, assume that the initial point is close enough to the optimal point:

$$\|x_0 - x^*\|_2 < \frac{2\mu}{3M} \quad \text{where} \quad \nabla^2 f(x^*) \succeq \mu I \quad \text{and} \quad \|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq M \|x - y\|_2$$

$$x_{t+1} = x_t - (\nabla^2 f(x_t))^{-1} \nabla f(x_t)$$

converges quadratically according to:

$$\|x_{t+1} - x^*\|_2 \leq \frac{M \|x_t - x^*\|_2^2}{2(\mu - M \|x_t - x^*\|_2)} \quad “$$

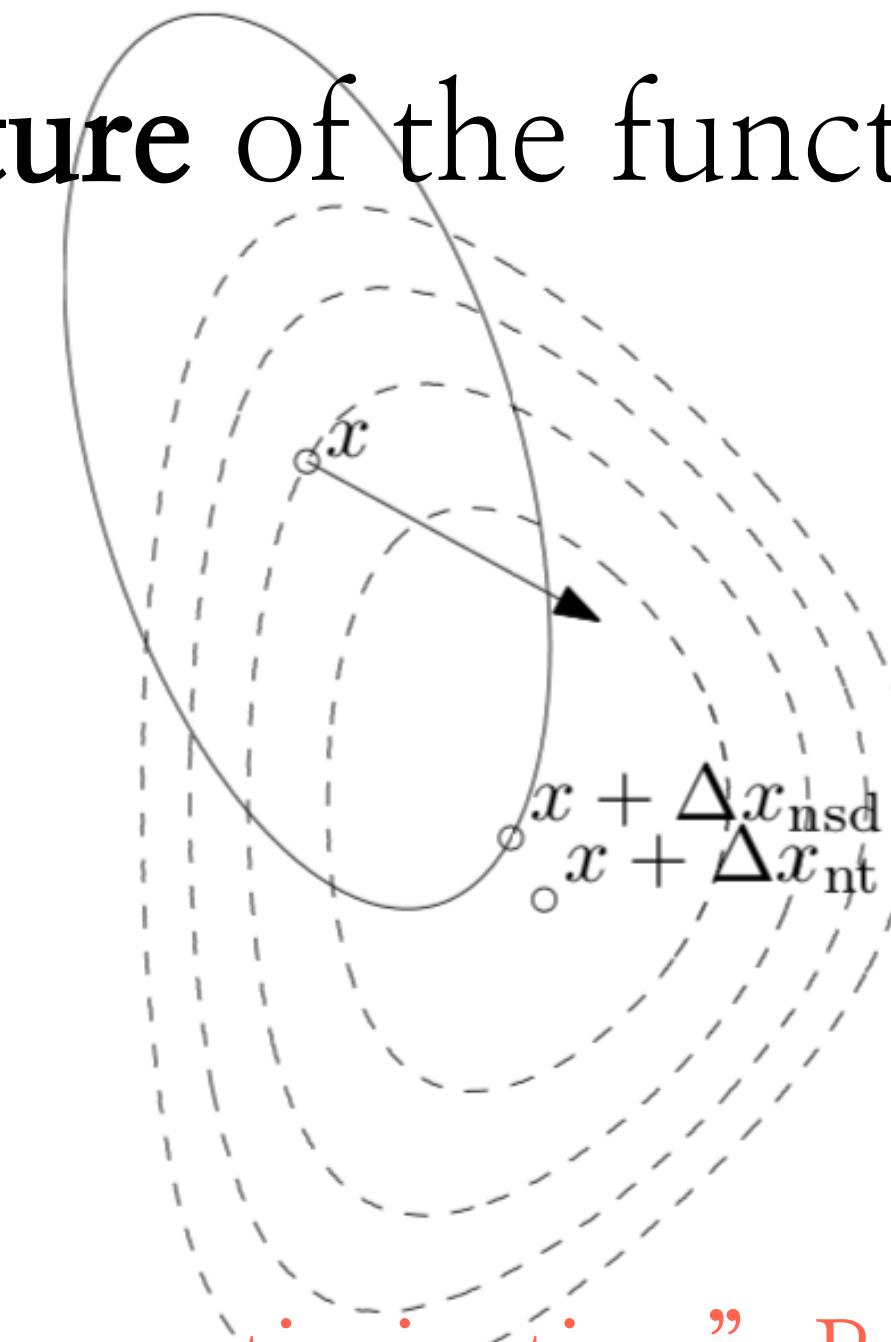
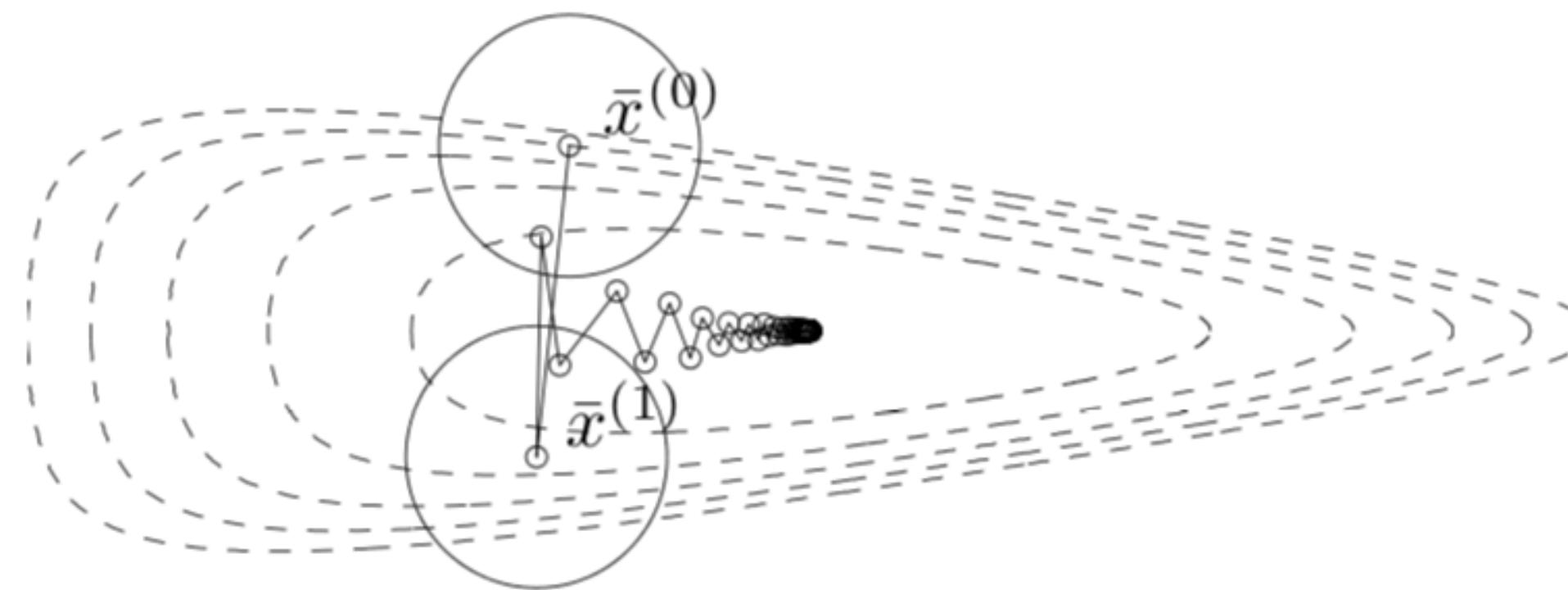
Guarantees of Newton's method

Whiteboard

General comments of Newton's method

General comments of Newton's method

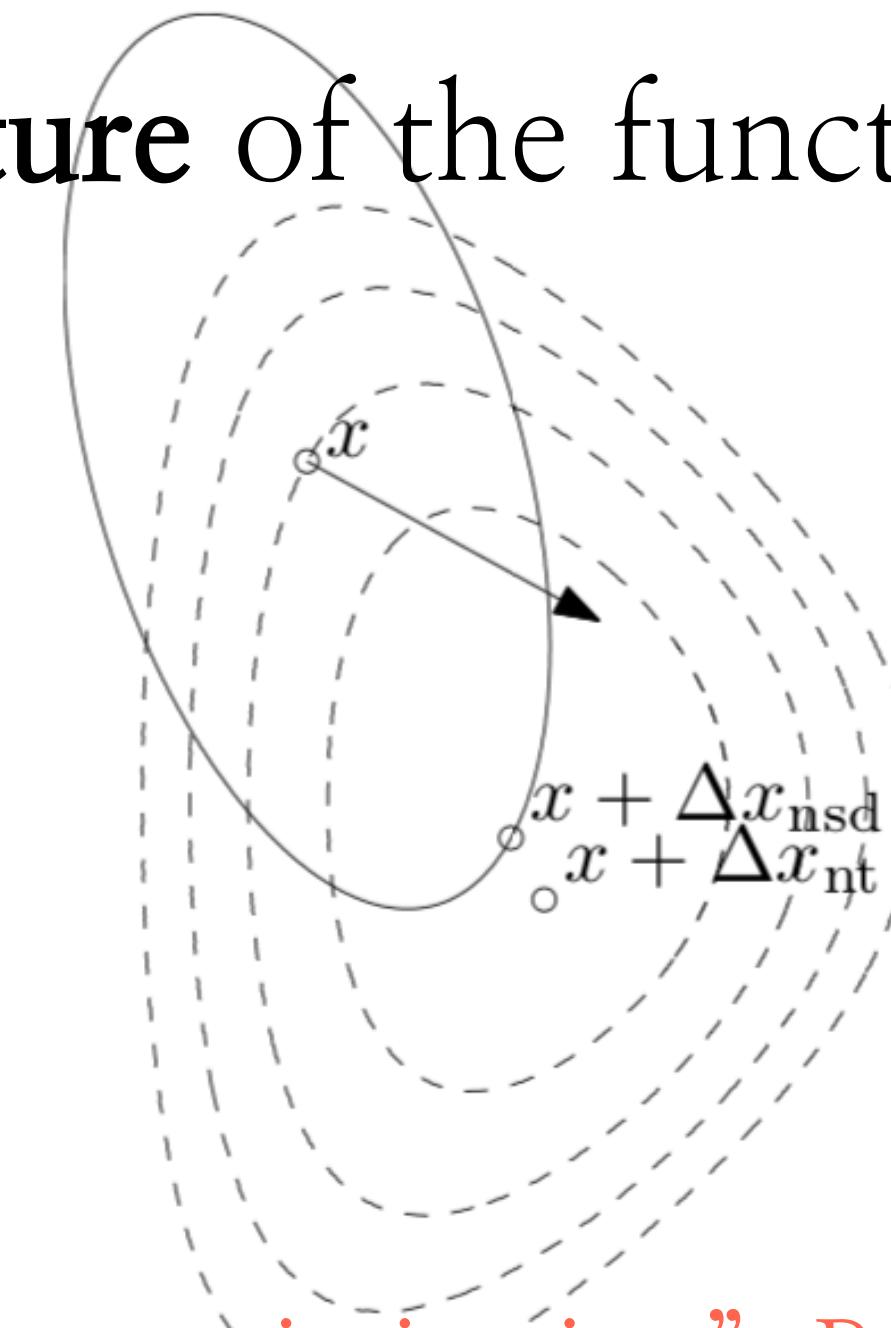
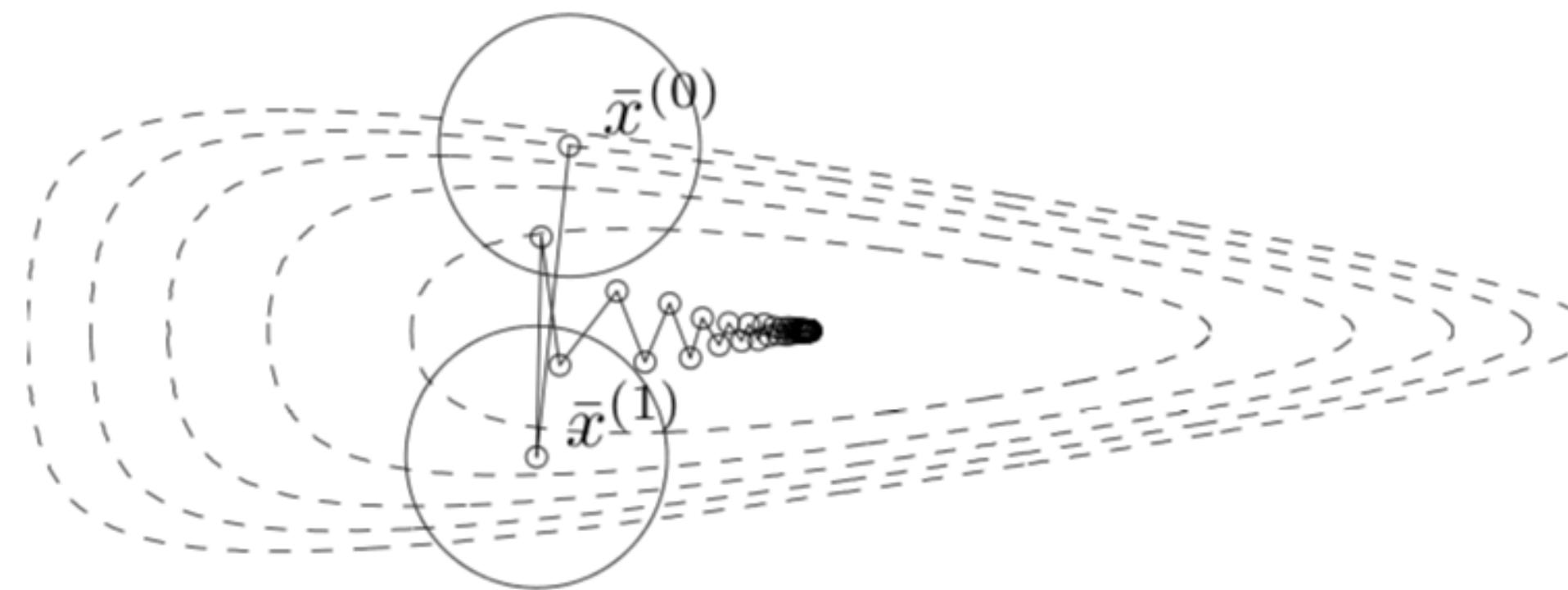
- Newton's method exploits the **local curvature** of the function



“Convex optimization”, Boyd and Vandenberghe

General comments of Newton's method

- Newton's method exploits the **local curvature** of the function

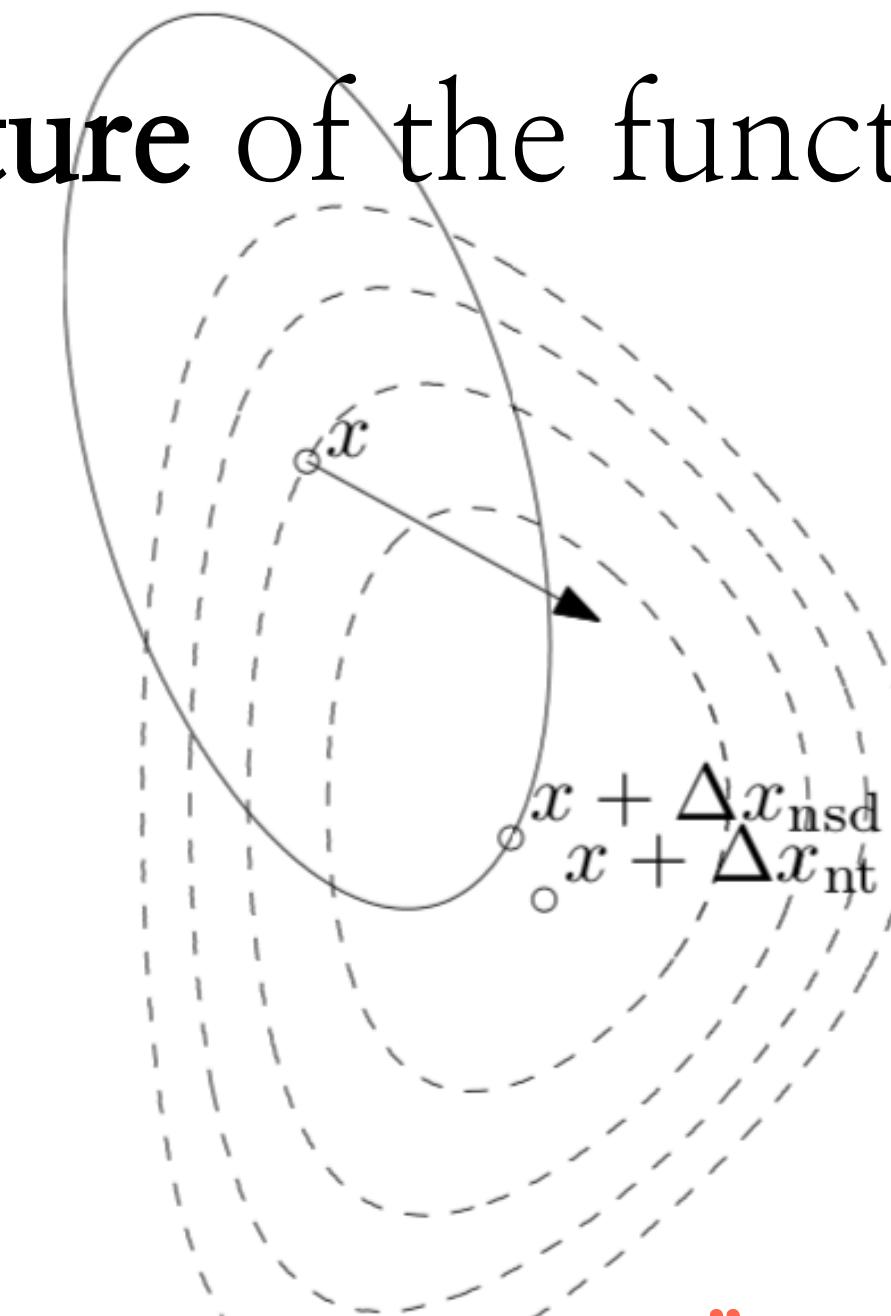
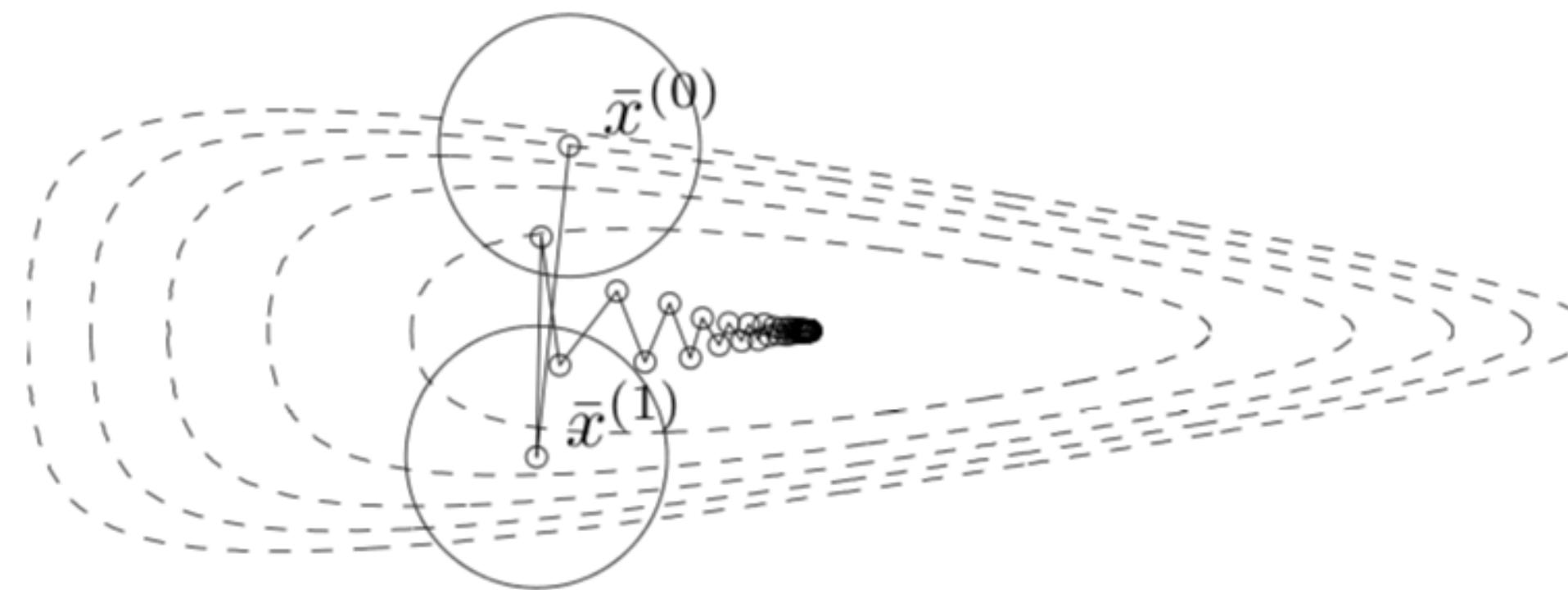


“Convex optimization”, Boyd and Vandenberghe

- Each iteration is **more computationally expensive**

General comments of Newton's method

- Newton's method exploits the **local curvature** of the function

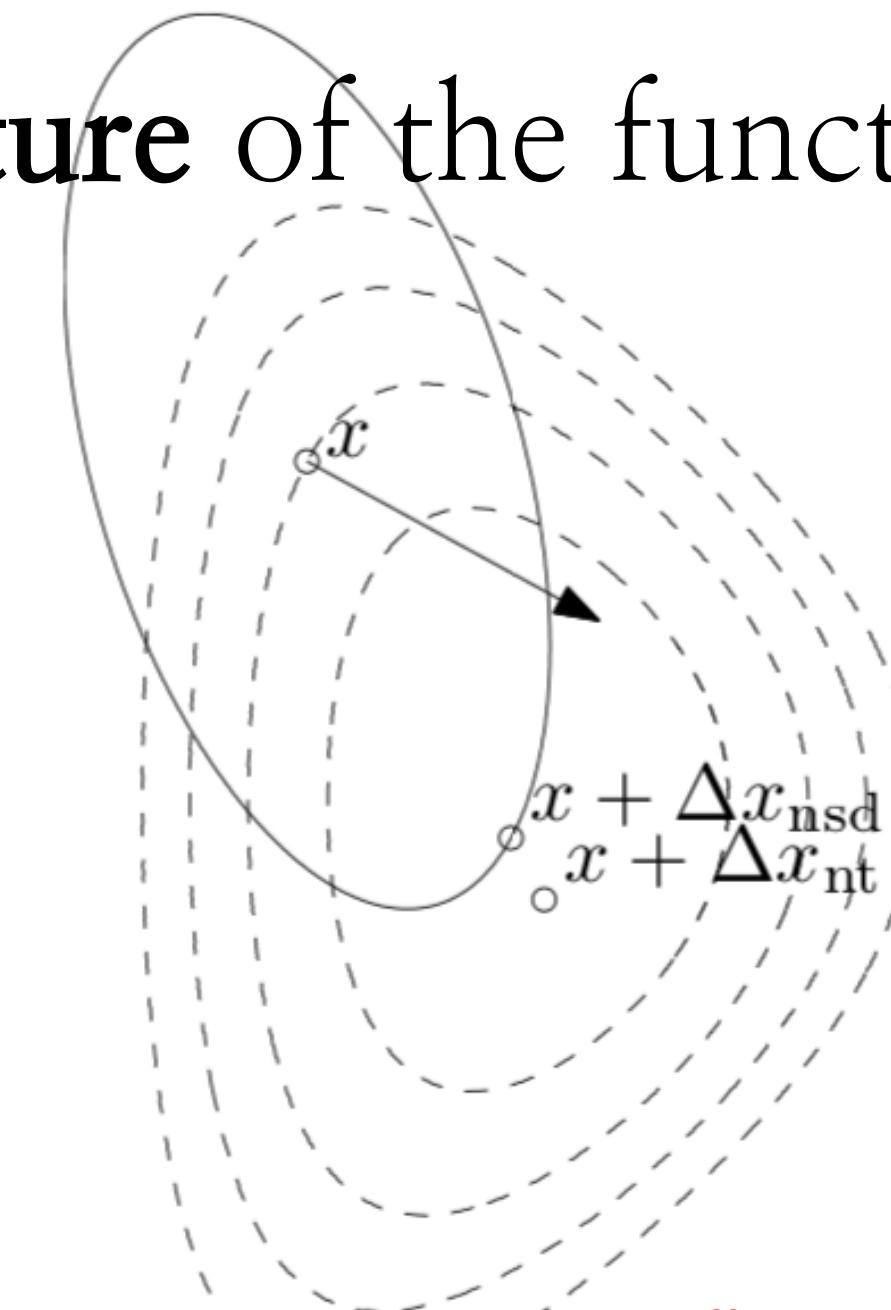
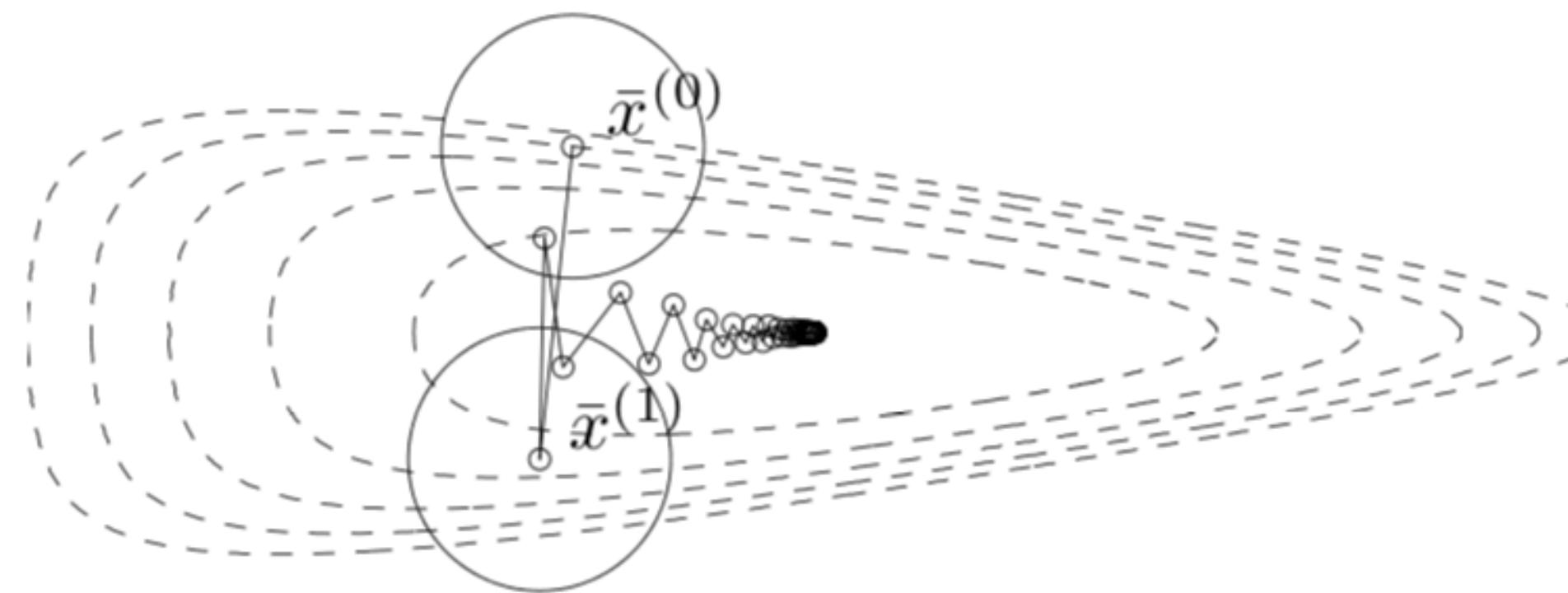


“Convex optimization”, Boyd and Vandenberghe

- Each iteration is **more computationally expensive**
- Theory **assumes a good initial point** for quadratic convergence
(We often observe a two-phase behavior: A linear convergence at first, and then a quadratic one)

General comments of Newton's method

- Newton's method exploits the **local curvature** of the function



“Convex optimization”, Boyd and Vandenberghe

- Each iteration is **more computationally expensive**
- Theory **assumes a good initial point** for quadratic convergence
(We often observe a two-phase behavior: A linear convergence at first, and then a quadratic one)
- Useful for exact solutions; not often the situation in machine learning

Between gradient descent and Newton's method

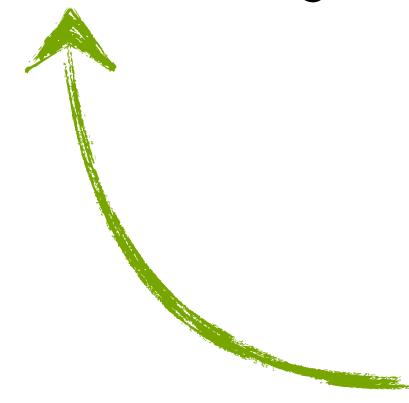
- Quasi–Newton methods

$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$

Between gradient descent and Newton's method

- Quasi–Newton methods

$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$

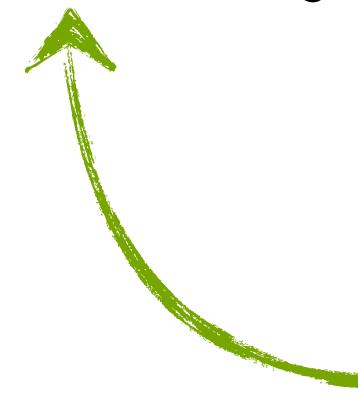


Approximation of the
inverse Hessian

Between gradient descent and Newton's method

- Quasi–Newton methods

$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$



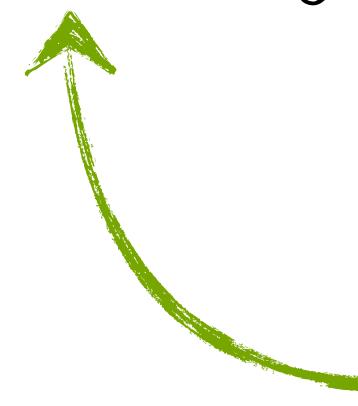
Approximation of the
inverse Hessian

- “Quasi–Newton” reveals that we want to avoid second–order calculations

Between gradient descent and Newton's method

- Quasi–Newton methods

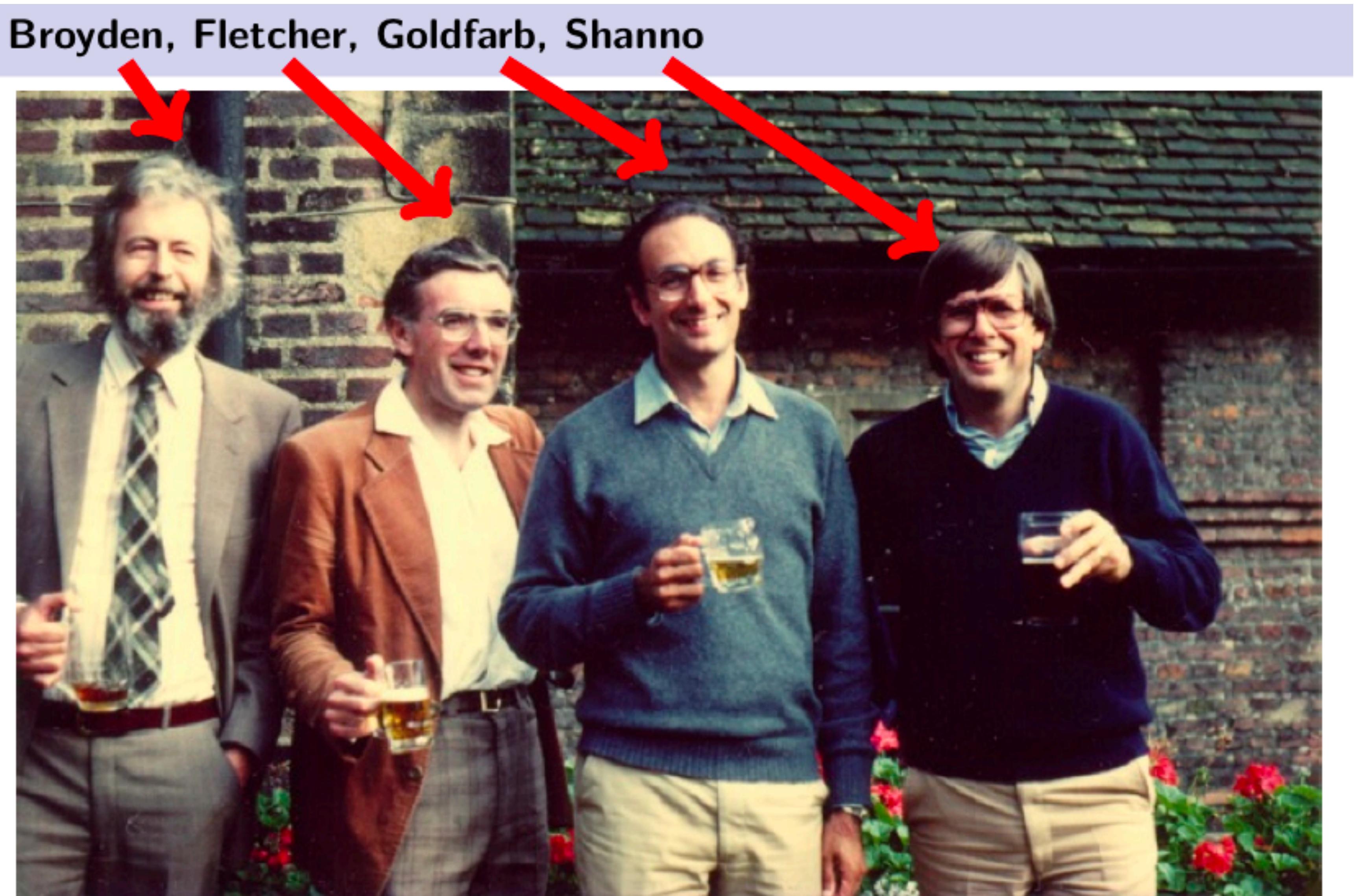
$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$



Approximation of the
inverse Hessian

- “Quasi–Newton” reveals that we want to avoid second–order calculations
- There are various ways to construct this approximation
 - (L)–BFGS approximation
 - SR1 approximation

The BFGS method



The BFGS method

- Quadratic approximations around current point

$$g_t(\Delta x) := f(x_t) + \langle \nabla f(x_t), \Delta x \rangle + \frac{1}{2} \langle H_t \Delta x, \Delta x \rangle$$

The BFGS method

- Quadratic approximations around current point


$$g_t(\Delta x) := f(x_t) + \langle \nabla f(x_t), \Delta x \rangle + \frac{1}{2} \langle H_t \Delta x, \Delta x \rangle$$

Local quadratic
approx.

The BFGS method

- Quadratic approximations around current point

$$g_t(\Delta x) := f(x_t) + \langle \nabla f(x_t), \Delta x \rangle + \frac{1}{2} \langle H_t \Delta x, \Delta x \rangle$$

Local quadratic
approx.

The direction we take
as in $x_{t+1} = x_t + \Delta x$



The BFGS method

- Quadratic approximations around current point

$$g_t(\Delta x) := f(x_t) + \langle \nabla f(x_t), \Delta x \rangle + \frac{1}{2} \langle H_t \Delta x, \Delta x \rangle$$

Local quadratic
approx.

The direction we take
as in $x_{t+1} = x_t + \Delta x$

We look for an
approximation of the
Hessian

The BFGS method

- Quadratic approximations around current point

$$g_t(\Delta x) := f(x_t) + \langle \nabla f(x_t), \Delta x \rangle + \frac{1}{2} \langle H_t \Delta x, \Delta x \rangle$$

Local quadratic
approx.

The direction we take
as in $x_{t+1} = x_t + \Delta x$

We look for an
approximation of the
Hessian

- Instead of estimating from scratch H_{t+1} , we require the new model $g_{t+1}(\cdot)$ satisfy two gradient conditions:

$\nabla g_{t+1}(0) = \nabla f(x_{t+1})$ (i.e., the new approximation should give back the gradient when no update step is performed)

The BFGS method

- Quadratic approximations around current point

$$g_t(\Delta x) := f(x_t) + \langle \nabla f(x_t), \Delta x \rangle + \frac{1}{2} \langle H_t \Delta x, \Delta x \rangle$$

Local quadratic
approx.

The direction we take
as in $x_{t+1} = x_t + \Delta x$



We look for an
approximation of the
Hessian

- Instead of estimating from scratch H_{t+1} , we require the new model $g_{t+1}(\cdot)$ satisfy two gradient conditions:

$$\nabla g_{t+1}(0) = \nabla f(x_{t+1}) \quad (\text{i.e., the new approximation should give back the gradient when no update step is performed})$$

$$\nabla g_{t+1}(-\Delta x) = \nabla f(x_t) \quad (\text{i.e., we take the opposite step and compute the gradient, the latter should match the gradient of the previous quad. approx.})$$

The BFGS method

- Secant equation

$$\nabla g_{t+1}(-\Delta x) = \nabla f(x_t) \longrightarrow H_{t+1}\Delta x = \nabla f(x_{t+1}) - \nabla f(x_t)$$

(Some of you might have seen the expression $H_{t+1}s_t = y_k$)

The BFGS method

- Secant equation

$$\nabla g_{t+1}(-\Delta x) = \nabla f(x_t) \longrightarrow H_{t+1}\Delta x = \nabla f(x_{t+1}) - \nabla f(x_t)$$

(Some of you might have seen the expression $H_{t+1}s_t = y_k$)

- Requirement: $H_{t+1} \succ 0 \rightarrow \Delta x^\top (\nabla f(x_{t+1}) - \nabla f(x_t)) > 0$

(Why?)

The BFGS method

- Secant equation

$$\nabla g_{t+1}(-\Delta x) = \nabla f(x_t) \longrightarrow H_{t+1}\Delta x = \nabla f(x_{t+1}) - \nabla f(x_t)$$

(Some of you might have seen the expression $H_{t+1}s_t = y_k$)

- Requirement: $H_{t+1} \succ 0 \rightarrow \Delta x^\top (\nabla f(x_{t+1}) - \nabla f(x_t)) > 0$

(Why?)

- How many H_{t+1} satisfy this? Infinite!

(How do we choose which one?)

The BFGS method

- Secant equation

$$\nabla g_{t+1}(-\Delta x) = \nabla f(x_t) \longrightarrow H_{t+1}\Delta x = \nabla f(x_{t+1}) - \nabla f(x_t)$$

(Some of you might have seen the expression $H_{t+1}s_t = y_k$)

- Requirement: $H_{t+1} \succ 0 \rightarrow \Delta x^\top (\nabla f(x_{t+1}) - \nabla f(x_t)) > 0$

(Why?)

- How many H_{t+1} satisfy this? Infinite!

(How do we choose which one?)

- By solving:

$$\min_{H \succ 0} \|H - H_t\|_F^2 \quad \longleftarrow \quad \text{(Intuition?)}$$

$$\text{s.t. } H = H^\top,$$

$$H\Delta x = \nabla f(x_t) - \nabla f(x_{t-1})$$

The BFGS method

- The BFGS method goes a bit further:

$$\begin{aligned} & \min_{B \succ 0} \|B - B_t\|_F^2 \\ \text{s.t. } & B = B^\top, \\ & \Delta x = B (\nabla f(x_t) - \nabla f(x_{t-1})) \end{aligned}$$

Approximates directly the inverse!

The BFGS method

- The BFGS method goes a bit further:

Approximates directly the inverse!

$$\begin{aligned} \min_{B \succ 0} \quad & \|B - B_t\|_F^2 \\ \text{s.t.} \quad & B = B^\top, \\ & \Delta x = B (\nabla f(x_t) - \nabla f(x_{t-1})) \end{aligned}$$

- The BFGS method has an easy closed form solution:

$$B_{t+1} = \left(I - \frac{s_t y_t^\top}{s_t^\top y_t} \right) B_t \left(I - \frac{y_t s_t^\top}{s_t^\top y_t} \right) + \frac{s_t s_t^\top}{s_t^\top y_t}$$

$$s_t := \Delta x$$

$$y_t := \nabla f(x_{t+1}) - \nabla f(x_t)$$



(Only inner product/outer product computations!)
(Only uses gradient information)

The SR1 method

- SR1 = Symmetric-Rank-1 update (in contrast to BFGS which is rank-2)

The SR1 method

- SR1 = Symmetric-Rank-1 update (in contrast to BFGS which is rank-2)
- Find H_{t+1} such that

$$H_{t+1} = H_t + \sigma v v^\top \quad \text{and secant equation is satisfied}$$

(Rank-1 update)

$$\sigma \in \{\pm 1\}$$

The SR1 method

- SR1 = Symmetric-Rank-1 update (in contrast to BFGS which is rank-2)
- Find H_{t+1} such that

$$H_{t+1} = H_t + \sigma v v^\top \quad \text{and secant equation is satisfied}$$

(Rank-1 update)

- SR1 rule:

$$B_{t+1} = B_t + \frac{(s_t - B_t y_t)(s_t - B_t y_t)^\top}{(s_t - B_t y_t)^\top y_t}$$

$$\sigma \in \{\pm 1\}$$

The SR1 method

- SR1 = Symmetric-Rank-1 update (in contrast to BFGS which is rank-2)
- Find H_{t+1} such that

$$H_{t+1} = H_t + \sigma v v^\top \quad \text{and secant equation is satisfied}$$

(Rank-1 update)

$$\sigma \in \{\pm 1\}$$

- SR1 rule:

$$B_{t+1} = B_t + \frac{(s_t - B_t y_t)(s_t - B_t y_t)^\top}{(s_t - B_t y_t)^\top y_t}$$

- No guarantee for positive definiteness!

- Might be useful to generate indefinite Hessian approximations in non convex optimization

(Could be a project proposal)

For the sake of saving lecture time

$$\|x_{k+1} - x^*\|_2 \leq c_k \|x_k - x^*\|_2 \quad \text{where} \quad c_k \rightarrow 0$$

For the sake of saving lecture time

$$\|x_{k+1} - x^*\|_2 \leq c_k \|x_k - x^*\|_2 \quad \text{where} \quad c_k \rightarrow 0$$

No theory or demo

(But willing to prepare some if
people get interested)

For the sake of saving lecture time

$$\|x_{k+1} - x^*\|_2 \leq c_k \|x_k - x^*\|_2 \quad \text{where} \quad c_k \rightarrow 0$$

No theory or demo

(But willing to prepare some if people get interested)

- Have in mind the formula:

$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$



Preconditioner matrix

Instead of forming higher order approximations..

..can we use 0-th order information?

0-th order optimization

- Some examples: Bisection method, genetic algorithms, simulated annealing
Metropolis methods..

0-th order optimization

- Some examples: Bisection method, genetic algorithms, simulated annealing Metropolis methods..
- There are problems where **we don't have access to gradients, or are computationally expensive to compute**

0-th order optimization

- Some examples: Bisection method, genetic algorithms, simulated annealing Metropolis methods..
- There are problems where **we don't have access to gradients, or are computationally expensive to compute**
- Here we will briefly describe the finite differences method:

$$x_{t+1} = x_t - \eta \frac{f(x_t + \mu_t u) - f(x_t)}{\mu_t} \cdot u$$

(we have access to function evaluations)

0-th order optimization

- Some examples: Bisection method, genetic algorithms, simulated annealing Metropolis methods..
- There are problems where **we don't have access to gradients, or are computationally expensive to compute**
- Here we will briefly describe the finite differences method:

$$x_{t+1} = x_t - \eta \frac{f(x_t + \mu_t u) - f(x_t)}{\mu_t} \cdot u$$

(we have access to function evaluations)

- Based on the approximation of the gradient:

$$f'(x) \approx \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

Application: Adversarial examples in NN training

(A quick description)

Application: Adversarial examples in NN training

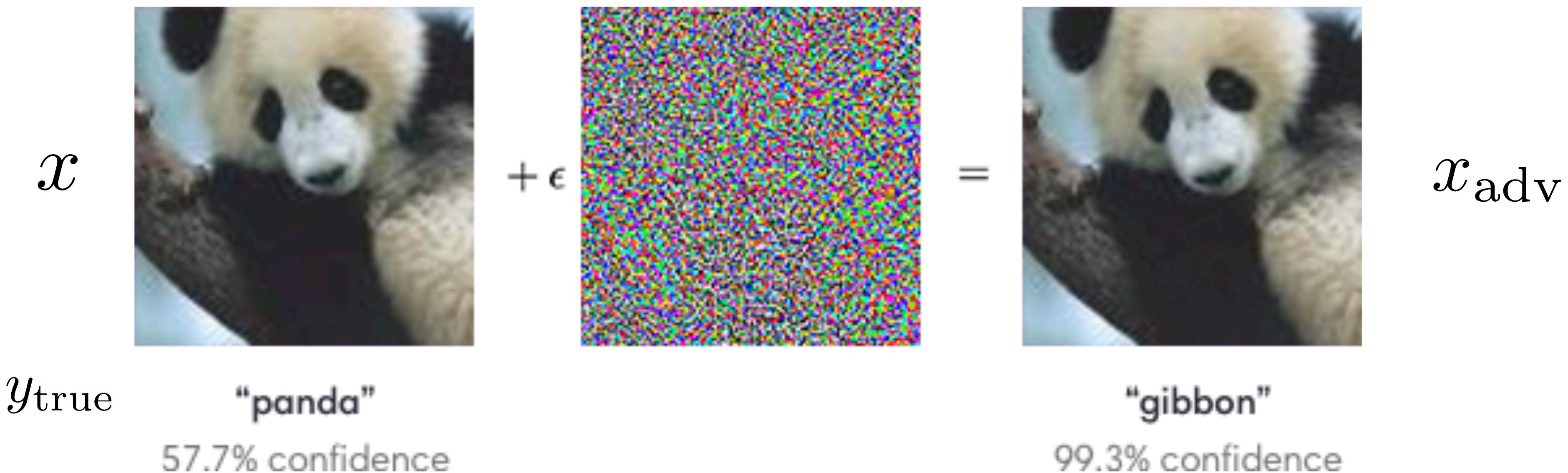
(A quick description)

- The idea of adversarial examples: small perturbations lead to misclassification

Application: Adversarial examples in NN training

(A quick description)

- The idea of adversarial examples: small perturbations lead to misclassification



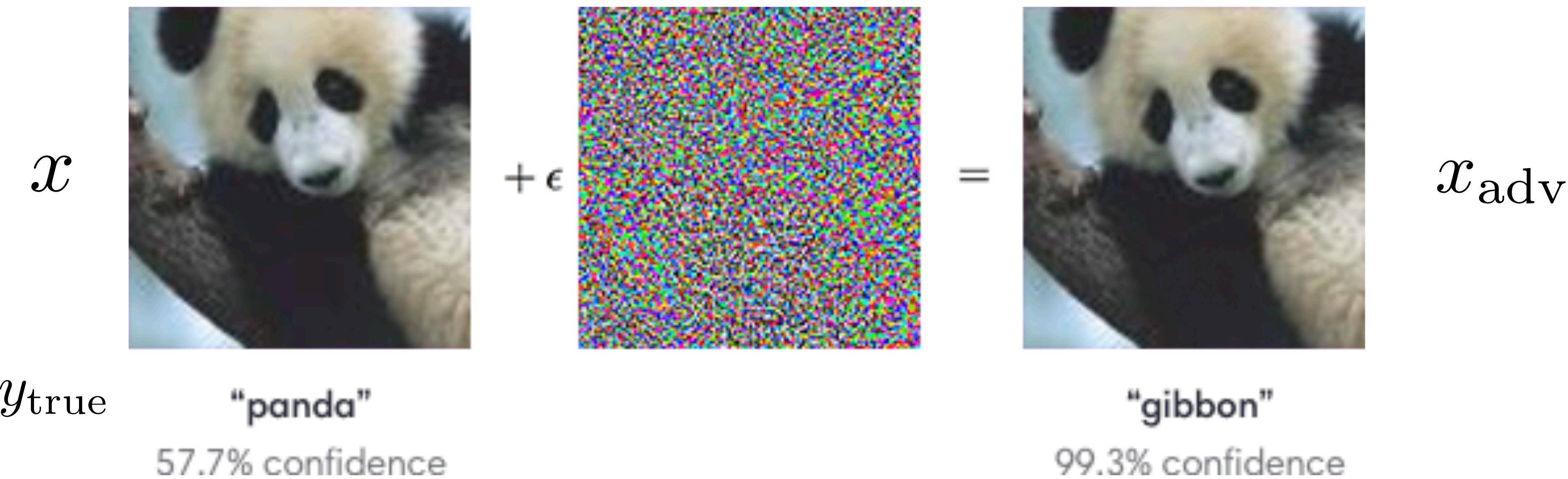
$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla f(x, y_{\text{true}}))$$

(The objective represents a complex model like a neural network)

Application: Adversarial examples in NN training

(A quick description)

- The idea of adversarial examples: small perturbations lead to misclassification



We are looking into directions that move away from the minimum

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla f(x, y_{\text{true}}))$$

(The objective represents a complex model like a neural network)

Application: Adversarial examples in NN training

(A quick description)

- This problematic behavior created a series of **defenses to adversarial attacks**

Application: Adversarial examples in NN training

(A quick description)

- This problematic behavior created a series of **defenses to adversarial attacks**
- A large class of such defenses is based on the idea of “**obfuscating**” **the gradient information**. E.g., in this attack

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla f(x, y_{\text{true}}))$$

is disturbed or nullified (through transformations that disturb the back-Propagation (=gradient calculation))

Application: Adversarial examples in NN training

(A quick description)

- This problematic behavior created a series of **defenses to adversarial attacks**
- A large class of such defenses is based on the idea of “**obfuscating**” **the gradient information**. E.g., in this attack

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla f(x, y_{\text{true}}))$$

is disturbed or nullified (through transformations that disturb the back-Propagation (=gradient calculation))

- However, **the forward operation remains intact**: this means that the function evaluations are normally computed

Application: Adversarial examples in NN training

(A quick description)

- SPSA attack (Simultaneous Perturbation Stochastic Approximation)

Algorithm 1 SPSA adversarial attack

Input: function to minimize f , initial image $x_0 \in \mathbb{R}^D$, perturbation size δ , step size $\alpha > 0$, batch size n
for $t = 0$ **to** $T - 1$ **do**

 Sample $v_1, \dots, v_n \sim \{1, -1\}^D$

 Define $v_i^{-1} = [v_{i,1}^{-1}, \dots, v_{i,D}^{-1}]$

 Calculate $g_i = (f(x_t + \delta v_i) - f(x_t - \delta v_i))v_i^{-1}/(2\delta)$

 Set $x'_t = x_t - \alpha(1/n) \sum_{i=1}^n g_i$

 Project $x_{t+1} = \arg \min_{x \in N_\epsilon(x_0)} \|x'_t - x\|$

end for

From previous lecture: lower bounds

- For objectives with Lipschitz continuous gradients:

$$f(x_t) - f(x^*) \geq \frac{3L\|x_0 - x^*\|_2^2}{32(t+1)^2}$$

(Under these assumptions, and using only gradients, we cannot achieve better than $O\left(\frac{1}{t^2}\right)$)

- In addition, for objectives that are strongly convex:

$$\|x_t - x^*\|_2^2 \geq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^{2t} \|x_0 - x^*\|_2^2$$

$$\kappa := \frac{L}{\mu}$$

(The case we described has near optimal exponent, but does not involve the square root of κ)

From previous lecture: lower bounds

- For objectives with Lipschitz continuous gradients:

$$f(x_t) - f(x^*) \geq \frac{3L\|x_0 - x^*\|_2^2}{32(t+1)^2}$$

(Under these assumptions, and using only gradients, we cannot achieve better than $O\left(\frac{1}{t^2}\right)$)

- In addition, for objectives that are strongly convex:

$$\|x_t - x^*\|_2^2 \geq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^{2t} \|x_0 - x^*\|_2^2$$

$$\kappa := \frac{L}{\mu}$$

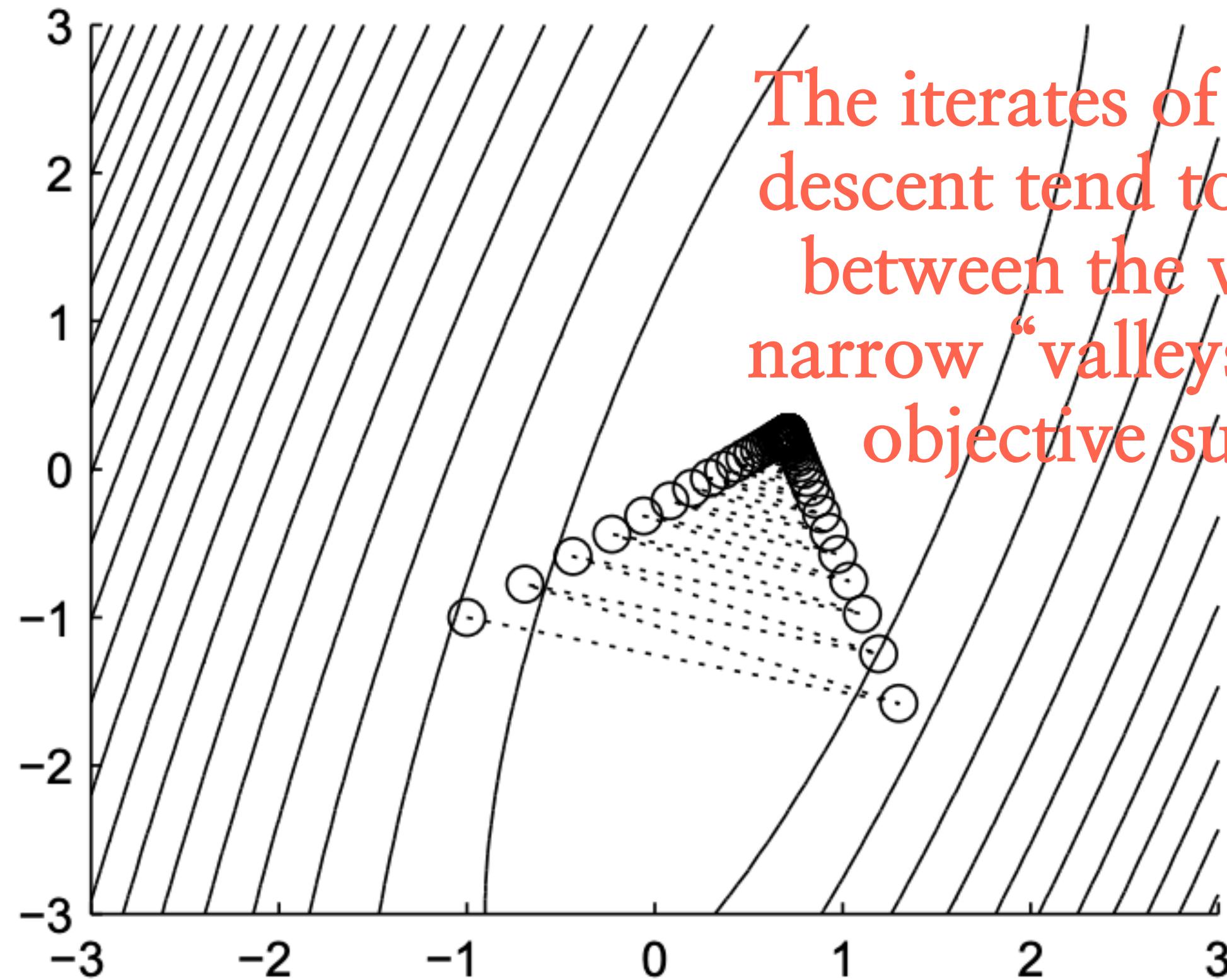
(The case we described has near optimal exponent, but does not involve the square root of κ)

So far, we are “cheating”: use/approximate of higher-order information

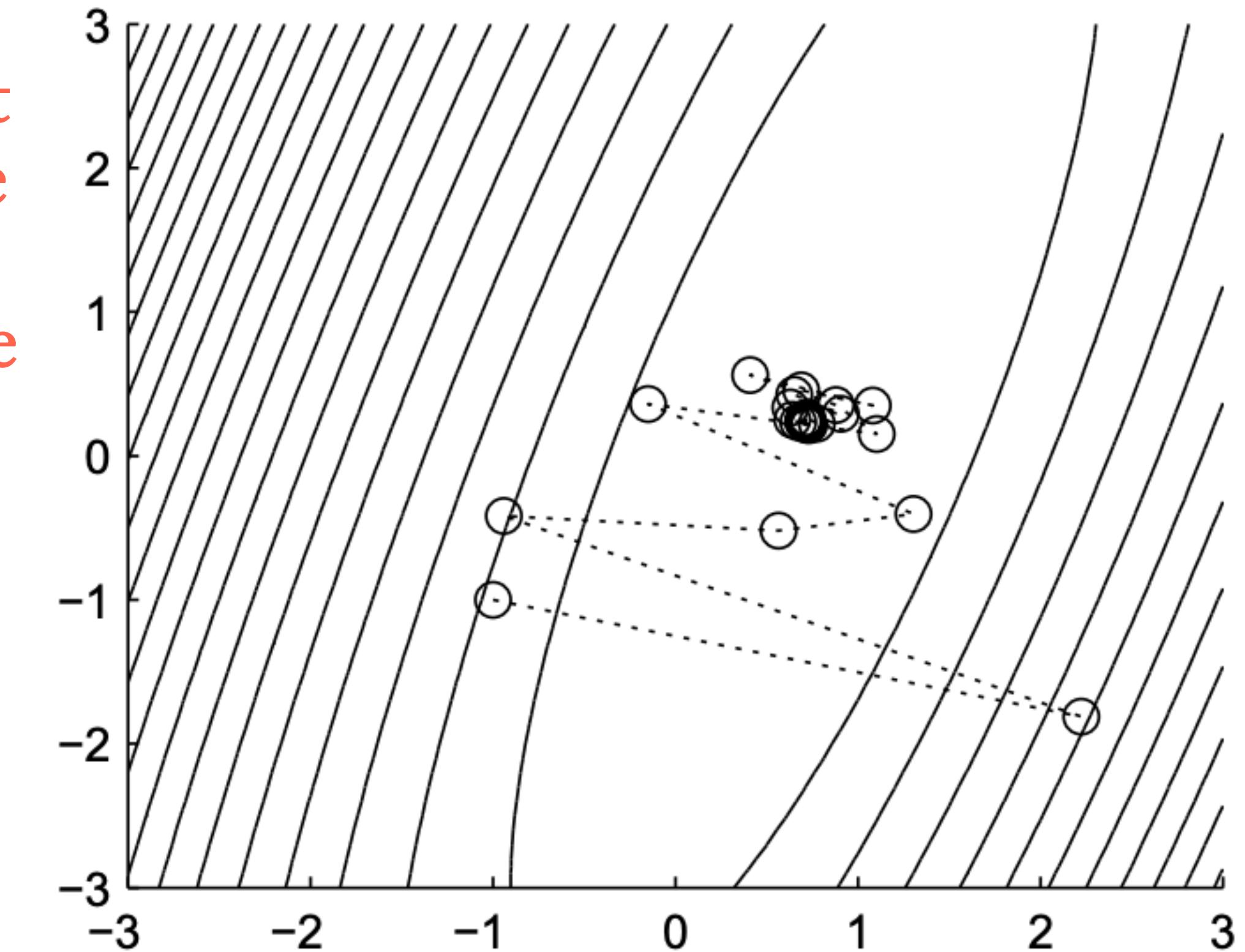
“Can we accelerate having as our basis the standard gradient descent?”

Acceleration #1: Momentum acceleration

- Heavy ball method

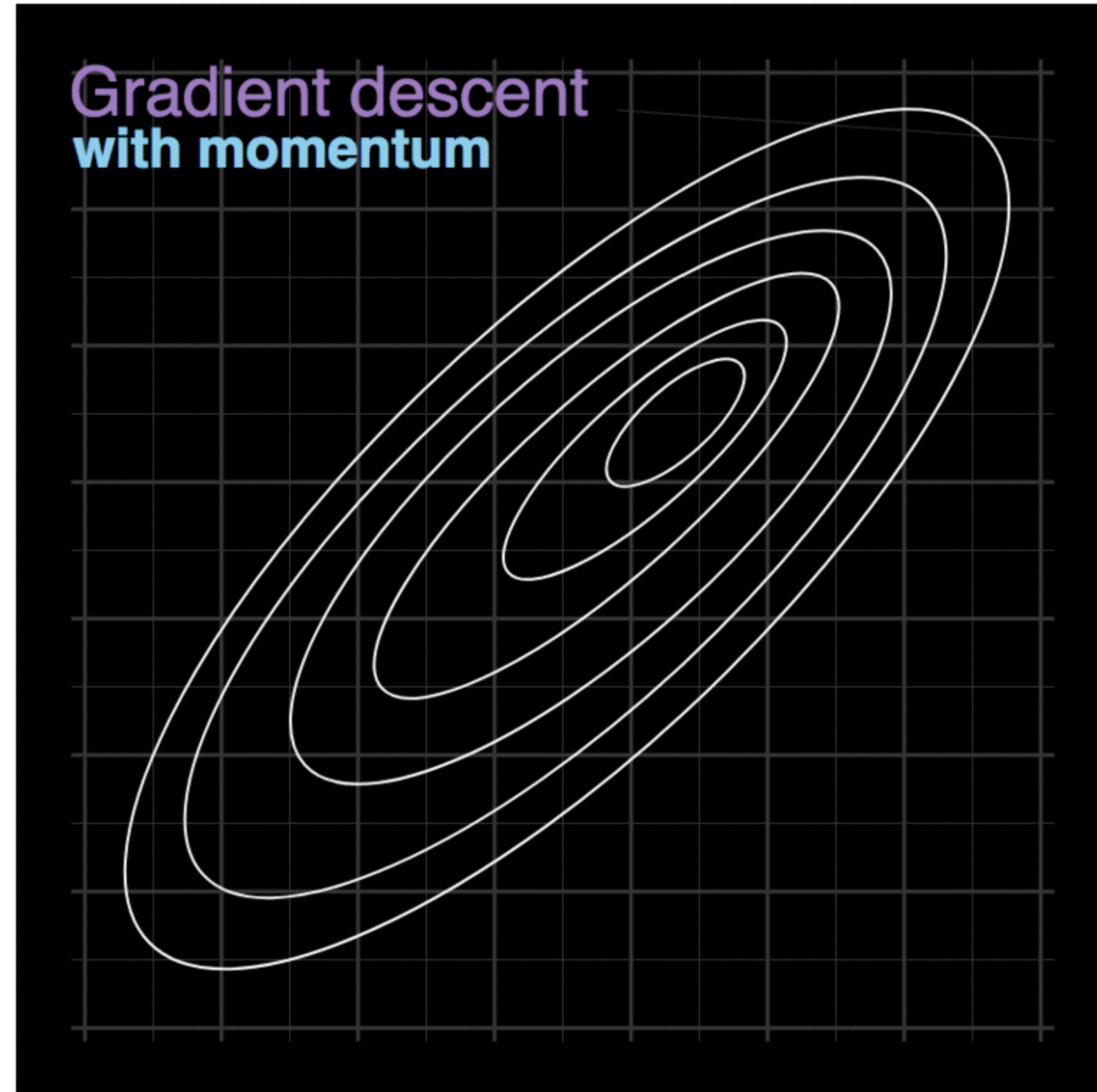


Gradient descent



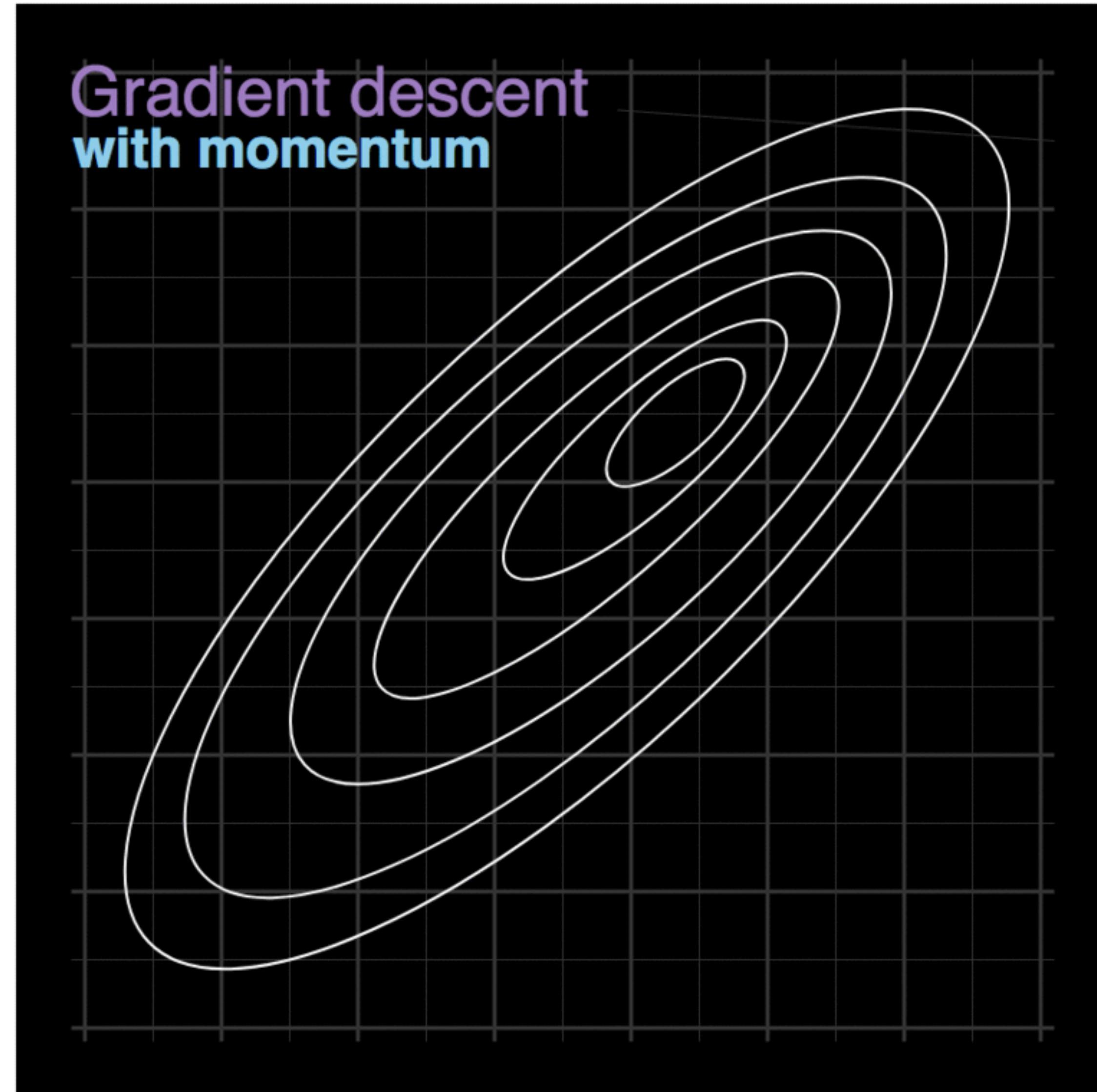
Extrapolating previous directions

Acceleration #1: Momentum acceleration



Tribute to
Piotr Skalski

Acceleration #1: Momentum acceleration



Tribute to
Piotr Skalski

Acceleration #1: Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Acceleration #1: Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$



Standard gradient step

Acceleration #1: Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$



Standard gradient step



Momentum step

Acceleration #1: Momentum acceleration

- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$



Standard gradient step



Momentum step

x_{t-1}

Acceleration #1: Momentum acceleration

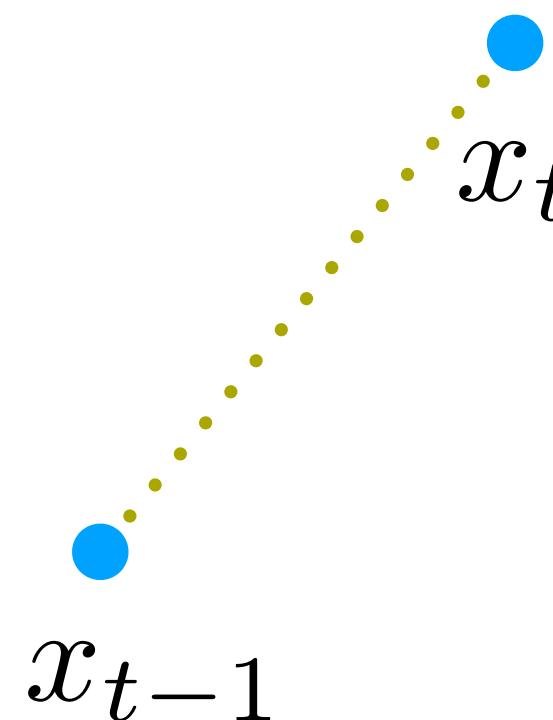
- Heavy ball method

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$



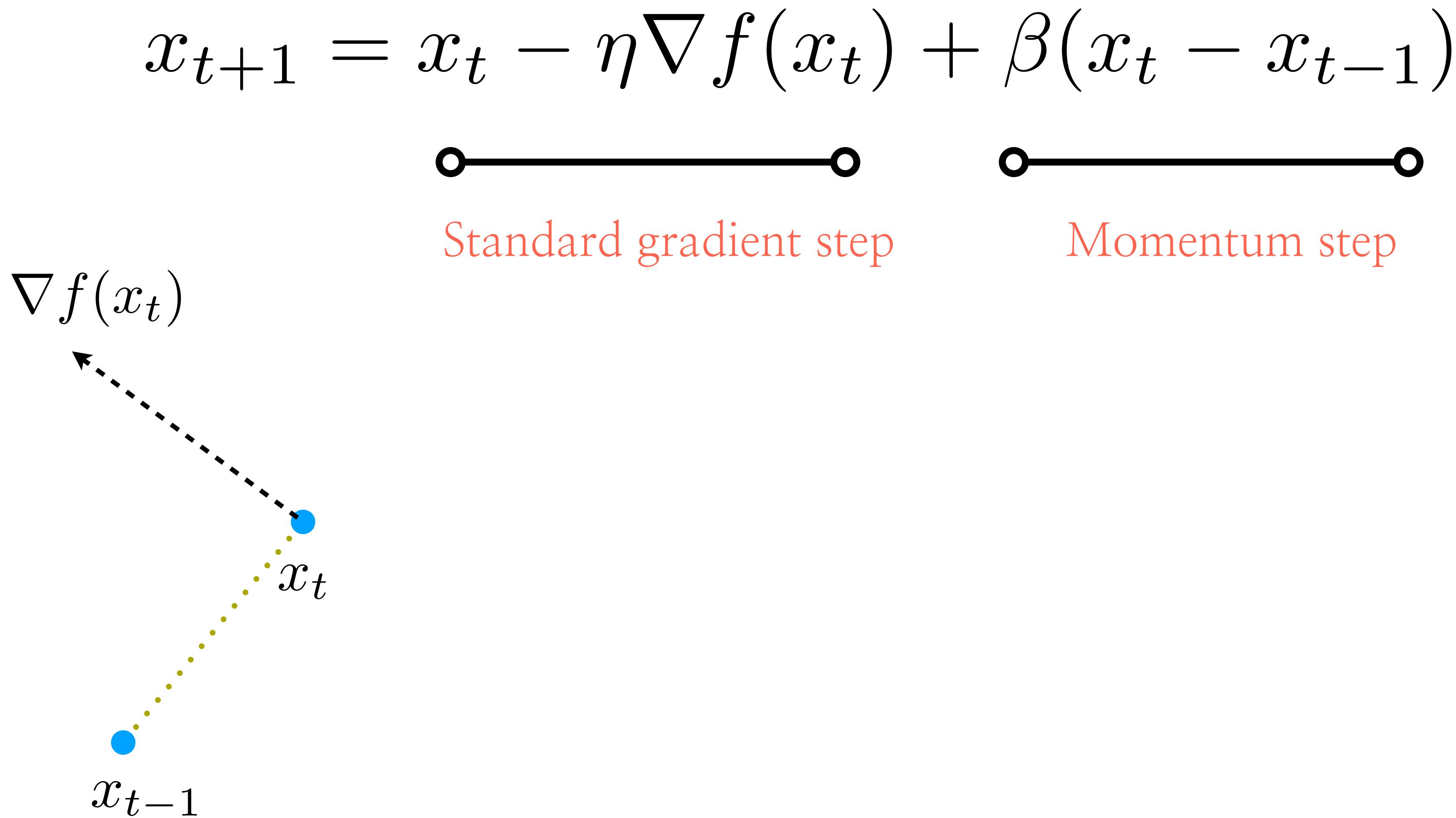
Standard gradient step

Momentum step



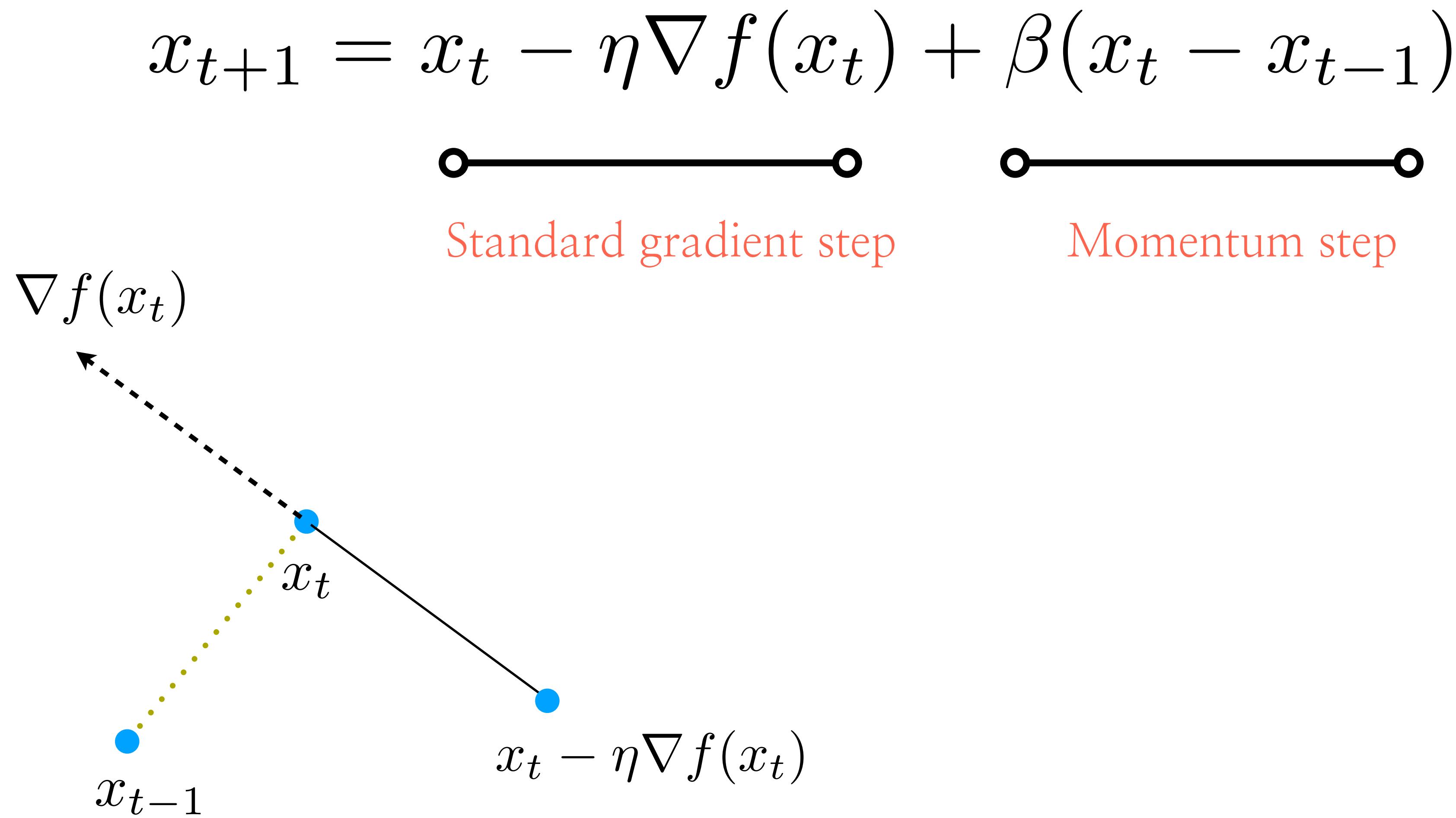
Acceleration #1: Momentum acceleration

- Heavy ball method



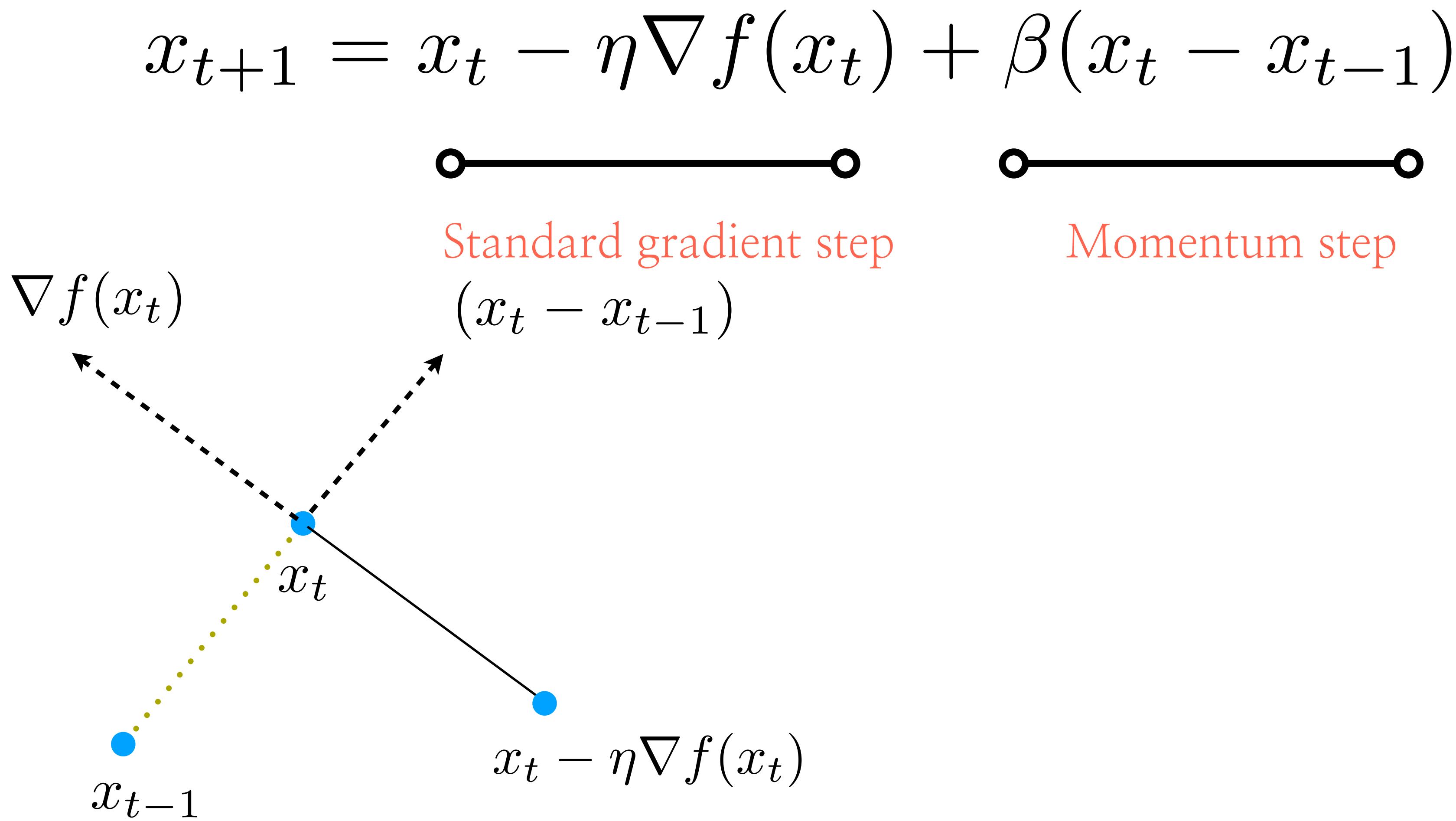
Acceleration #1: Momentum acceleration

- Heavy ball method



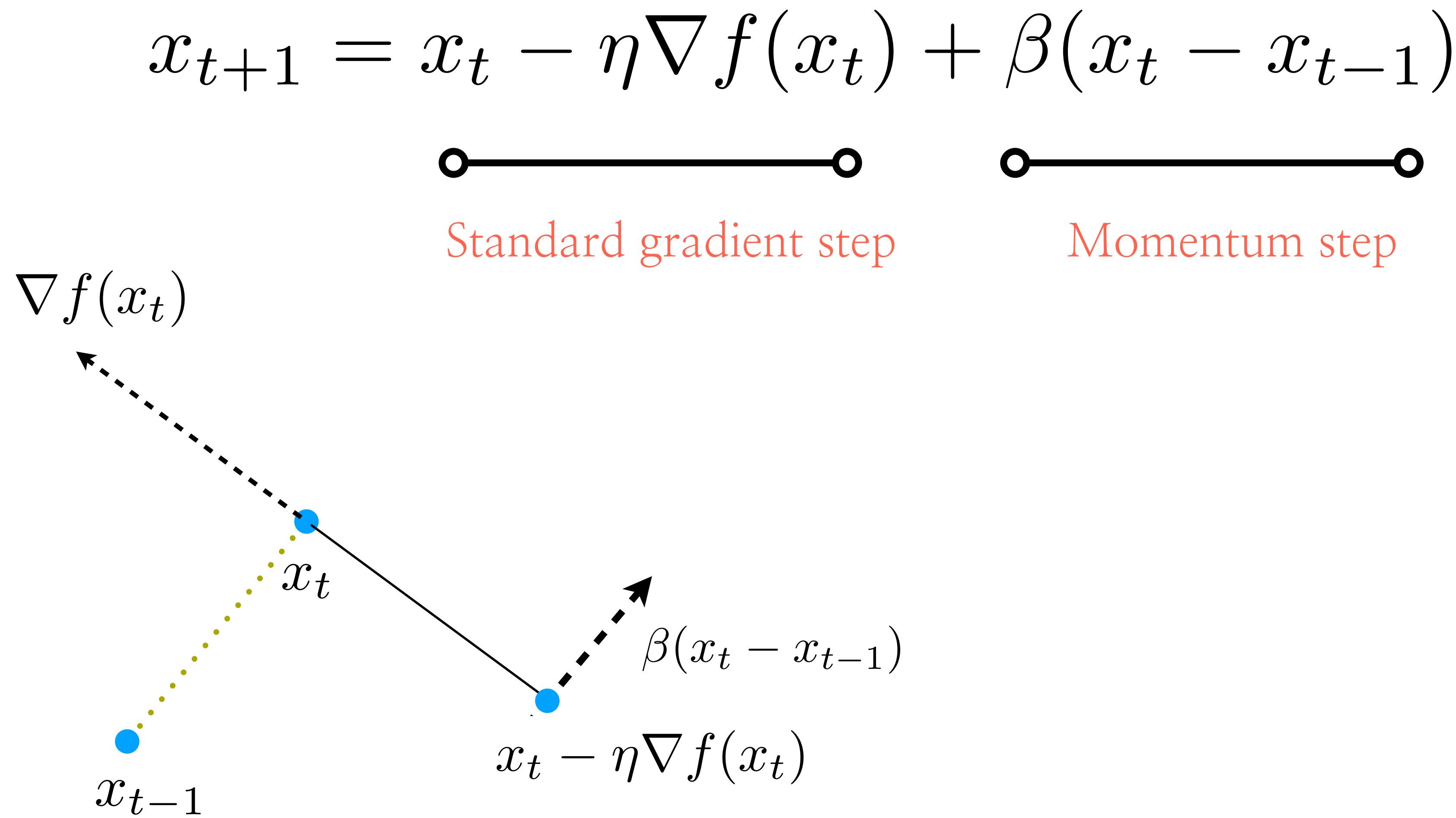
Acceleration #1: Momentum acceleration

- Heavy ball method



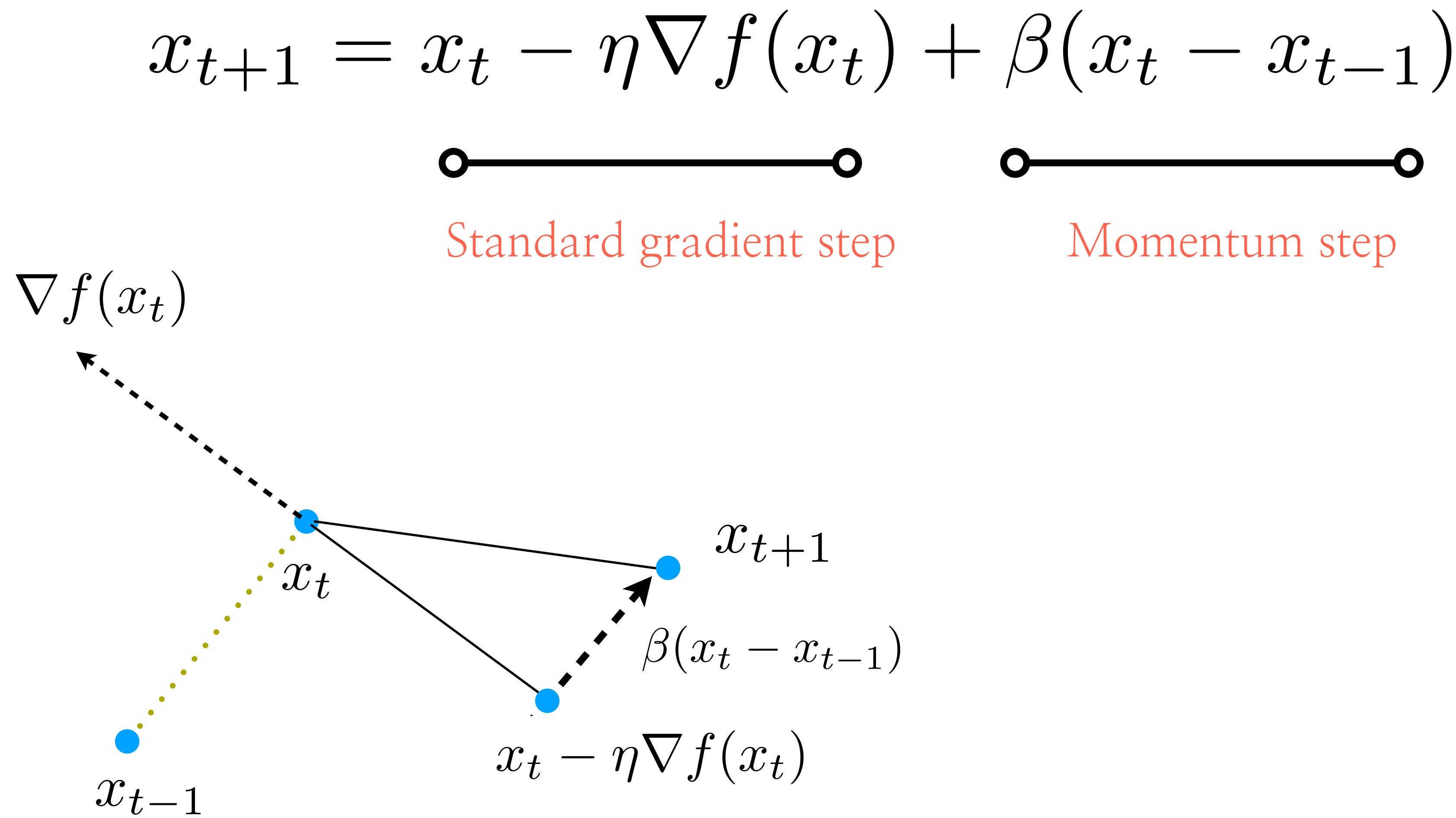
Acceleration #1: Momentum acceleration

- Heavy ball method



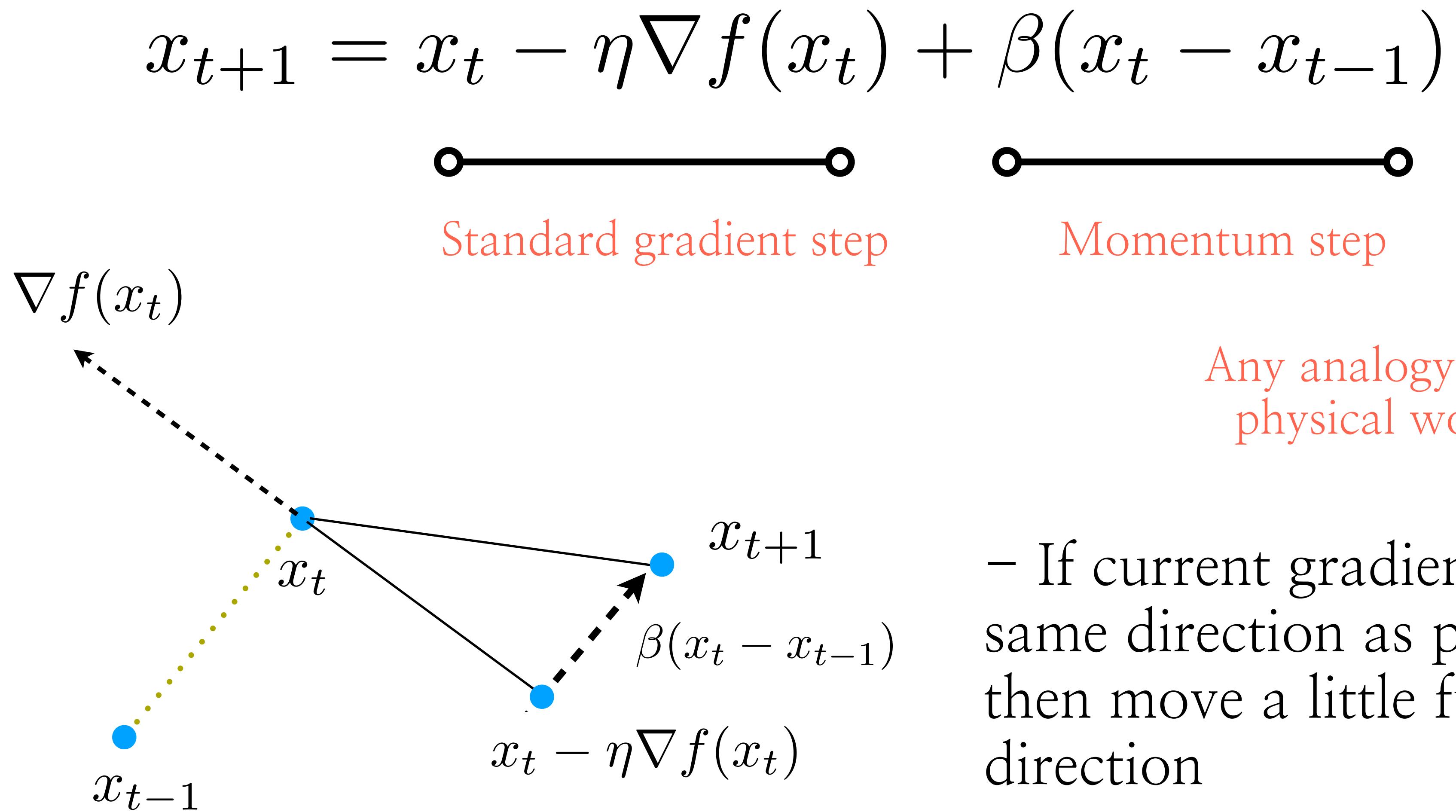
Acceleration #1: Momentum acceleration

- Heavy ball method



Acceleration #1: Momentum acceleration

- Heavy ball method



- If current gradient step is in same direction as previous step, then move a little further in that direction

Guarantees of Heavy Ball method

$$\min_{x \in \mathbb{R}^p} f(x)$$

“Assume the objective is has Lipschitz continuous gradients, and it is strongly convex. Then:

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

for $\eta = \frac{4}{\sqrt{L} + \sqrt{\mu}}$ and $\beta = \max\{|1 - \sqrt{\eta\mu}|, |1 - \sqrt{\eta L}|\}^2$

converges linearly according to:

$$\|x_{t+1} - x^*\|_2 \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^t \|x_0 - x^*\|_2 \quad “$$

Guarantees of Heavy Ball method

Whiteboard

Guarantees of Heavy Ball method

- It achieves the lower bound for strongly convex cases!

$$\|x_t - x^*\|_2^2 \geq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^{2t} \|x_0 - x^*\|_2^2 \quad \kappa := \frac{L}{\mu}$$

Guarantees of Heavy Ball method

- It achieves the lower bound for strongly convex cases!

$$\|x_t - x^*\|_2^2 \geq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^{2t} \|x_0 - x^*\|_2^2 \quad \kappa := \frac{L}{\mu}$$

- In comparison with simple gradient descent:

$$O\left(\kappa \log \frac{1}{\varepsilon}\right) \quad \text{vs} \quad O\left(\sqrt{\kappa} \log \frac{1}{\varepsilon}\right)$$

Performance of Heavy Ball method

Demo

Acceleration #1: Momentum acceleration

– Nesterov's work: a collection of acceleration methods

Constant Step Scheme, I

0. Choose $x_0 \in R^n$ and $\gamma_0 > 0$. Set $v_0 = x_0$.
1. k th iteration ($k \geq 0$).
 - a). Compute $\alpha_k \in (0, 1)$ from the equation

$$L\alpha_k^2 = (1 - \alpha_k)\gamma_k + \alpha_k\mu.$$

Set $\gamma_{k+1} = (1 - \alpha_k)\gamma_k + \alpha_k\mu$.

b). Choose $y_k = \frac{\alpha_k\gamma_k v_k + \gamma_{k+1}x_k}{\gamma_k + \alpha_k\mu}$.

Compute $f(y_k)$ and $f'(y_k)$.

c). Set $x_{k+1} = y_k - \frac{1}{L}f'(y_k)$ and

$$v_{k+1} = \frac{1}{\gamma_{k+1}}[(1 - \alpha_k)\gamma_k v_k + \alpha_k\mu y_k - \alpha_k f'(y_k)].$$

Constant Step Scheme, II

0. Choose $x_0 \in R^n$ and $\alpha_0 \in (0, 1)$.
Set $y_0 = x_0$ and $q = \frac{\mu}{L}$.
1. k th iteration ($k \geq 0$).
 - a). Compute $f(y_k)$ and $f'(y_k)$. Set

$$x_{k+1} = y_k - \frac{1}{L}f'(y_k).$$

- b). Compute $\alpha_{k+1} \in (0, 1)$ from equation

$$\alpha_{k+1}^2 = (1 - \alpha_{k+1})\alpha_k^2 + q\alpha_k +$$

and set $\beta_k = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2 + \alpha_{k+1}}$,

$$y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k)$$

Constant step scheme, III

0. Choose $y_0 = x_0 \in R^n$.

1. k th iteration ($k \geq 0$).
 - $x_{k+1} = y_k - \frac{1}{L}f'(y_k),$
 - $y_{k+1} = x_{k+1} + \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(x_{k+1} - x_k).$

Acceleration #1: Momentum acceleration

- Nesterov's work: a collection of acceleration methods

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

Acceleration #1: Momentum acceleration

- Nesterov's work: a collection of acceleration methods

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$



$$\tilde{x} = x_t - \eta \nabla f(x_t)$$

$$x_{t+1} = \tilde{x} + \beta(x_t - x_{t-1})$$

Acceleration #1: Momentum acceleration

- Nesterov's work: a collection of acceleration methods

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$

$$\begin{array}{c} \downarrow \\ \widetilde{x} = x_t - \eta \nabla f(x_t) \end{array}$$

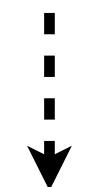
Evaluate gradient at
current point

$$x_{t+1} = \widetilde{x} + \beta(x_t - x_{t-1})$$

Acceleration #1: Momentum acceleration

- Nesterov's work: a collection of acceleration methods

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$



Evaluate gradient at
current point

$$\tilde{x} = x_t - \eta \nabla f(x_t)$$

$$x_{t+1} = \tilde{x} + \beta(x_t - x_{t-1})$$



$$\tilde{x} = x_t - \eta \nabla f(x_t + \beta(x_t - x_{t-1}))$$

$$x_{t+1} = \tilde{x} + \beta(x_t - x_{t-1})$$

What if we evaluate the
gradient at the point we
end up?

Acceleration #1: Momentum acceleration

- Nesterov's work: a collection of acceleration methods

$$x_{t+1} = x_t - \eta \nabla f(x_t) + \beta(x_t - x_{t-1})$$



Evaluate gradient at
current point

$$\tilde{x} = x_t - \eta \nabla f(x_t)$$

$$x_{t+1} = \tilde{x} + \beta(x_t - x_{t-1})$$



What if we evaluate the
gradient at the point we
end up?

Nesterov's acceleration (1/2)

$$\tilde{x} = x_t - \eta \nabla f(x_t + \beta(x_t - x_{t-1}))$$

$$x_{t+1} = \tilde{x} + \beta(x_t - x_{t-1})$$

Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

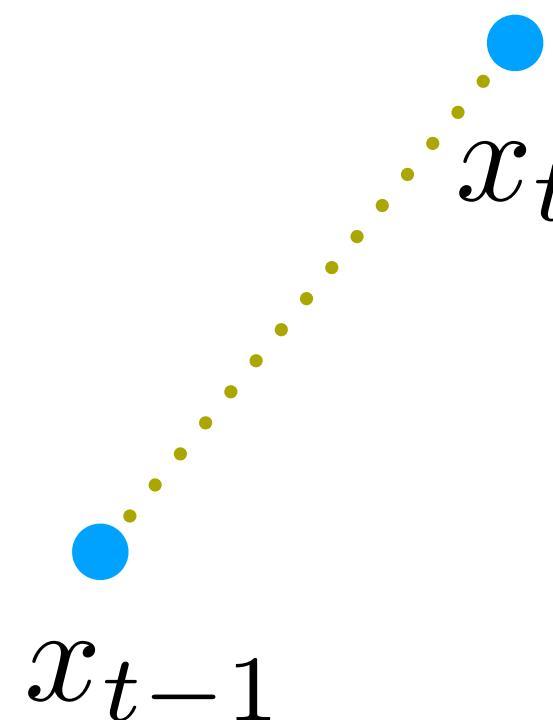

 x_{t-1}

Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

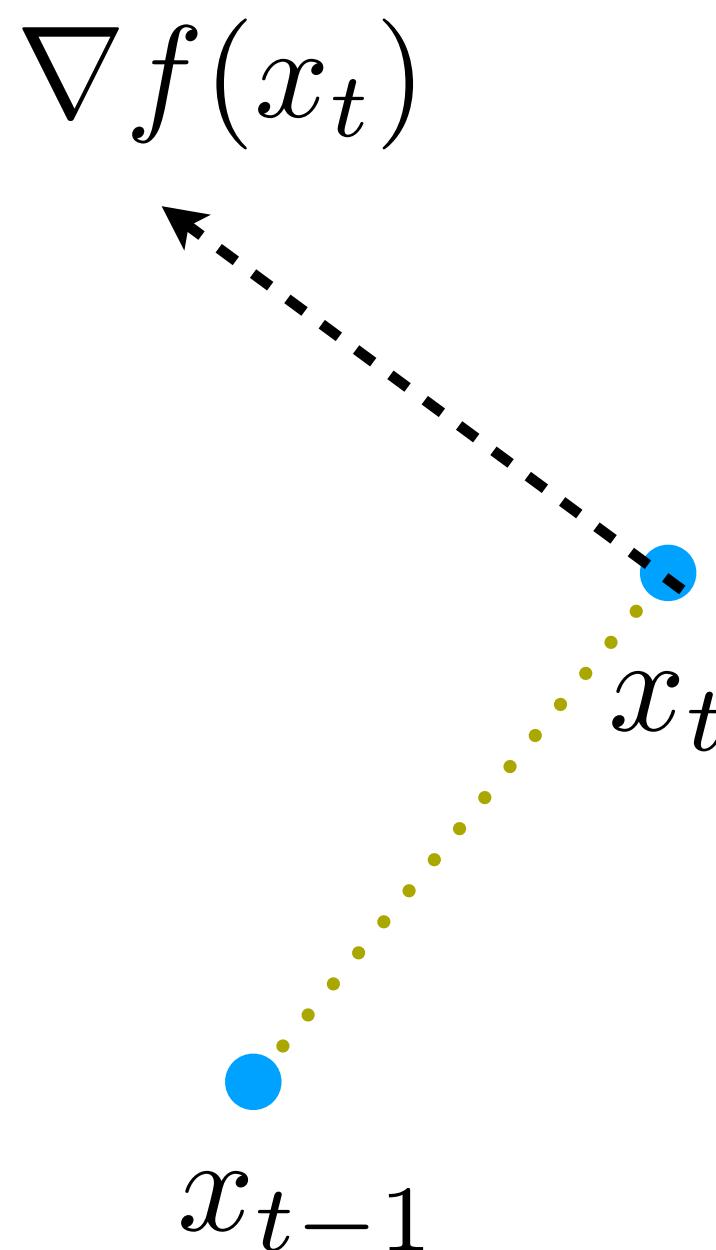


Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

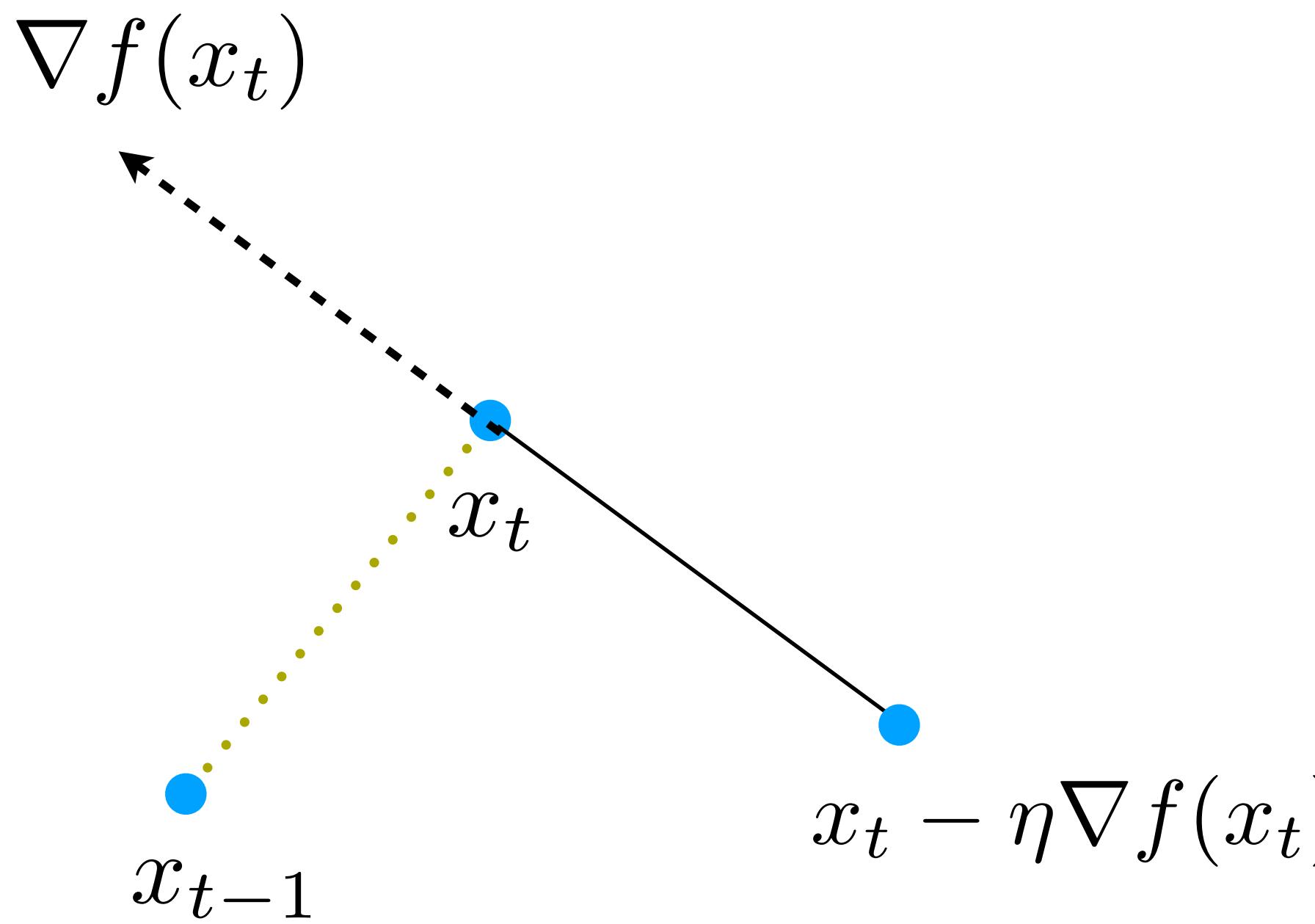


Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

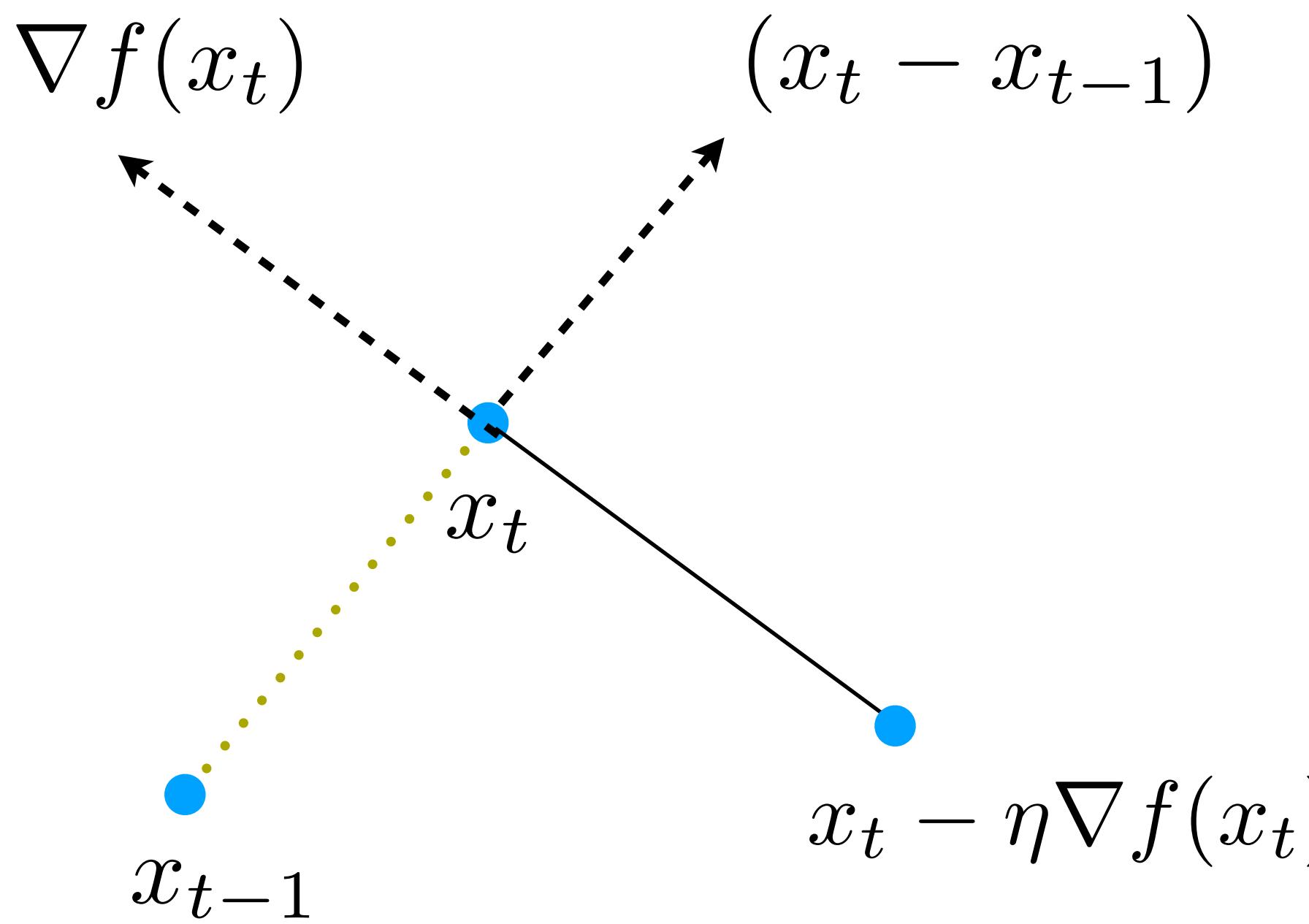


Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

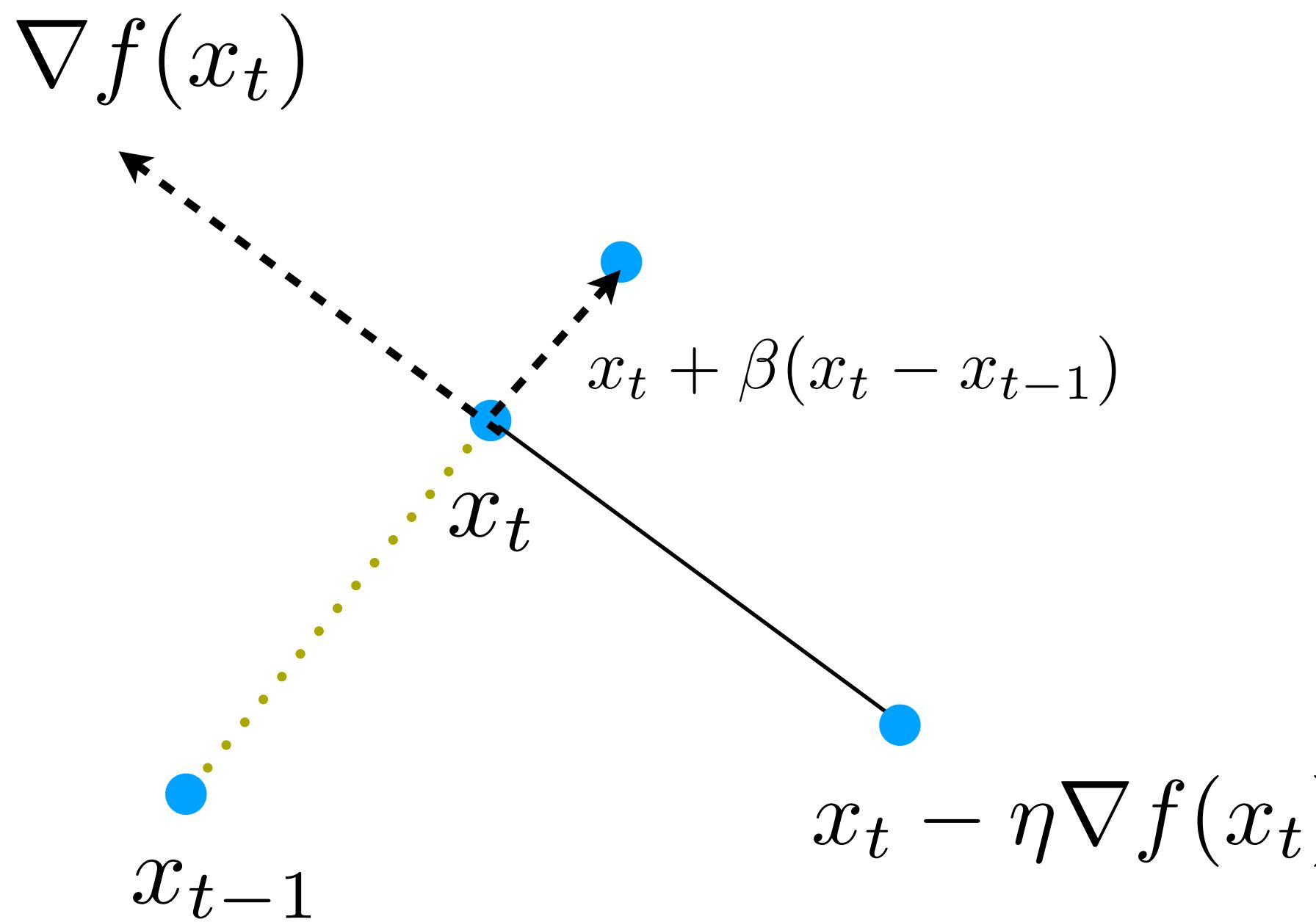


Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

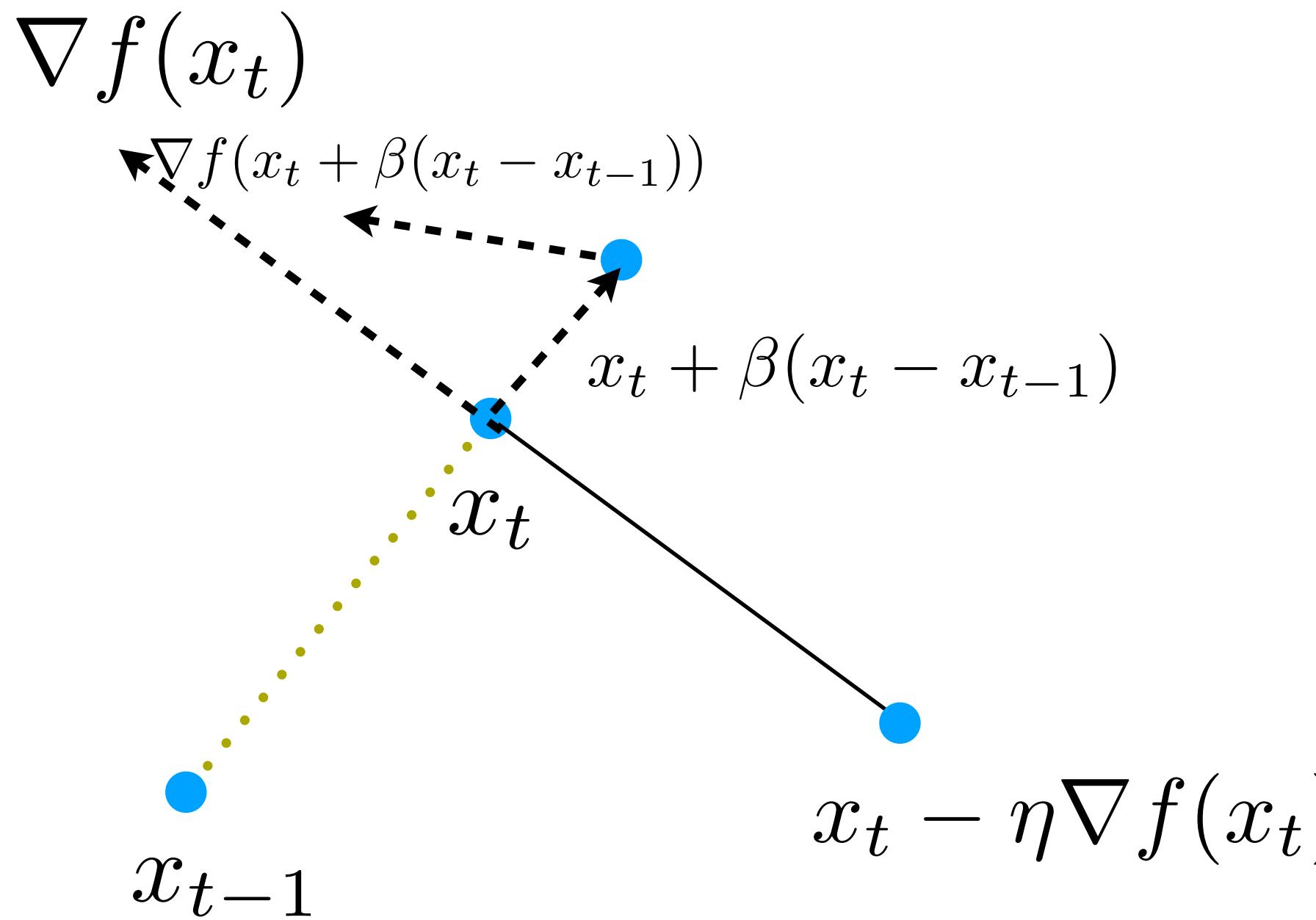


Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

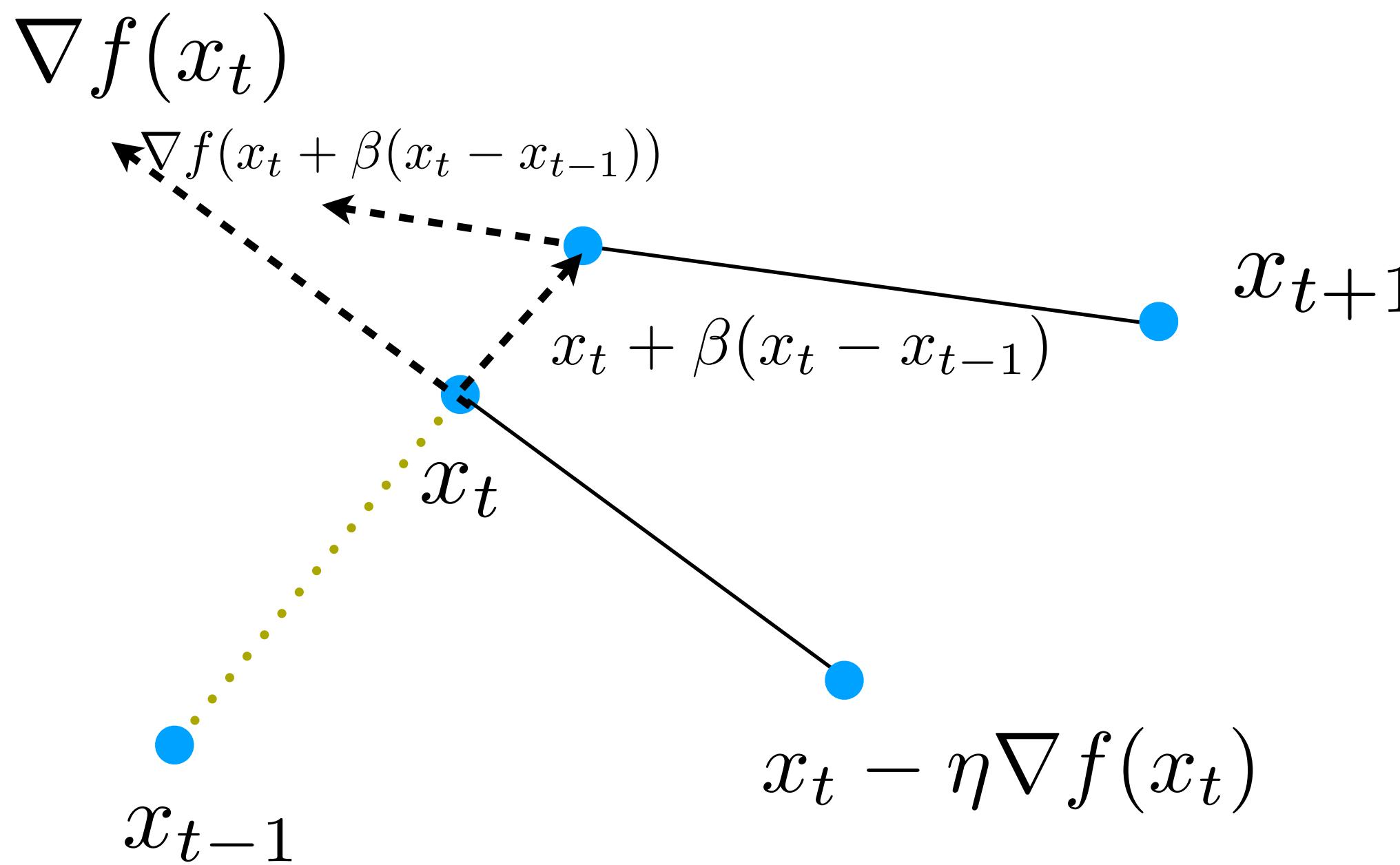


Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

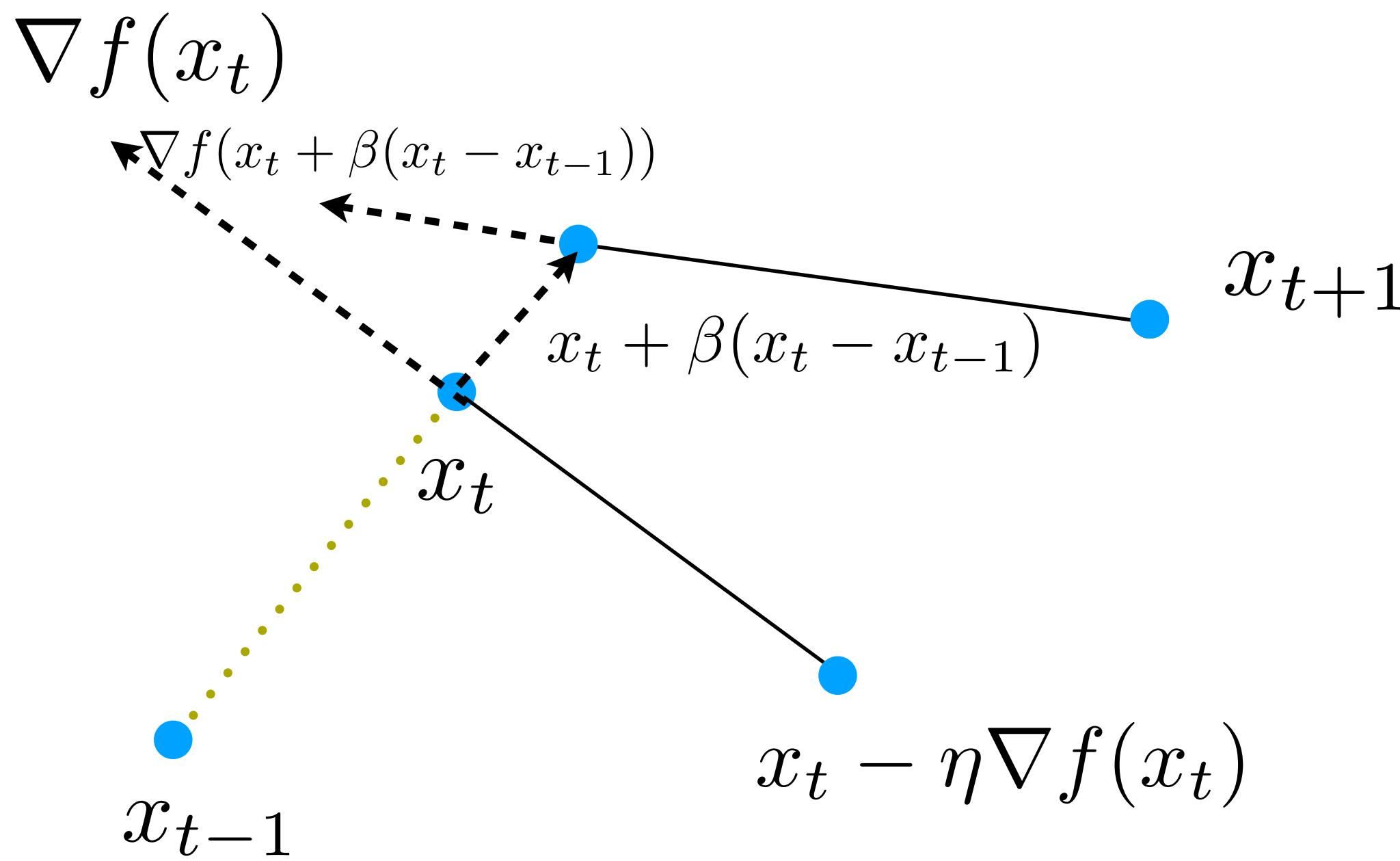


Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



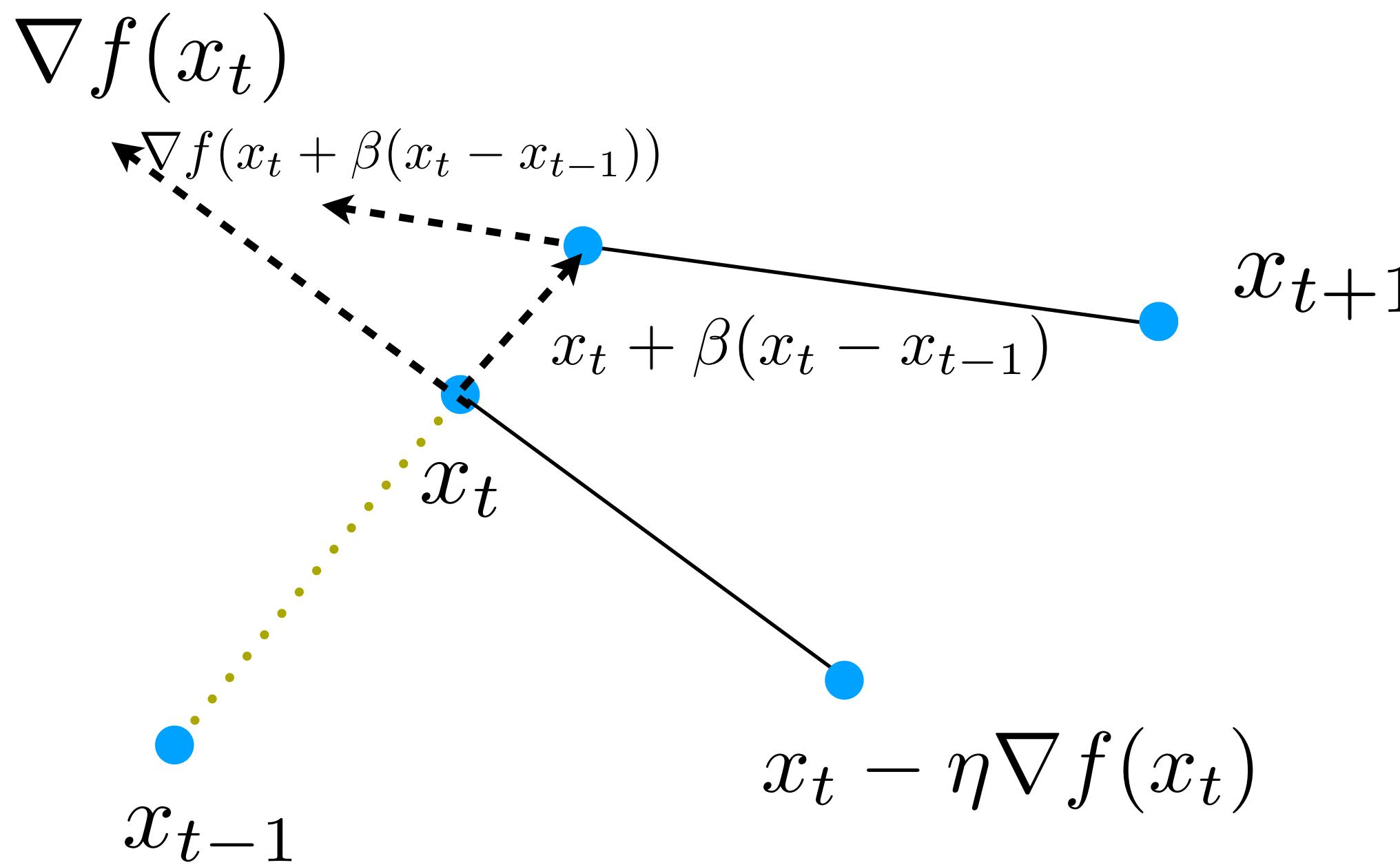
- Main difference: the point that we are calculating the gradient at.

Acceleration #1: Momentum acceleration

- Nesterov's work: most famous version

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



- Main difference: the point that we are calculating the gradient at.
- Heavy ball can fail converging in cases where Nesterov's scheme still succeeds

Acceleration #1: Momentum acceleration

- Nesterov's work: how do we set up the momentum parameter?

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

Acceleration #1: Momentum acceleration

- Nesterov's work: how do we set up the momentum parameter?

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

$$1. \quad \beta = \frac{\theta_t - 1}{\theta_{t+1}} \quad \text{where} \quad \theta_0 = 1, \quad \theta_{t+1} = \frac{1 + \sqrt{1 + 4\theta_t^2}}{2}$$

Acceleration #1: Momentum acceleration

- Nesterov's work: how do we set up the momentum parameter?

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

$$1. \beta = \frac{\theta_t - 1}{\theta_{t+1}} \text{ where } \theta_0 = 1, \quad \theta_{t+1} = \frac{1 + \sqrt{1 + 4\theta_t^2}}{2}$$

$$2. \beta = \frac{t}{t+3}$$

Acceleration #1: Momentum acceleration

- Nesterov's work: how do we set up the momentum parameter?

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

$$1. \beta = \frac{\theta_t - 1}{\theta_{t+1}} \text{ where } \theta_0 = 1, \quad \theta_{t+1} = \frac{1 + \sqrt{1 + 4\theta_t^2}}{2}$$

$$2. \beta = \frac{t}{t+3}$$

$$3. \beta = 0.9$$

Acceleration #1: Momentum acceleration

- Nesterov's work: how do we set up the momentum parameter?

$$x_{t+1} = y_t - \eta \nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$

$$1. \beta = \frac{\theta_t - 1}{\theta_{t+1}} \quad \text{where } \theta_0 = 1, \quad \theta_{t+1} = \frac{1 + \sqrt{1 + 4\theta_t^2}}{2}$$

$$2. \beta = \frac{t}{t+3}$$

$$3. \beta = 0.9$$



One of the mysteries of
optimization..

Performance of Nesterov's acceleration

Demo

Guarantees of Nesterov's acceleration

- Gradient descent in the absence of strong convexity (No theory but willing to provide links for whoever is interested)

$$f(x_t) - f(x^*) \leq \frac{2L\|x_0 - x^*\|_2^2}{t + 4}$$

Guarantees of Nesterov's acceleration

(No theory but willing to provide links for whoever is interested)

- Gradient descent in the absence of strong convexity

$$f(x_t) - f(x^*) \leq \frac{2L\|x_0 - x^*\|_2^2}{t + 4}$$

- Nesterov's acceleration (with momentum similarly set up as in previous slide)

$$f(x_t) - f(x^*) \leq \frac{4L\|x_0 - x^*\|_2^2}{(t + 2)^2}$$

Guarantees of Nesterov's acceleration

(No theory but willing to provide

- Gradient descent in the absence of strong convexity [links for whoever is interested](#)

$$f(x_t) - f(x^*) \leq \frac{2L\|x_0 - x^*\|_2^2}{t + 4}$$

- Nesterov's acceleration (with momentum similarly set up as in previous slide)

$$f(x_t) - f(x^*) \leq \frac{4L\|x_0 - x^*\|_2^2}{(t + 2)^2}$$

- Reminder of lower bounds for Lipschitz continuous gradients:

$$f(x_t) - f(x^*) \geq \frac{3L\|x_0 - x^*\|_2^2}{32(t + 1)^2}$$

Guarantees of Nesterov's acceleration

(No theory but willing to provide
links for whoever is interested)

- Gradient descent in the absence of strong convexity

$$f(x_t) - f(x^*) \leq \frac{2L\|x_0 - x^*\|_2^2}{t + 4}$$

- Nesterov's acceleration (with momentum similarly set up as in previous slide)

$$f(x_t) - f(x^*) \leq \frac{4L\|x_0 - x^*\|_2^2}{(t + 2)^2}$$

Optimal!

- Reminder of lower bounds for Lipschitz continuous gradients:

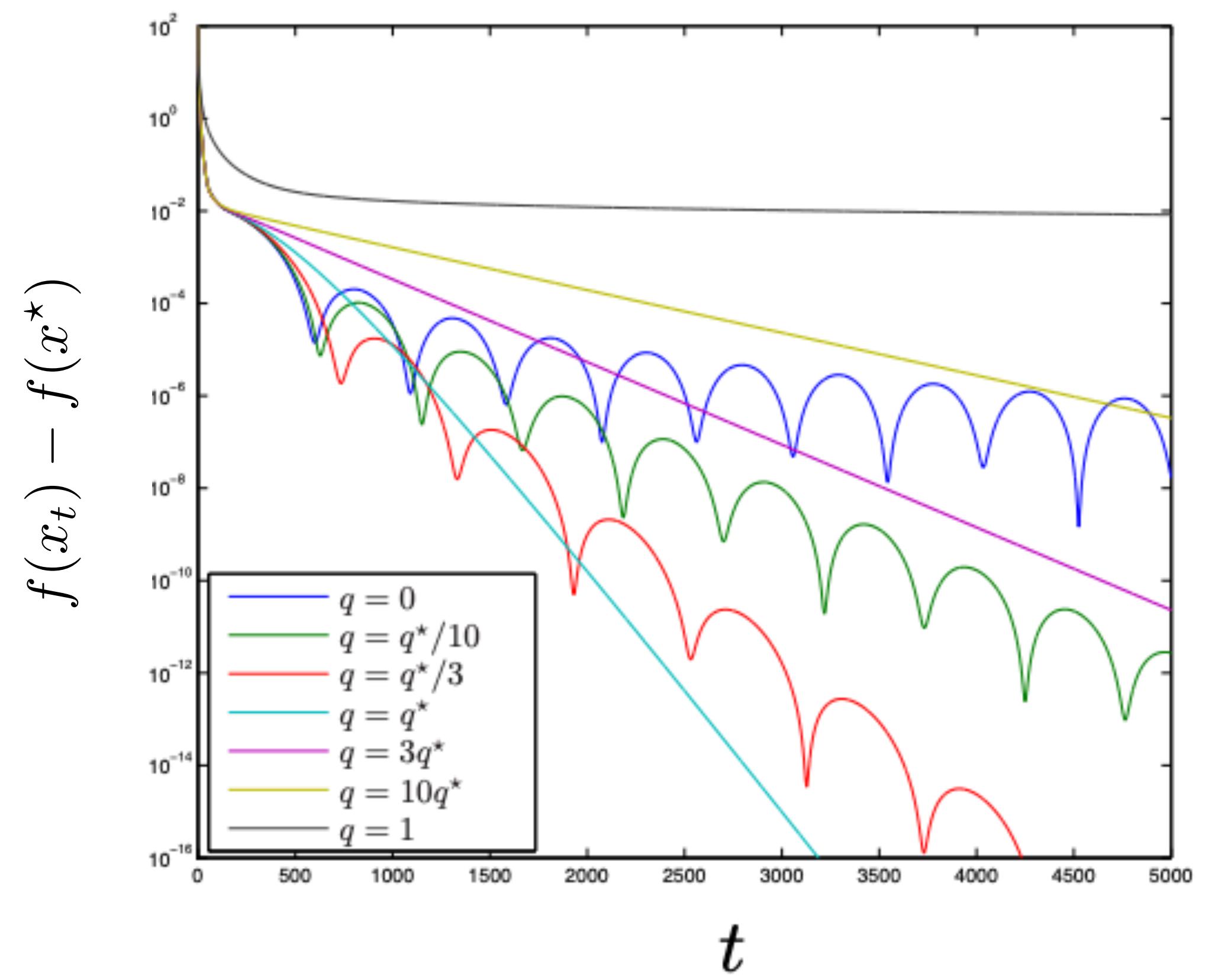
$$f(x_t) - f(x^*) \geq \frac{3L\|x_0 - x^*\|_2^2}{32(t + 1)^2}$$

Notes on Nesterov's acceleration

- The original paper of 1983 does not converge linearly for strongly convex functions, but there is a fix to this

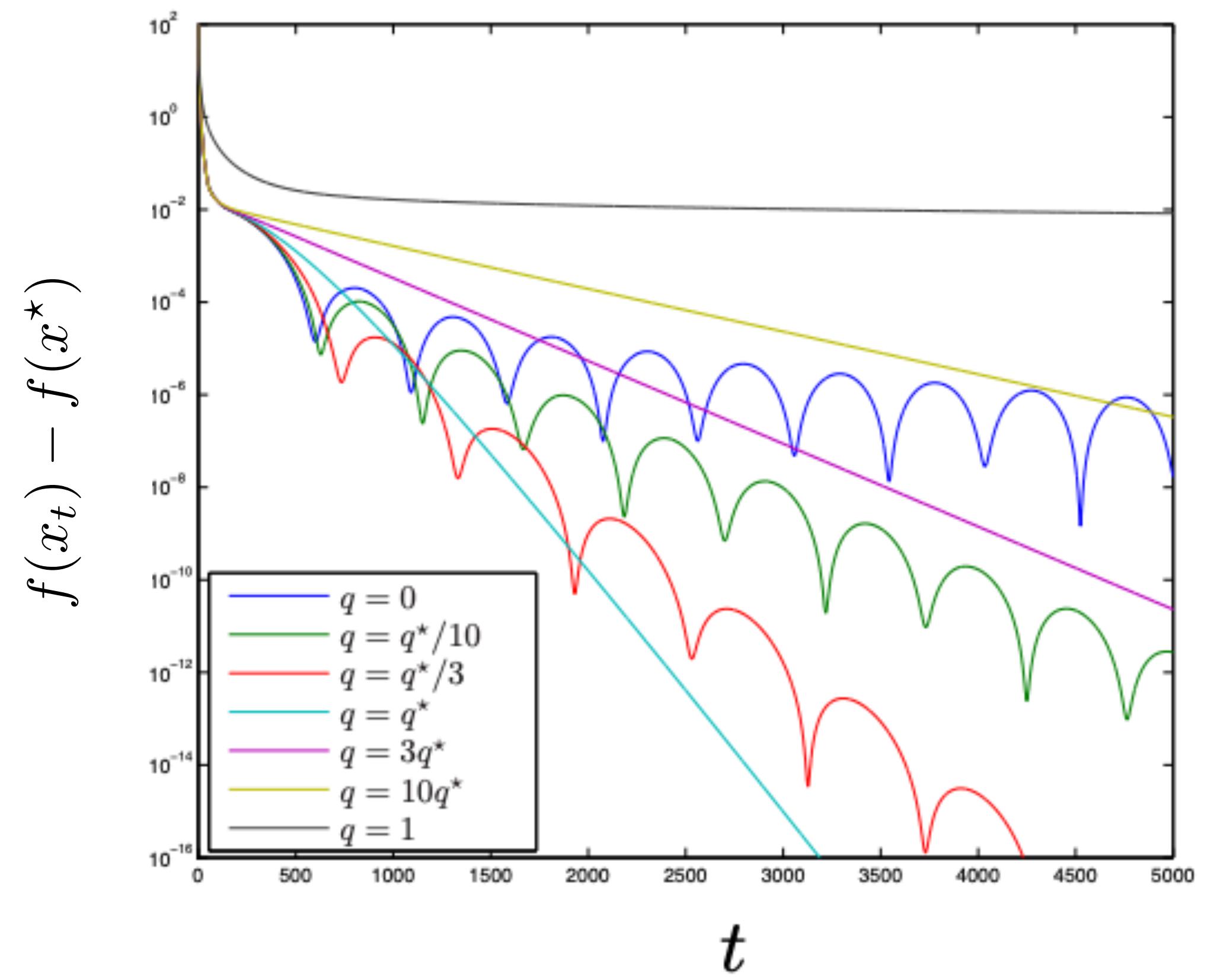
Notes on Nesterov's acceleration

- The original paper of 1983 does not converge linearly for strongly convex functions, but there is a fix to this
- It is a common observation to see ripples



Notes on Nesterov's acceleration

- The original paper of 1983 does not converge linearly for strongly convex functions, but there is a fix to this
- It is a common observation to see ripples
- There are heuristics for resetting the momentum term to zero that improves the convergence rate.
- Often used even in cases where it is not guaranteed to work: deep learning



Acceleration #2: Cut-off complexity per iteration

- Common situation in machine learning/signal processing

$$f(x) = \frac{1}{n} \sum_{i=1}^n f_i(x) \quad (\text{Empirical risk minimization})$$

where each $f_i(x)$ depends on, let's say, different part of input data.

Acceleration #2: Cut-off complexity per iteration

- Common situation in machine learning/signal processing

$$f(x) = \frac{1}{n} \sum_{i=1}^n f_i(x) \quad (\text{Empirical risk minimization})$$

where each $f_i(x)$ depends on, let's say, different part of input data.

- Examples:
 - Least squares: $f_i(x) = \frac{1}{2}(y_i - \alpha_i^\top x)^2$
 - Logistic regression: $f_i(x) = \log(1 + \exp(-y_i \alpha_i^\top x))$
- Dimensions to worry about: $x \in \mathbb{R}^p$, number of samples n

Acceleration #2: Cut-off complexity per iteration

- Stochastic gradient descent (SGD)

$$x_{t+1} = x_t - \eta_t \nabla f(x_t) \quad \longrightarrow \quad x_{t+1} = x_t - \eta_t \nabla f_{i_t}(x_t), \quad i_t \in [n]$$

where per iteration we select randomly $i_t \in [n]$.

Acceleration #2: Cut-off complexity per iteration

- Stochastic gradient descent (SGD)

$$x_{t+1} = x_t - \eta_t \nabla f(x_t) \quad \longrightarrow \quad x_{t+1} = x_t - \eta_t \nabla f_{i_t}(x_t), \quad i_t \in [n]$$

where per iteration we select randomly $i_t \in [n]$.

- “Why do we want to do this?”

What is the complexity of
computing full gradient?

$$O(np)$$

(Assume least-squares objective)

What is the complexity of
computing a single gradient?

$$O(p)$$

Acceleration #2: Cut-off complexity per iteration

- Stochastic gradient descent (SGD)

$$x_{t+1} = x_t - \eta_t \nabla f(x_t) \quad \longrightarrow \quad x_{t+1} = x_t - \eta_t \nabla f_{i_t}(x_t), \quad i_t \in [n]$$

where per iteration we select randomly $i_t \in [n]$.

- “Why do we want to do this?”

What is the complexity of
computing full gradient?

$$O(np)$$

(Assume least-squares objective)

What is the complexity of
computing a single gradient?

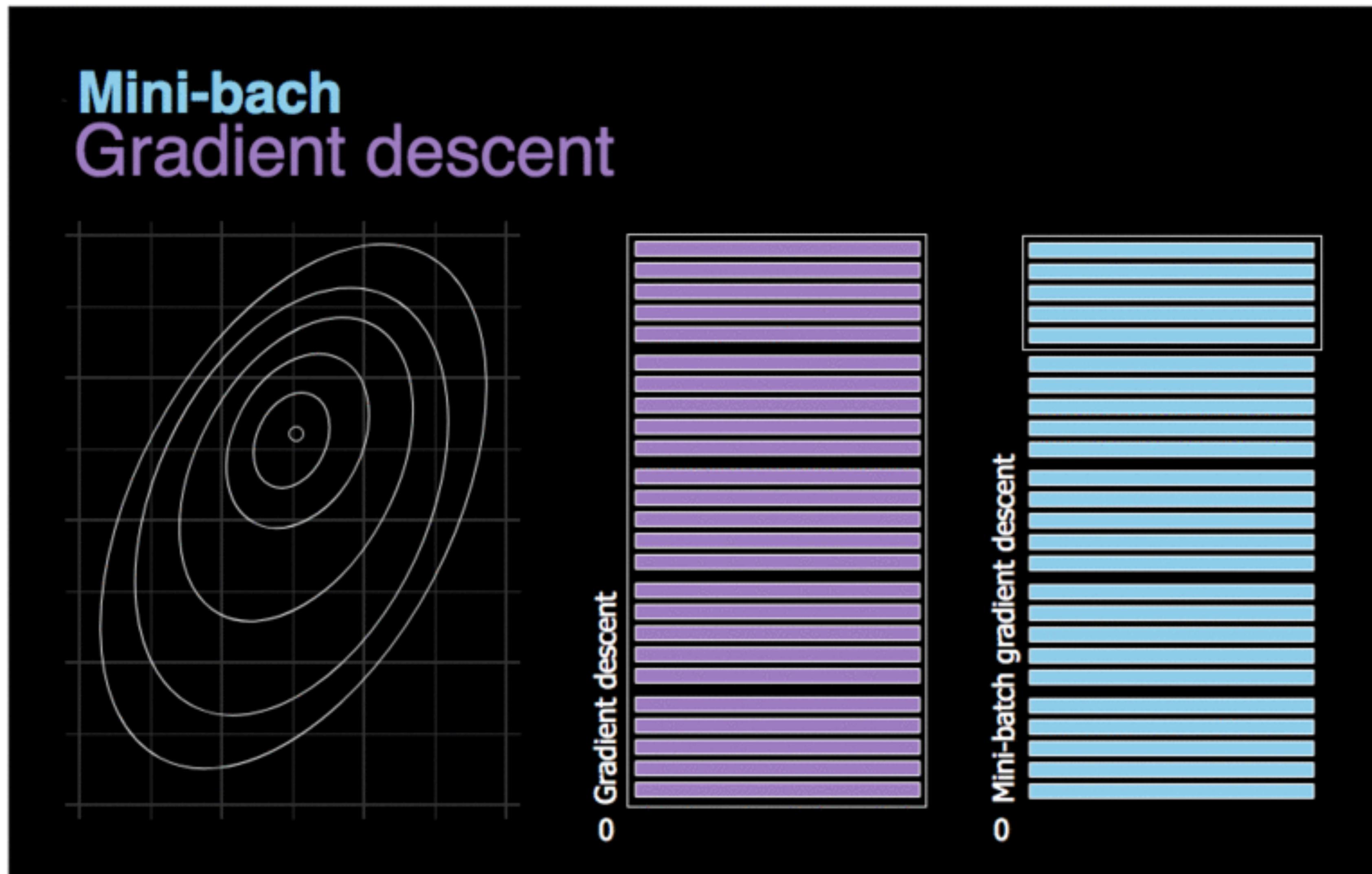
$$O(p)$$

- When is $n \gg p$? Big-data regime!

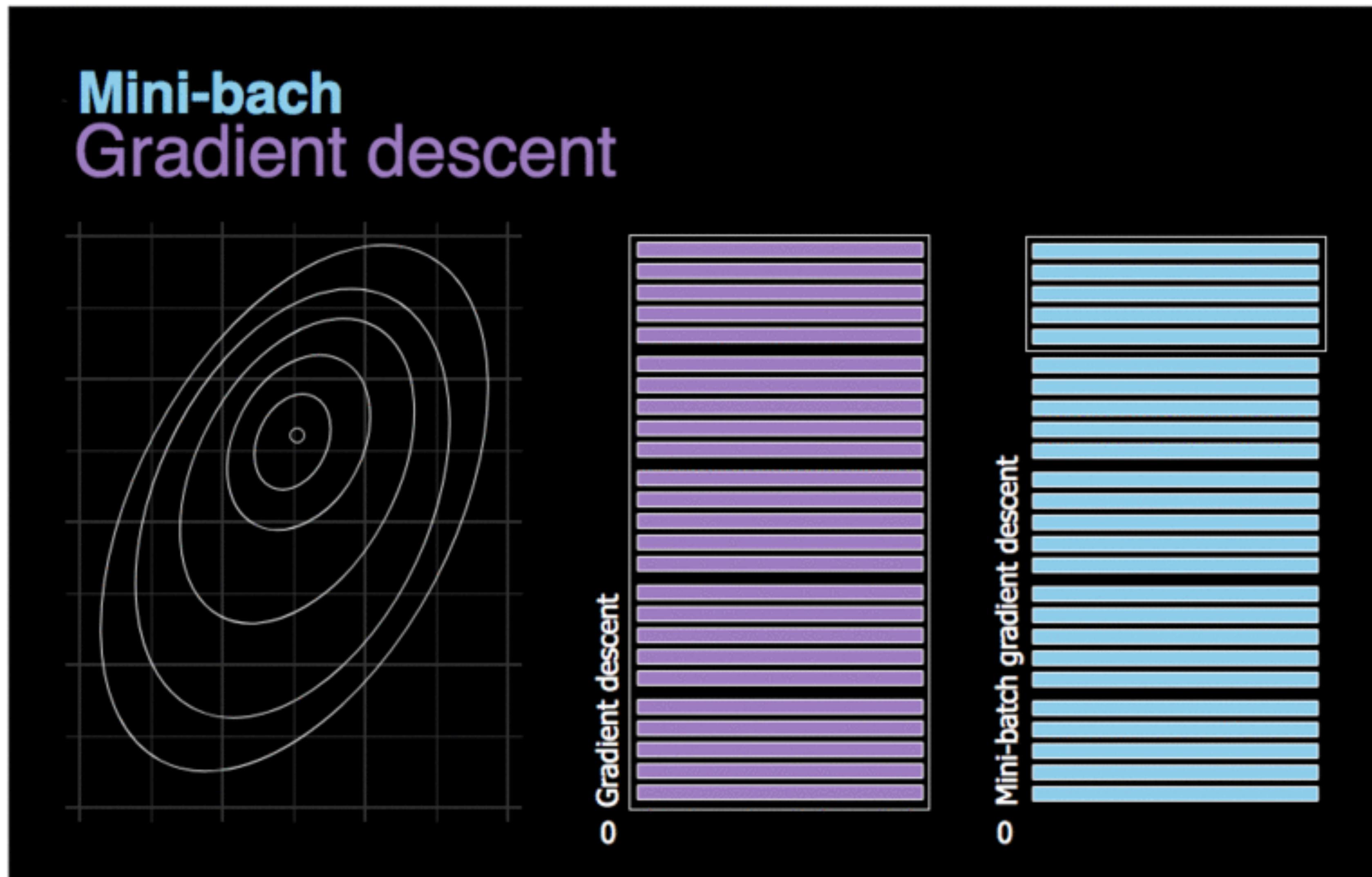
- There is redundancy in data

- Far from the optimal, exact gradients might have small returns

Acceleration #2: Cut-off complexity per iteration



Acceleration #2: Cut-off complexity per iteration



Acceleration #2: Cut-off complexity per iteration

- Some notes on SGD (before we proceed)

Acceleration #2: Cut-off complexity per iteration

- Some notes on SGD (before we proceed)
 1. It is a stochastic process that depends on a random sequence $i_t \in [n]$

(This means you will see some probability involved in theory)

Acceleration #2: Cut-off complexity per iteration

- Some notes on SGD (before we proceed)
 1. It is a stochastic process that depends on a random sequence $i_t \in [n]$
(This means you will see some probability involved in theory)
 2. While $-\nabla f(x_t)$ might be a descent direction, $-\nabla f_{i_t}(x_t)$ might not be
(This means that some tools from deterministic optimization do not hold here)

Acceleration #2: Cut-off complexity per iteration

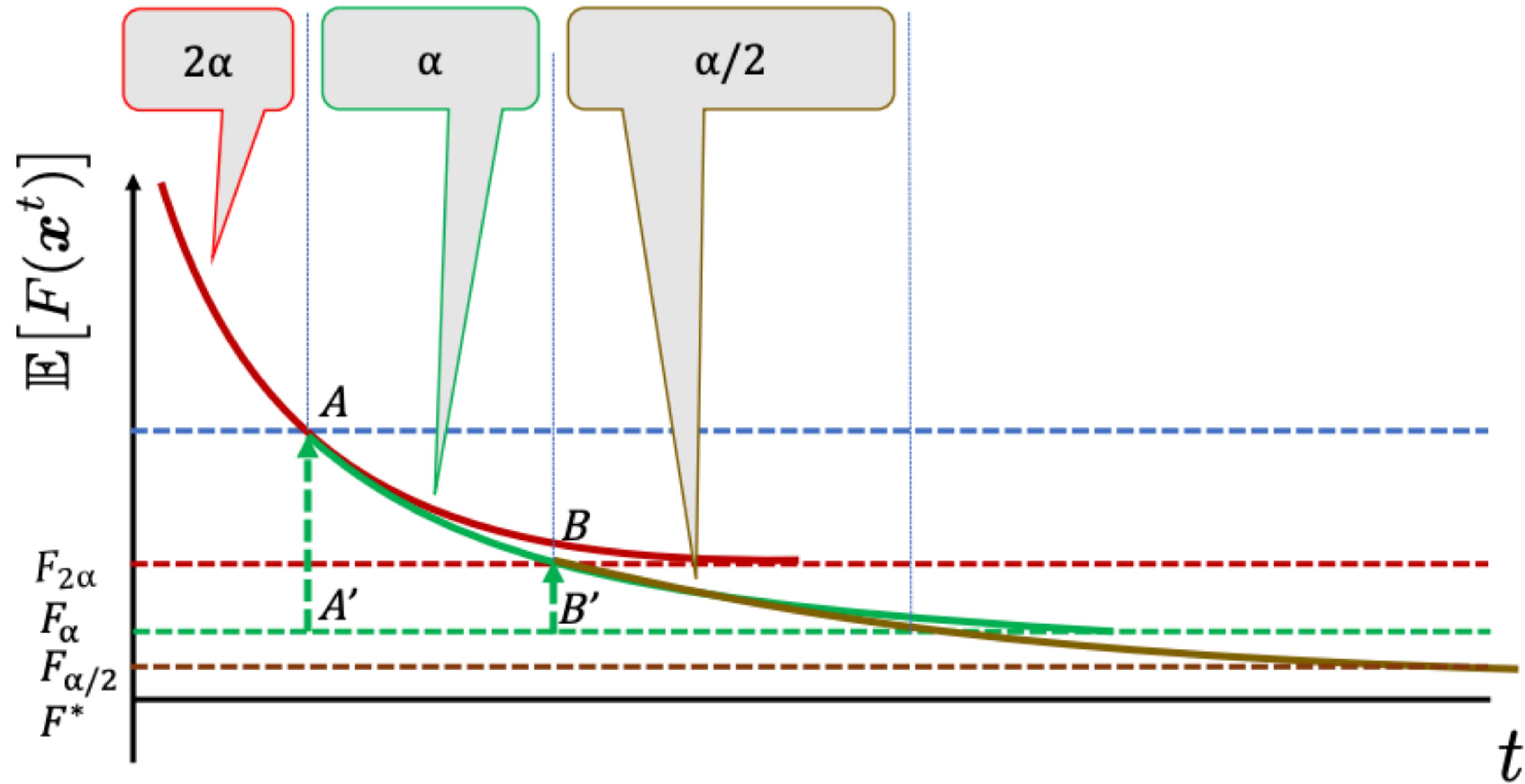
- Some notes on SGD (before we proceed)
 1. It is a stochastic process that depends on a random sequence $i_t \in [n]$
(This means you will see some probability involved in theory)
 2. While $-\nabla f(x_t)$ might be a descent direction, $-\nabla f_{i_t}(x_t)$ might not be
(This means that some tools from deterministic optimization do not hold here)
 3. The above lead to the intuition that if we have a descent direction in **expectation**, we probably will do just fine
(This means that we will work with expectations w.r.t. the random sequence)

Guarantees of SGD

Whiteboard

Guarantees of SGD

Pic. from: “Optimization Methods for Large-Scale Machine Learning”



- Start with large step size; decrease it when SGD “stalls”

Guarantees of SGD

Whiteboard

Intuition behind preference for SGD

- Overall, for strongly convex and smooth functions

	iteration complexity	per-iteration cost	total comput. cost
batch GD	$\log \frac{1}{\varepsilon}$	n	$n \log \frac{1}{\varepsilon}$
SGD	$\frac{1}{\varepsilon}$	1	$\frac{1}{\varepsilon}$

Intuition behind preference for SGD

- Overall, for strongly convex and smooth functions

	iteration complexity	per-iteration cost	total comput. cost
batch GD	$\log \frac{1}{\varepsilon}$	n	$n \log \frac{1}{\varepsilon}$
SGD	$\frac{1}{\varepsilon}$	1	$\frac{1}{\varepsilon}$

- The real comparison is between $n \log \frac{1}{\varepsilon}$? $\frac{1}{\varepsilon}$
- In the big data regime, n can be huge!

Intuition behind preference for SGD

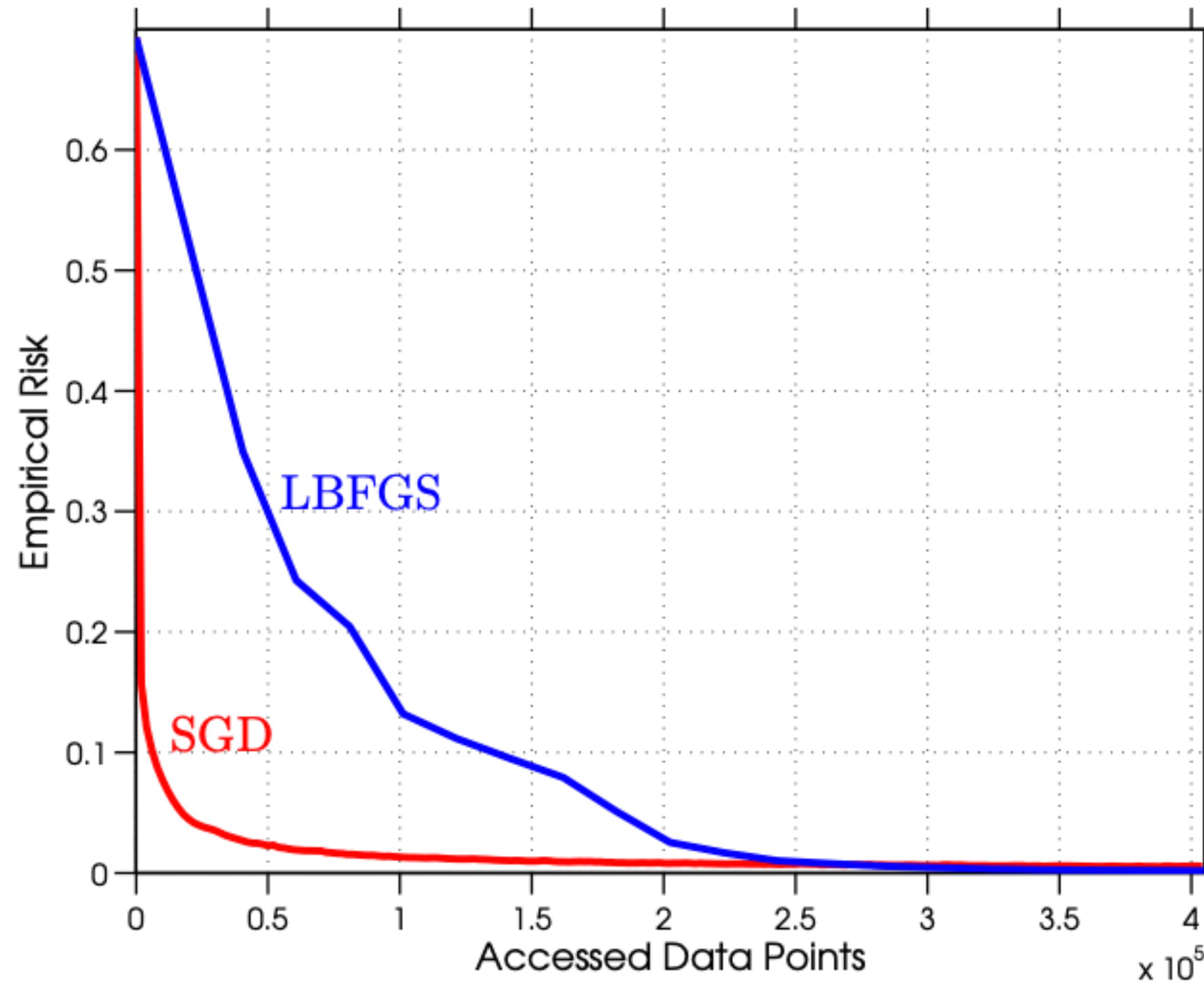
- Overall, for strongly convex and smooth functions

	iteration complexity	per-iteration cost	total comput. cost
batch GD	$\log \frac{1}{\varepsilon}$	n	$n \log \frac{1}{\varepsilon}$
SGD	$\frac{1}{\varepsilon}$	1	$\frac{1}{\varepsilon}$

- The real comparison is between $n \log \frac{1}{\varepsilon}$? $\frac{1}{\varepsilon}$
- In the big data regime, n can be huge!
- Gradient descent uses full dataset per iteration; there might be **redundancies**
- It actually works great in practice!

Intuition behind preference for SGD

Pic. from: “Optimization Methods for Large-Scale Machine Learning“



Variants of SGD

- Mini-batch SGD: instead of picking one sample, pick multiple

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

- Still less time than computing the full gradient
- Converges to a smaller ball around optimum: trade-off

Variants of SGD

- Mini-batch SGD: instead of picking one sample, pick multiple

$$x_{t+1} = x_t - \eta_t \nabla f_{\mathcal{I}_t}(x_t) = x_t - \eta_t \cdot \sum_{j=1}^{|\mathcal{I}_t|} \nabla f_j(x_t)$$

- Still less time than computing the full gradient
- Converges to a smaller ball around optimum: trade-off
- SGD with importance sampling: select “carefully” the next sample
 - Select $i_t \in [n]$ according to distribution $p \in [0, 1]^n$, $\sum_i p_i = 1$
 - Main question: can we compute a good probability distribution without too much effort?

Variants of SGD

- Stochastic variance-reduced gradient (SVRG)

$$x_{t+1} = x_t - \eta_t (\nabla f_{i_t}(x_t) - (\nabla f_{i_t}(\tilde{x}_q) - \nabla f(\tilde{x}_q)))$$

Variants of SGD

- Stochastic variance-reduced gradient (SVRG)

$$x_{t+1} = x_t - \eta_t (\nabla f_{i_t}(x_t) - (\nabla f_{i_t}(\tilde{x}_q) - \nabla f(\tilde{x}_q)))$$

Variants of SGD

- Stochastic variance-reduced gradient (SVRG)

$$x_{t+1} = x_t - \eta_t (\nabla f_{i_t}(x_t) - (\nabla f_{i_t}(\tilde{x}_q) - \nabla f(\tilde{x}_q)))$$



Bias in gradient estimate
Correction term

Variants of SGD

- Stochastic variance-reduced gradient (SVRG)

$$x_{t+1} = x_t - \eta_t (\nabla f_{i_t}(x_t) - (\nabla f_{i_t}(\tilde{x}_q) - \nabla f(\tilde{x}_q)))$$



Bias in gradient estimate
Correction term

- Observe that: $\mathbb{E}[\nabla f_{i_t}(\cdot)] = \nabla f(\cdot)$; then

$$\mathbb{E} [\nabla f_{i_t}(x_t) - \nabla f_{i_t}(\tilde{x}_q) + \nabla f(\tilde{x}_q)] = \nabla f(x_t)$$

Unbiased estimator!
We expect smaller variance

Variants of SGD

- Stochastic variance-reduced gradient (SVRG)

$$x_{t+1} = x_t - \eta_t (\nabla f_{i_t}(x_t) - (\nabla f_{i_t}(\tilde{x}_q) - \nabla f(\tilde{x}_q)))$$



Bias in gradient estimate
Correction term

- Observe that: $\mathbb{E}[\nabla f_{i_t}(\cdot)] = \nabla f(\cdot)$; then

$$\mathbb{E} [\nabla f_{i_t}(x_t) - \nabla f_{i_t}(\tilde{x}_q) + \nabla f(\tilde{x}_q)] = \nabla f(x_t)$$

Unbiased estimator!
We expect smaller variance

- Theoretical guarantees:

$$\mathbb{E} [f(x_{t+1}) - f(x^*)] \leq \rho \cdot \mathbb{E} [f(x_t) - f(x^*)] , \quad \rho = O\left(\frac{1}{1-2\eta L} \cdot \left(\frac{1}{m\eta} + 2L\eta\right)\right) < 1$$

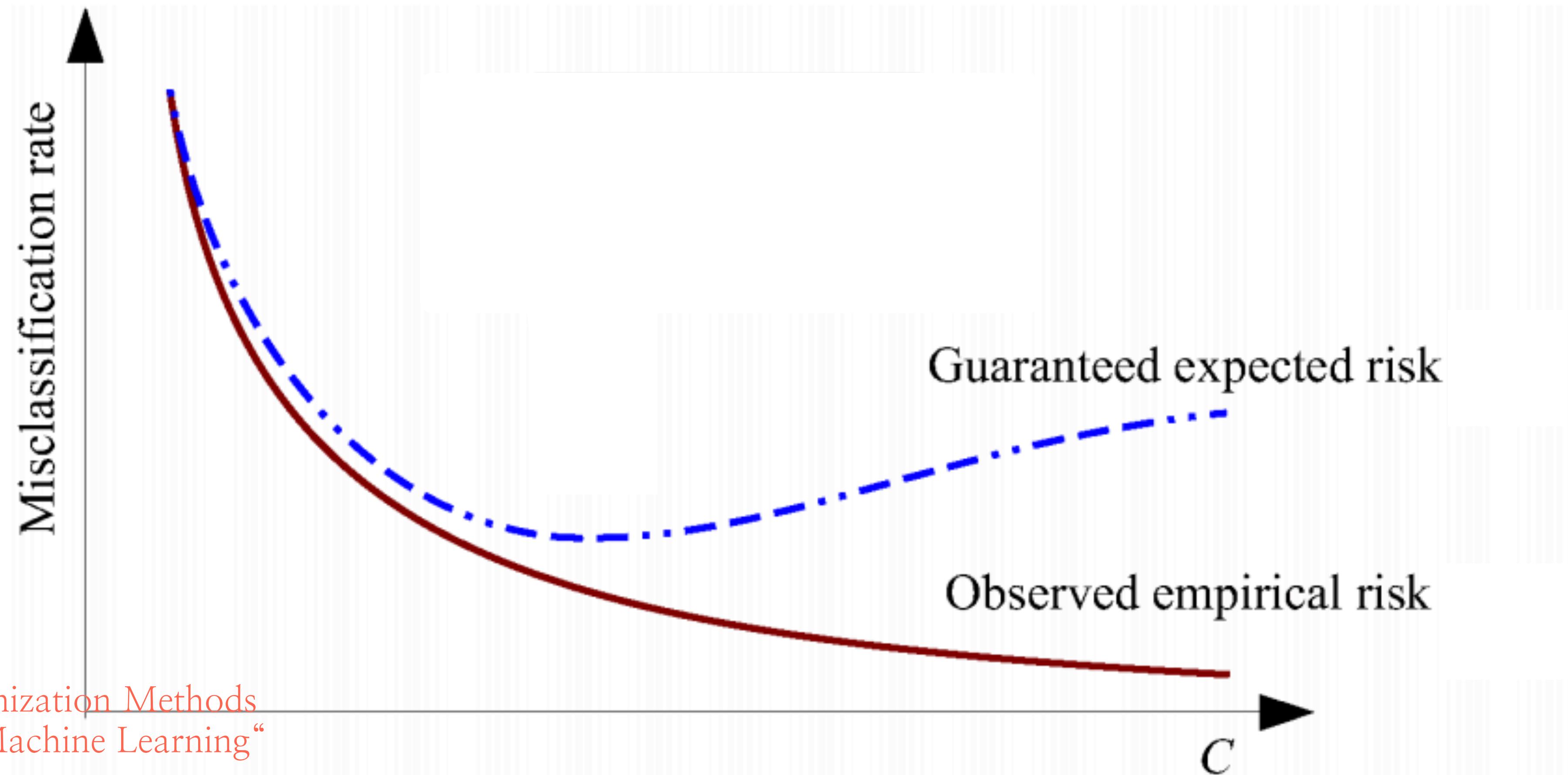
- Main drawback: Full gradient, but overall complexity $O\left((n + \kappa) \log \frac{1}{\varepsilon}\right)$

Performance of SGD

Demo

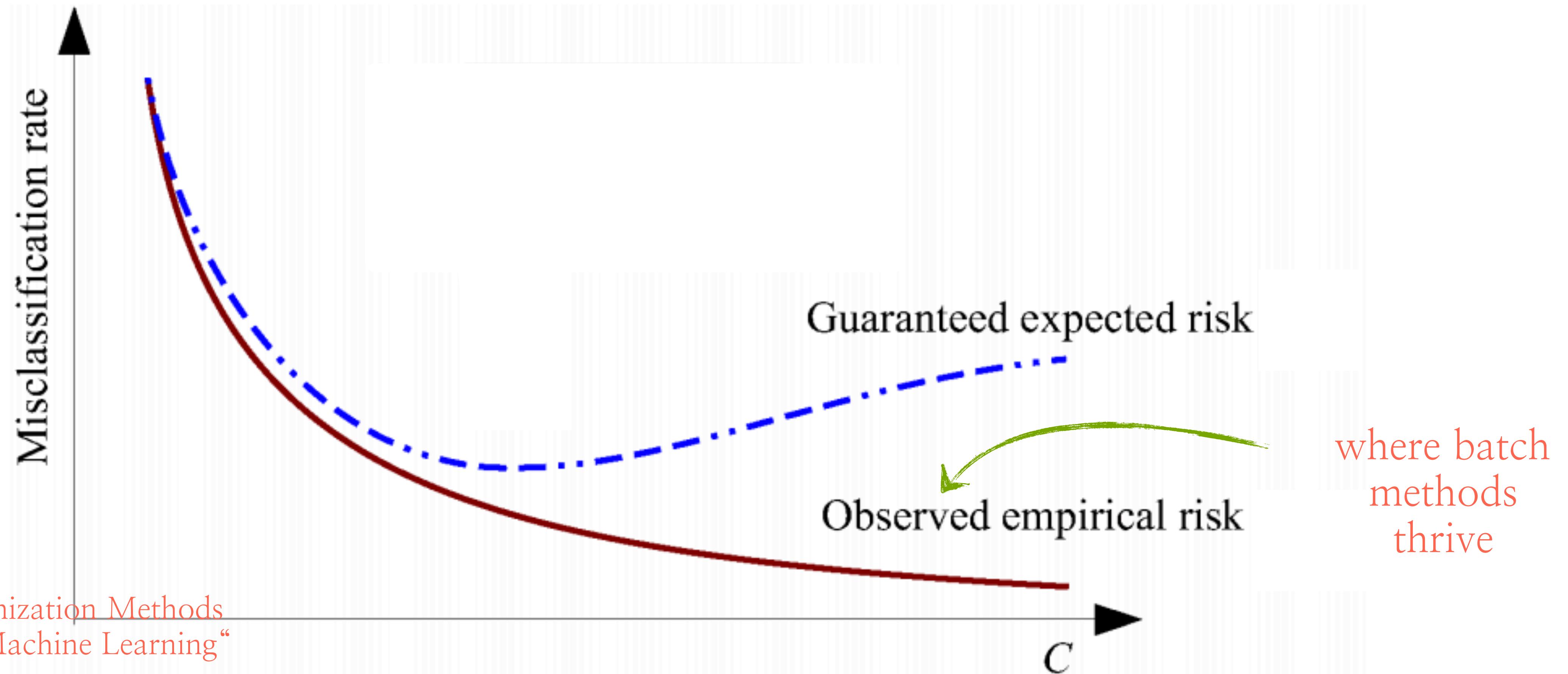
Why SGD is so important in machine learning?

(or some of the reasons)



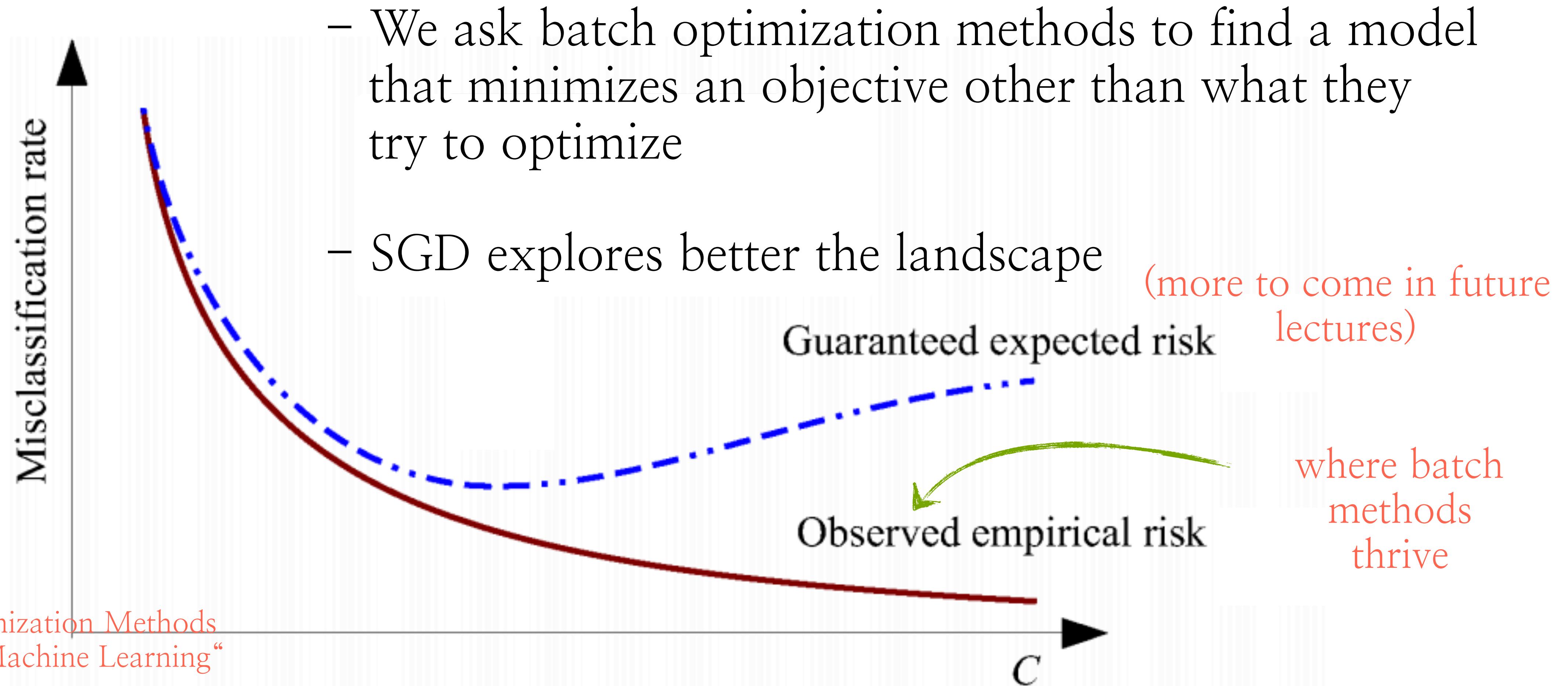
Why SGD is so important in machine learning?

(or some of the reasons)



Why SGD is so important in machine learning?

(or some of the reasons)



Why using vanilla Newton's method might not be a
good idea

(but it is an active area of research, as
things are not 100% clear)

Why using vanilla Newton's method might not be a good idea

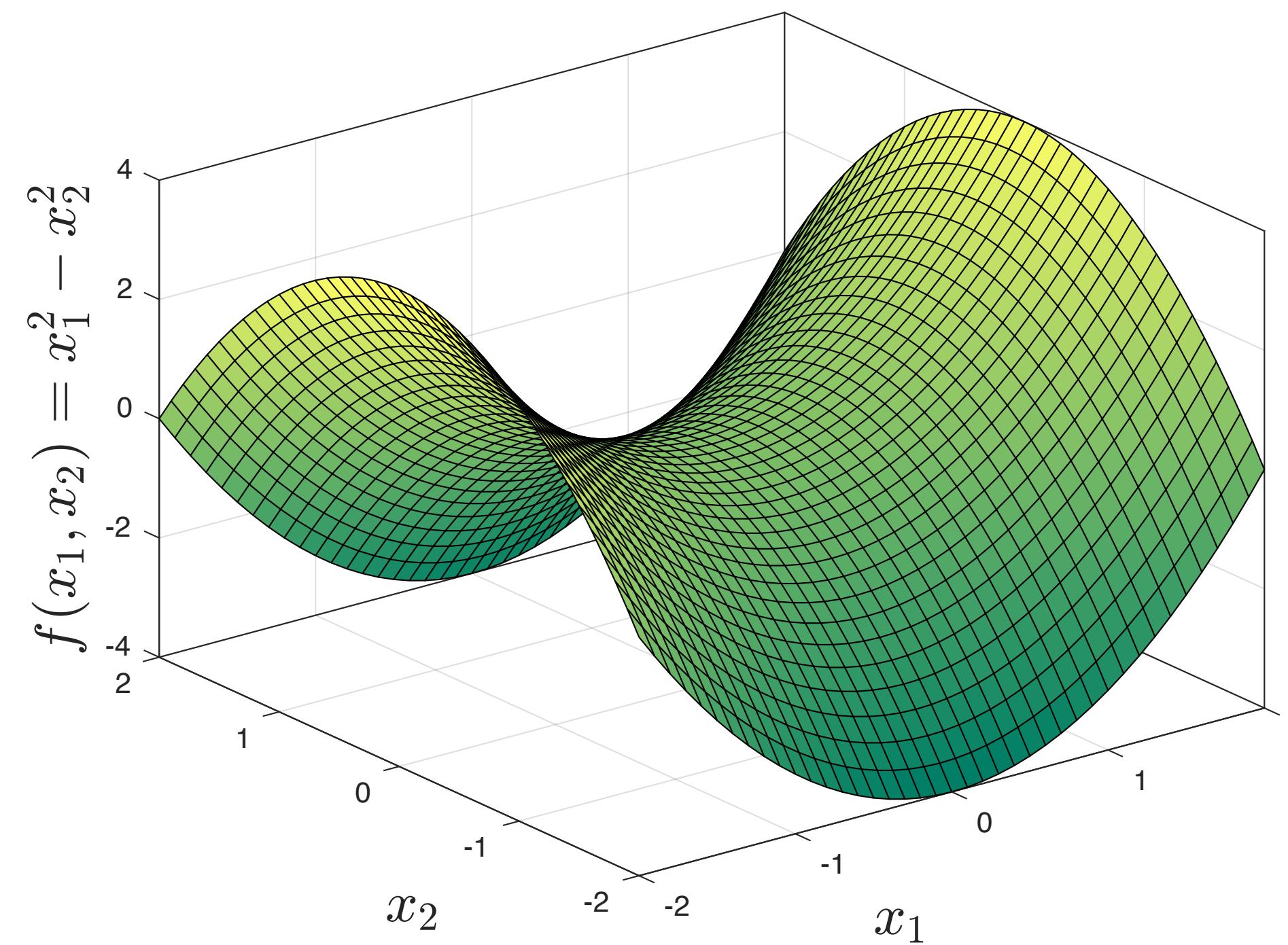
(but it is an active area of research, as things are not 100% clear)

- Consider the following simple example: $f(x_1, x_2) = x_1^2 - x_2^2$

Why using vanilla Newton's method might not be a good idea

(but it is an active area of research, as things are not 100% clear)

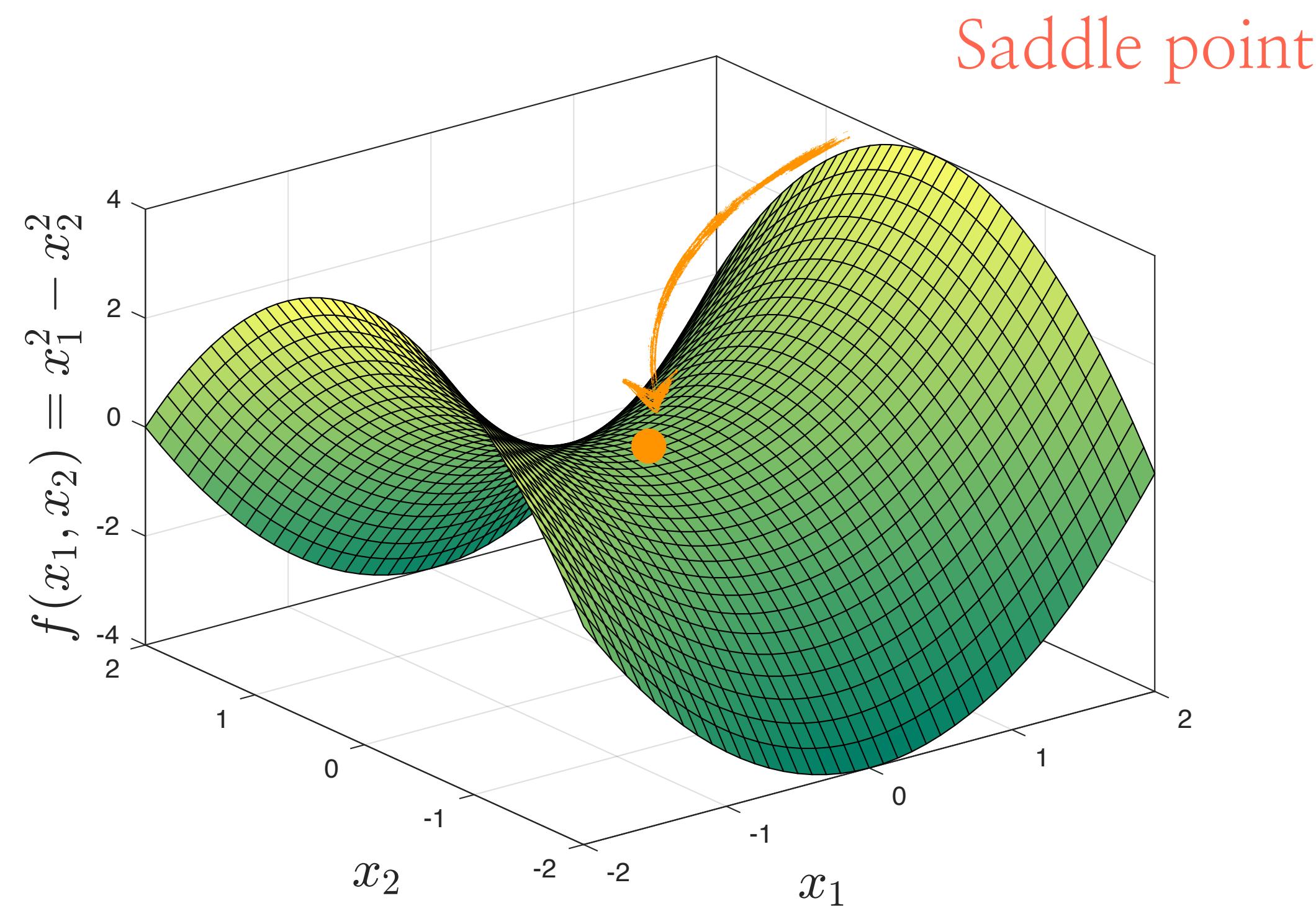
- Consider the following simple example: $f(x_1, x_2) = x_1^2 - x_2^2$



Why using vanilla Newton's method might not be a good idea

(but it is an active area of research, as things are not 100% clear)

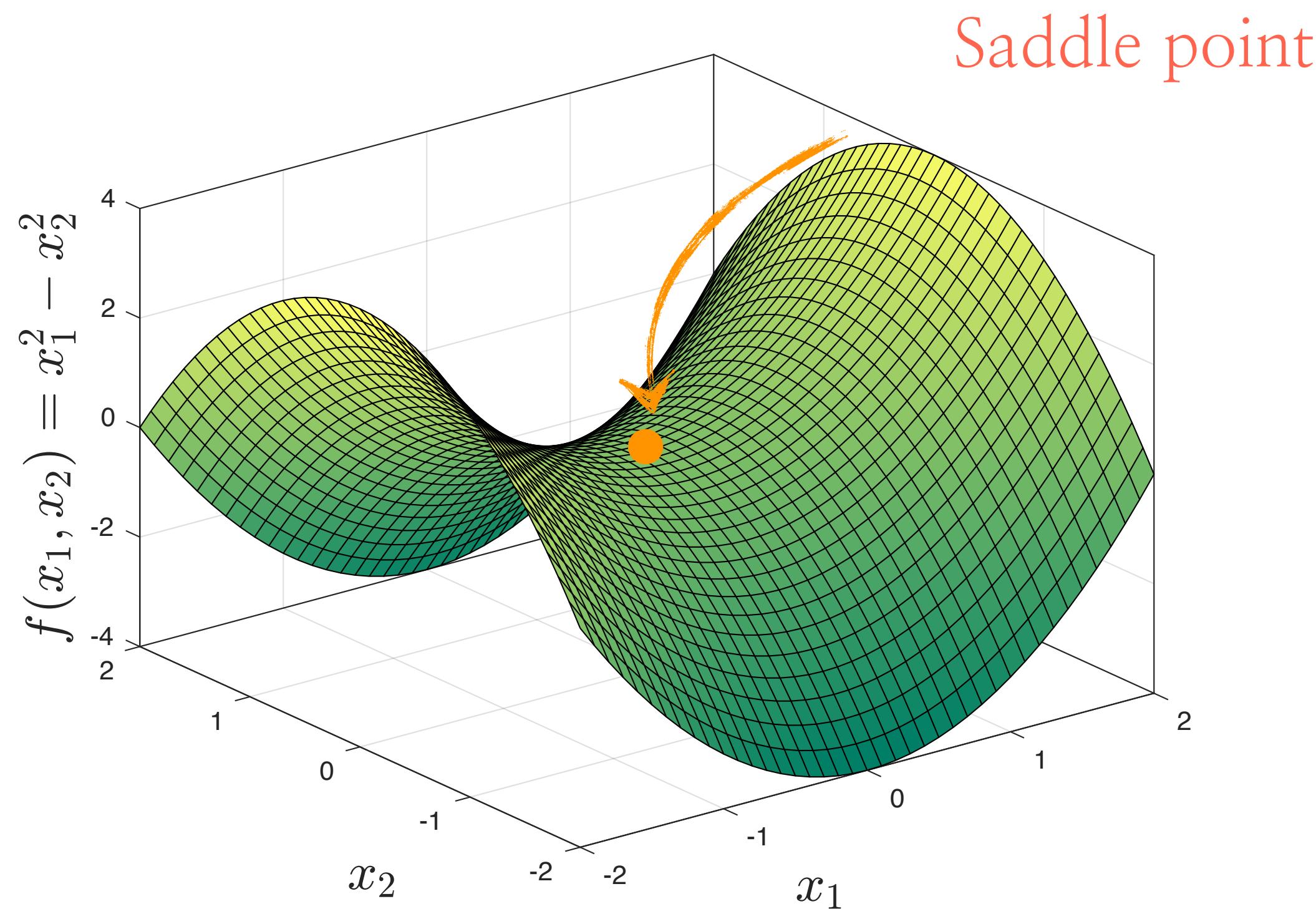
- Consider the following simple example: $f(x_1, x_2) = x_1^2 - x_2^2$



Why using vanilla Newton's method might not be a good idea

(but it is an active area of research, as things are not 100% clear)

- Consider the following simple example: $f(x_1, x_2) = x_1^2 - x_2^2$



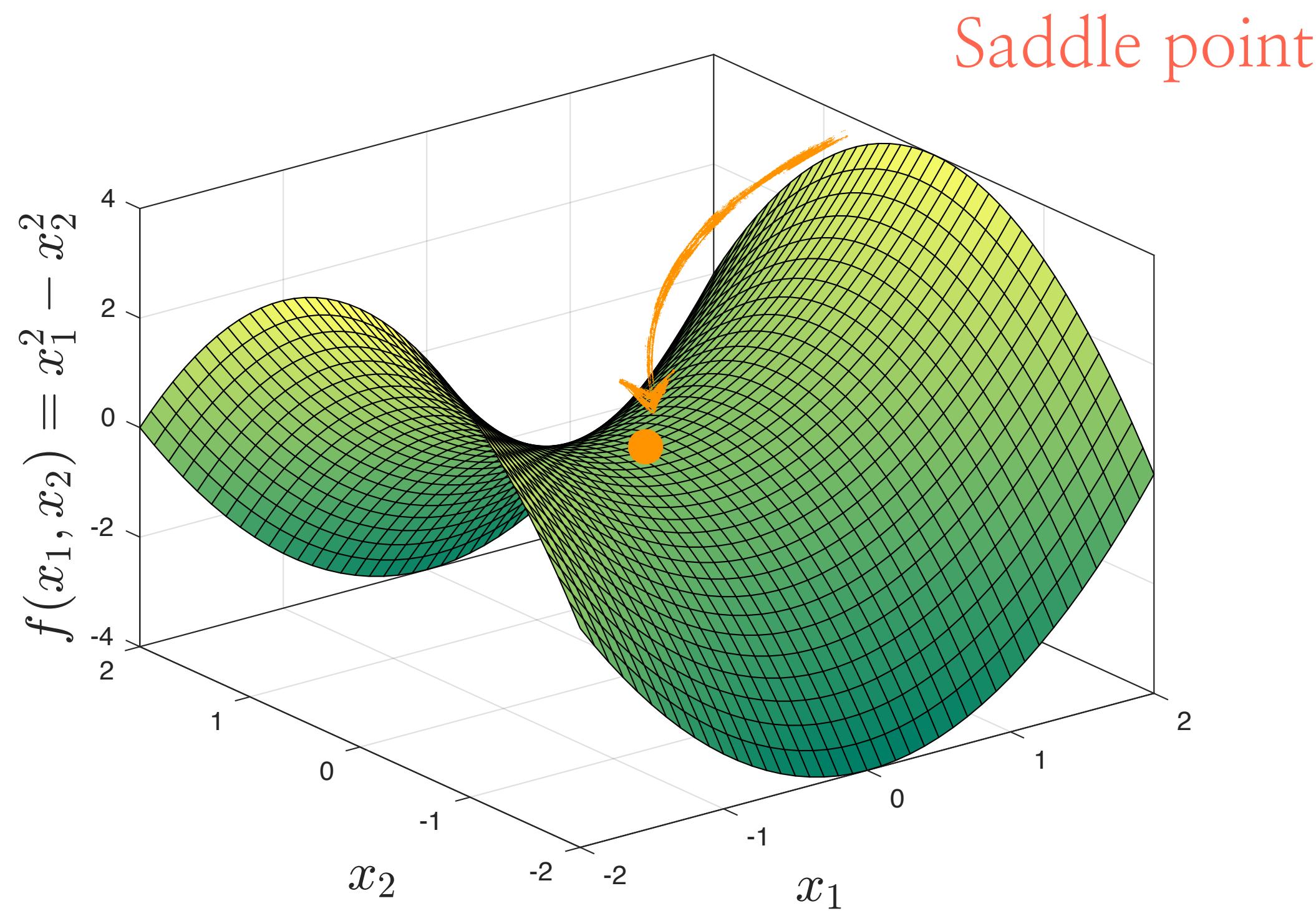
- Compute the Hessian:

$$H = \begin{bmatrix} \frac{\partial^2 f(x_1, x_2)}{\partial x_1^2} & \frac{\partial^2 f(x_1, x_2)}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f(x_1, x_2)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x_1, x_2)}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$$

Why using vanilla Newton's method might not be a good idea

(but it is an active area of research, as things are not 100% clear)

- Consider the following simple example: $f(x_1, x_2) = x_1^2 - x_2^2$



- Compute the Hessian:

$$H = \begin{bmatrix} \frac{\partial^2 f(x_1, x_2)}{\partial x_1^2} & \frac{\partial^2 f(x_1, x_2)}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f(x_1, x_2)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x_1, x_2)}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$$

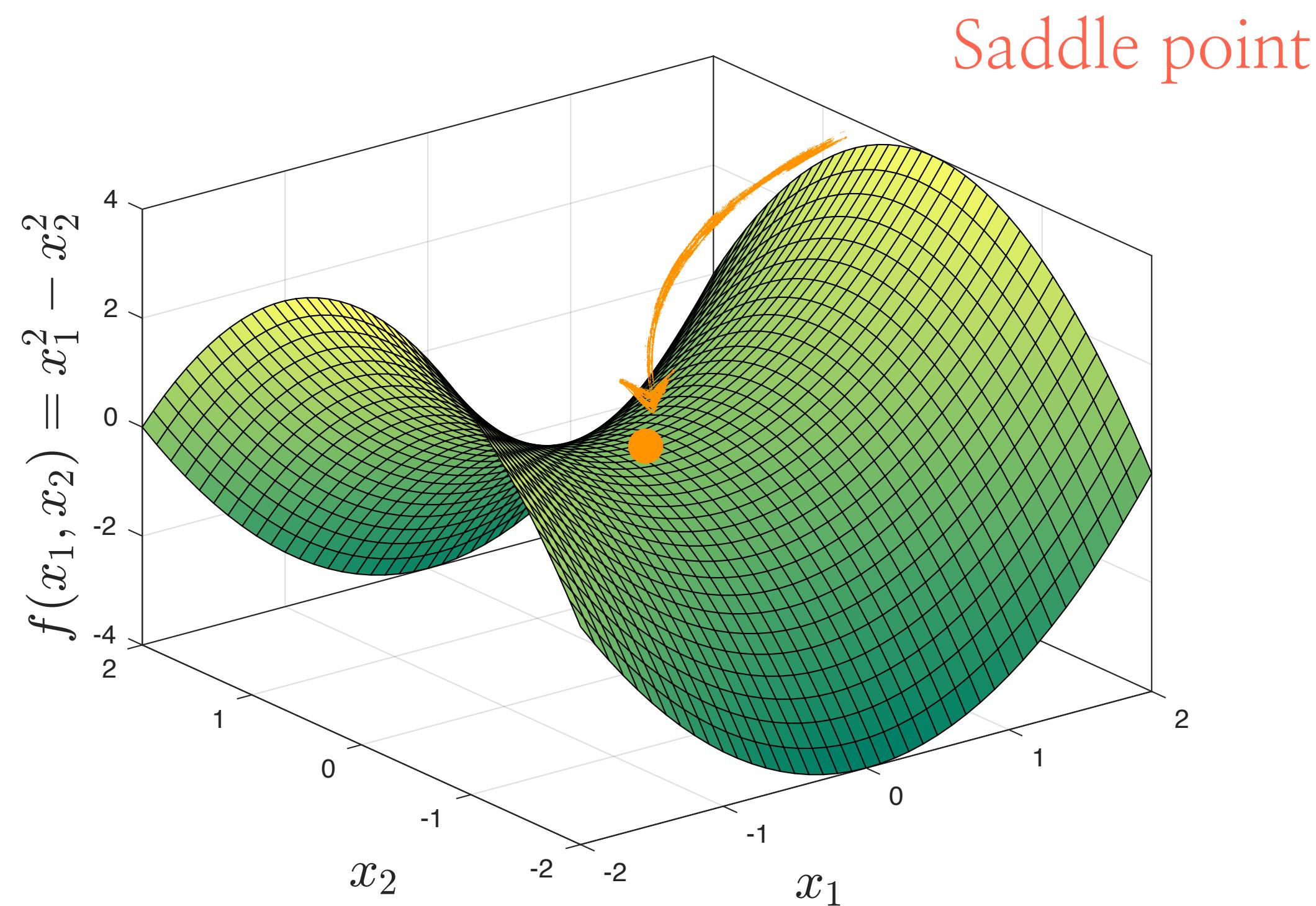
- Compute the inverse Hessian:

$$H^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & -1/2 \end{bmatrix}$$

Why using vanilla Newton's method might not be a good idea

(but it is an active area of research, as things are not 100% clear)

- Consider the following simple example: $f(x_1, x_2) = x_1^2 - x_2^2$



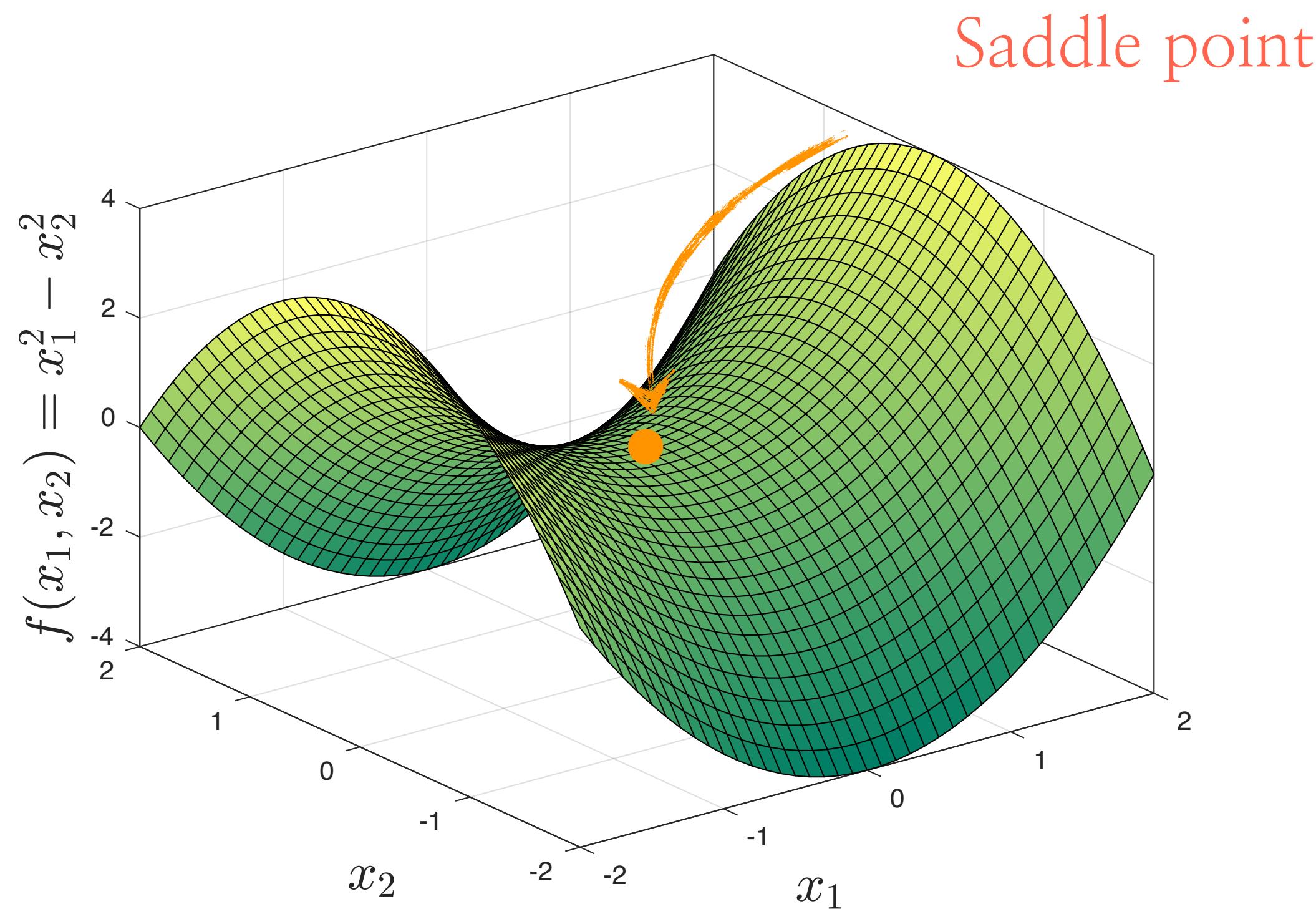
- Let's perform a single Newton iteration:

$$\begin{aligned}x_{t+1} &= x_t - H^{-1} \nabla f(x_t) \\&= \begin{bmatrix}(x_t)_1 \\ (x_t)_2\end{bmatrix} - \begin{bmatrix}1/2 & 0 \\ 0 & -1/2\end{bmatrix} \begin{bmatrix}2(x_t)_1 \\ 2(x_t)_2\end{bmatrix} \\&= \begin{bmatrix}(x_t)_1 \\ (x_t)_2\end{bmatrix} - \begin{bmatrix}(x_t)_1 \\ (x_t)_2\end{bmatrix} = \begin{bmatrix}0 \\ 0\end{bmatrix}\end{aligned}$$

Why using vanilla Newton's method might not be a good idea

(but it is an active area of research, as things are not 100% clear)

- Consider the following simple example: $f(x_1, x_2) = x_1^2 - x_2^2$



- Let's perform a single Newton iteration:

$$\begin{aligned}x_{t+1} &= x_t - H^{-1} \nabla f(x_t) \\&= \begin{bmatrix}(x_t)_1 \\ (x_t)_2\end{bmatrix} - \begin{bmatrix}1/2 & 0 \\ 0 & -1/2\end{bmatrix} \begin{bmatrix}2(x_t)_1 \\ 2(x_t)_2\end{bmatrix} \\&= \begin{bmatrix}(x_t)_1 \\ (x_t)_2\end{bmatrix} - \begin{bmatrix}(x_t)_1 \\ (x_t)_2\end{bmatrix} = \begin{bmatrix}0 \\ 0\end{bmatrix}\end{aligned}$$

For this toy example, Newton seems to be attracted to the saddle point!

Acceleration #3: Coordinate descent methods

- Stochastic gradient descent (SGD) selects mini batches of training data; what if we subselect variables to update per iteration?

$$x_{t+1} = x_t - \eta_t \nabla f(x_t) \quad \longrightarrow \quad (x_{t+1})_{i_t} = (x_t)_{i_t} - \eta_t \nabla_{i_t} f(x_t),$$

where per iteration we select randomly $i_t \in [p]$.

Acceleration #3: Coordinate descent methods

- Stochastic gradient descent (SGD) selects mini batches of training data; what if we subselect variables to update per iteration?

$$x_{t+1} = x_t - \eta_t \nabla f(x_t) \quad \longrightarrow \quad (x_{t+1})_{i_t} = (x_t)_{i_t} - \eta_t \nabla_{i_t} f(x_t),$$

where per iteration we select randomly $i_t \in [p]$.

- “Why do we want to do this?”

What is the complexity of
computing full gradient?

$O(np)$

(Assume least-squares objective)

What is the complexity of
computing a gradient for a
single variable?

$O(n)$

Acceleration #3: Coordinate descent methods

- Stochastic gradient descent (SGD) selects mini batches of training data; what if we subselect variables to update per iteration?

$$x_{t+1} = x_t - \eta_t \nabla f(x_t) \quad \longrightarrow \quad (x_{t+1})_{i_t} = (x_t)_{i_t} - \eta_t \nabla_{i_t} f(x_t),$$

where per iteration we select randomly $i_t \in [p]$.

- “Why do we want to do this?”

What is the complexity of
computing full gradient?

$O(np)$

(Assume least-squares objective)

What is the complexity of
computing a gradient for a
single variable?

$O(n)$

- When is $n \ll p$? High-dimensional case

- There is not enough data

- We will see that it provides solutions to distributed systems

Papers to review – due next Tuesday

(Select one of the following papers)

- “Adaptive Restart for Accelerated Gradient Schemes”, O’Donoghue and Candes, 2012.
- “Why momentum really works”, Goh, 2017.
- “Solving large-scale linear prediction problems using SGD”, Zhang, 2004.
- “Stochastic Gradient Descent, Weighted Sampling, and the Randomized Kaczmarz algorithm”, Needell et al., 2015.
- “Coordinate descent algorithms”, Wright, 2015.

(The first 20 pages)

Conclusion

- We studied algorithms beyond gradient descent: Newton’s method, quasi–Newton algorithms, derivative–free optimization, acceleration methods, and stochastic variants
- Which one to use depends on the problem at hand (accuracy, complexity)
- There’s a lot of tuning going on..
..which is probably the most crucial part, if we want to be fair in comparison!

Next lecture

- We will discuss a bit about **hyper-parameter tuning**