COMP 545: Advanced topics in optimization
From simple to complex ML systems

Lecture 9

Overview

- Different objective classes
- Different strategies within each problem
- Different approaches based on computational capabilities
- Different approaches based on constraints

$$\min_x \quad f(x)$$

$$\text{s.t.} \quad x \in \mathcal{C}$$

And, always having in mind applications in machine learning, AI and signal processing

About what we have talked so far

$$\min_{x} \quad f(x)$$

$$\text{s.t.} \quad x \in \mathcal{C}$$

About what we have talked so far

$$\min_{x} \quad f(x)$$

$$\cancel{\text{s.t.} \quad x \in \mathcal{C}}$$

Unconstrained

About what we have talked so far

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

$$\cancel{\text{s.t.} \quad x \in \mathcal{C}}$$

Unconstrained

About what we have talked so far

Convex

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

$$\cancel{\text{s.t.} \quad x \in \mathcal{C}}$$

Unconstrained

About what we have talked so far

Huge!

Convex

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

$$\cancel{\text{s.t.} \quad x \in \mathcal{C}}$$

Unconstrained

About what we have talked so far

Huge!

Convex

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

$$\cancel{\text{s.t.} \quad x \in \mathcal{C}}$$

Unconstrained

- There is a lot of work on such settings (mostly in the convex domain)

(Early stage of modern ML, but out of scope of this course)

About what we have talked so far

Huge!

Convex

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

$$\cancel{\text{s.t.} \quad x \in \mathcal{C}}$$

Unconstrained

- There is a lot of work on such settings (mostly in the convex domain)

  (Early stage of modern ML, but out of scope of this course)

- We focused though on complex ways to deal with such problems: **asynchrony**

  (We proved convergence under standard conditions and staleness conditions)

The focus of this lecture

Huge!

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

$$\text{s.t.} \quad x \in \mathcal{C}$$

The focus of this lecture

Huge!

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

$$\cancel{\text{s.t.} \quad x \in \mathcal{C}}$$

Unconstrained

The focus of this lecture

Huge!

Non−convex!

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

$$\cancel{\text{s.t.} \quad x \in \mathcal{C}}$$

Unconstrained

Overview

- In this lecture, we will:
  - Go back to the initial discussion of non-convex optimization

Overview

- In this lecture, we will:
    - Go back to the initial discussion of non-convex optimization
    - We will provide generic convergence results for stochastic methods

(More general case than whatever non-convex problem we considered so far)

Overview

- In this lecture, we will:
  - Go back to the initial discussion of non-convex optimization
  - We will provide generic convergence results for stochastic methods

    (More general case than whatever non-convex problem we considered so far)
  - Inspired by modern ML (neural networks), we will describe alternatives to SGD:
    - Accelerated SGD
    - RMSProp
    - AdaGrad
    - Adam

Overview

- In this lecture, we will:
  - Go back to the initial discussion of non-convex optimization
  - We will provide generic convergence results for stochastic methods

    (More general case than whatever non-convex problem we considered so far)
  - Inspired by modern ML (neural networks), we will describe alternatives to SGD:
    - Accelerated SGD
    - AdaGrad
    - RMSProp
    - Adam

  - Bonus discussion: **The marginal value of adaptive methods**

Recall: Stochastic gradient descent

- SGD is used **almost everywhere**: training classical ML tasks (linear prediction, linear classification), training modern ML tasks (non-linear classification, neural networks)

Recall: Stochastic gradient descent

- SGD is used **almost everywhere**: training classical ML tasks (linear prediction, linear classification), training modern ML tasks (non−linear classification, neural networks)
- In simple math, it satisfies:

$$x_{t+1} = x_t - \eta \nabla f_{i_t}(x_t)$$

based on the objective: $$\min_x \quad f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

Non−convex!

Recall: Stochastic gradient descent

- SGD is used **almost everywhere**: training classical ML tasks (linear prediction, linear classification), training modern ML tasks (non-linear classification, neural networks)

- In simple math, it satisfies:

$$x_{t+1} = x_t - \eta \nabla f_{i_t}(x_t)$$

based on the objective: $$\min_x \quad f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

Non-convex!

- Why SGD is preferable over full-batch GD?

Recall: Stochastic gradient descent

- SGD is used **almost everywhere**: training classical ML tasks (linear prediction, linear classification), training modern ML tasks (non-linear classification, neural networks)
- In simple math, it satisfies:

$$x_{t+1} = x_t - \eta \nabla f_{i_t}(x_t)$$

based on the objective: $\min_{x} \quad f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x)$

Non-convex!

- Why SGD is preferable over full-batch GD?
  - Full-batch GD performs **redundant computations** for large datasets

Recall: Stochastic gradient descent

- SGD is used **almost everywhere**: training classical ML tasks (linear prediction, linear classification), training modern ML tasks (non-linear classification, neural networks)

- In simple math, it satisfies:

$$x_{t+1} = x_t - \eta \nabla f_{i_t}(x_t)$$

based on the objective: $\min_x \quad f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x)$

Non-convex!

- Why SGD is preferable over full-batch GD?
  - Full-batch GD performs **redundant computations** for large datasets
  - SGD's fluctuations enables it to **jump to potentially better local minima**

Recall: Stochastic gradient descent

- SGD is used **almost everywhere**: training classical ML tasks (linear prediction, linear classification), training modern ML tasks (non-linear classification, neural networks)
- In simple math, it satisfies:

$$x_{t+1} = x_t - \eta \nabla f_{i_t}(x_t)$$

  based on the objective: $\min_x \quad f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x)$

  Non-convex!

- Why SGD is preferable over full-batch GD?
  - Full-batch GD performs **redundant computations** for large datasets
  - SGD's fluctuations enables it to **jump to potentially better local minima**
- However, SGD's proof for non-convex settings is more **complicated + weaker**

SGD convergence result in non-convex scenaria

Whiteboard

SGD convergence result in non-convex scenaria

Whiteboard

- Key observations:
  - For convergence, this theory assumes a small step size $O\left(\frac{1}{\sqrt{T}}\right)$
    - In a sense, we need to know a priori the number of iterations to achieve ε-approximation
    - Step size can be bad at the beginning – other step sizes used in practice

SGD convergence result in non-convex scenaria

Whiteboard

- Key observations:
  - For convergence, this theory assumes a small step size $O\left(\frac{1}{\sqrt{T}}\right)$
    - In a sense, we need to know a priori the number of iterations to achieve ε-approximation
    - Step size can be bad at the beginning – other step sizes used in practice
  - Nevertheless, in practice SGD performs favorably compared to full-batch GD.

SGD convergence result in non-convex scenaria

Whiteboard

- Key observations:
  - For convergence, this theory assumes a small step size $O\left(\frac{1}{\sqrt{T}}\right)$
    - In a sense, we need to know a priori the number of iterations to achieve ε-approximation
    - Step size can be bad at the beginning – other step sizes used in practice
  - Nevertheless, in practice SGD performs favorably compared to full-batch GD.
  - Assuming more structure (e.g., PL condition), one can achieve better rates with constant step sizes (independent on the number of iterations)

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: GD vs Acc. GD

Strongly Convex

Acceleration in SGD in non-convex scenaria

– General observation: moving results from convex to non-convex settings is not straightforward in most cases

– Recall: GD vs Acc. GD

Strongly Convex

$$O\left(\kappa \log \frac{f(x_0) - f^\star}{\varepsilon}\right)$$

Acceleration in SGD in non−convex scenaria

– General observation: moving results from convex to non−convex settings is not straightforward in most cases

– Recall: GD vs Acc. GD

Strongly Convex

$$O\left(\kappa \log \frac{f(x_0) - f^\star}{\varepsilon}\right) \qquad O\left(\sqrt{\kappa} \log \frac{f(x_0) - f^\star}{\varepsilon}\right)$$

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: GD vs Acc. GD

Strongly Convex

$$O\left(\boxed{\kappa}\log\frac{f(x_0)-f^\star}{\varepsilon}\right) \qquad \text{vs} \qquad O\left(\boxed{\sqrt{\kappa}}\log\frac{f(x_0)-f^\star}{\varepsilon}\right)$$

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: GD vs Acc. GD

Strongly Convex

$$O\left(\boxed{\kappa}\log\frac{f(x_0)-f^\star}{\varepsilon}\right) \qquad O\left(\boxed{\sqrt{\kappa}}\log\frac{f(x_0)-f^\star}{\varepsilon}\right)$$

- GD vs Acc. GD

Non Convex

(To get to a point such that $\|\nabla f(\cdot)\|_2 \leq \varepsilon$)

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: GD vs Acc. GD

Strongly Convex

$$O\left(\boxed{\kappa}\log\frac{f(x_0)-f^\star}{\varepsilon}\right) \qquad O\left(\boxed{\sqrt{\kappa}}\log\frac{f(x_0)-f^\star}{\varepsilon}\right)$$

- GD vs Acc. GD

Non Convex

$$O\left(\frac{1}{\varepsilon^2}\right)$$

(To get to a point such that $\|\nabla f(\cdot)\|_2 \leq \varepsilon$)

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: GD vs Acc. GD

Strongly Convex

$$O\left(\boxed{\kappa}\log \frac{f(x_0) - f^\star}{\varepsilon}\right) \qquad \text{vs} \qquad O\left(\boxed{\sqrt{\kappa}}\log \frac{f(x_0) - f^\star}{\varepsilon}\right)$$

- GD vs Acc. GD

Non Convex

$$O\left(\frac{1}{\varepsilon^2}\right) \qquad \text{vs} \qquad O\left(\frac{1}{\varepsilon^{7/4}} \cdot \log(1/\varepsilon)\right)$$

(To get to a point such that $\|\nabla f(\cdot)\|_2 \leq \varepsilon$)

Acceleration in SGD in non−convex scenaria

- General observation: moving results from convex to non−convex settings is not straightforward in most cases

- Recall: GD vs Acc. GD

Strongly Convex

$$O\left(\boxed{\kappa}\log\frac{f(x_0)-f^\star}{\varepsilon}\right) \qquad O\left(\boxed{\sqrt{\kappa}}\log\frac{f(x_0)-f^\star}{\varepsilon}\right)$$

- GD vs Acc. GD

Non Convex

$$O\left(\frac{1}{\varepsilon^2}\right) \qquad O\left(\frac{1}{\varepsilon^{7/4}}\cdot\log(1/\varepsilon)\right)$$

Acceleration: "get better than ε^{-2}"

(To get to a point such that $\|\nabla f(\cdot)\|_2 \le \varepsilon$)

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: SGD vs Acc. SGD

Strongly Convex

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: SGD vs Acc. SGD

Strongly Convex $O\left(\frac{1}{\varepsilon}\right)$

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: SGD vs Acc. SGD

	SGD	vs	Acc. SGD
Strongly Convex	$O\left(\frac{1}{\varepsilon}\right)$		(Results for specific cases – Still an open question in its most generality)

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

- Recall: SGD vs Acc. SGD

Strongly Convex

$O\left(\frac{1}{\varepsilon}\right)$

(Results for specific cases – Still an open question in its most generality)

- SGD vs Acc. SGD

Non Convex

(To get to a point such that $\|\nabla f(\cdot)\|_2 \leq \varepsilon$)

Acceleration in SGD in non-convex scenaria

– General observation: moving results from convex to non-convex settings is not straightforward in most cases

– Recall: SGD vs Acc. SGD

Strongly Convex

$O\left(\frac{1}{\varepsilon}\right)$

(Results for specific cases – Still an open question in its most generality)

– SGD vs Acc. SGD

Non Convex

$O\left(\frac{1}{\varepsilon^2}\right)$

(To get to a point such that $\|\nabla f(\cdot)\|_2 \leq \varepsilon$)

Acceleration in SGD in non-convex scenaria

– General observation: moving results from convex to non-convex settings is not straightforward in most cases

– Recall:	SGD	vs	Acc. SGD
Strongly Convex	$O\left(\frac{1}{\varepsilon}\right)$		(Results for specific cases – Still an open question in its most generality)
–	SGD	vs	Acc. SGD
Non Convex	$O\left(\frac{1}{\varepsilon^2}\right)$		(Results for specific cases – Still an open question in its most generality)

(To get to a point such that $\|\nabla f(\cdot)\|_2 \leq \varepsilon$)

Acceleration in SGD in non-convex scenaria

- General observation: moving results from convex to non-convex settings is not straightforward in most cases

– Recall:	SGD	vs	Acc. SGD
Strongly Convex	$O\left(\frac{1}{\varepsilon}\right)$		(Results for specific cases – Still an open question in its most generality)
–	SGD	vs	Acc. SGD
Non Convex	$O\left(\frac{1}{\varepsilon^2}\right)$		(Results for specific cases – Still an open question in its most generality)

(To get to a point such that $\|\nabla f(\cdot)\|_2 \leq \varepsilon$)

(We assume no variance reduction variants)

Acceleration in SGD in non-convex scenaria

Nevertheless, this does not prevent us from using acceleration in non-convex scenarios

https://www.tensorflow.org/api_docs/python/tf/train/MomentumOptimizer

Recall: Momentum acceleration

Recall: Momentum acceleration

Recall: Momentum acceleration

- Heavy ball method

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

Recall: Momentum acceleration

– Heavy ball method

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

Standard gradient step

Recall: Momentum acceleration

- Heavy ball method

$$x_{t+1} = \underbrace{x_t - \eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$

Recall: Momentum acceleration

- Heavy ball method

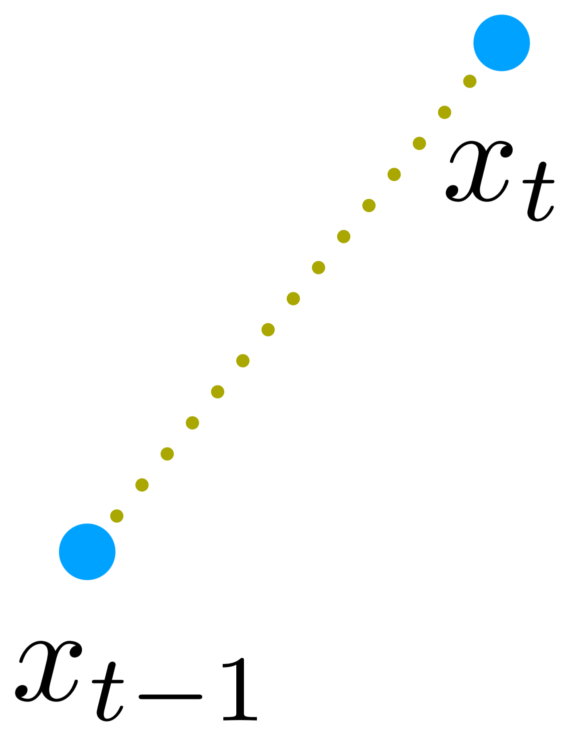
$$x_{t+1} = \underbrace{x_t - \eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$

x_{t-1}

Recall: Momentum acceleration

– Heavy ball method

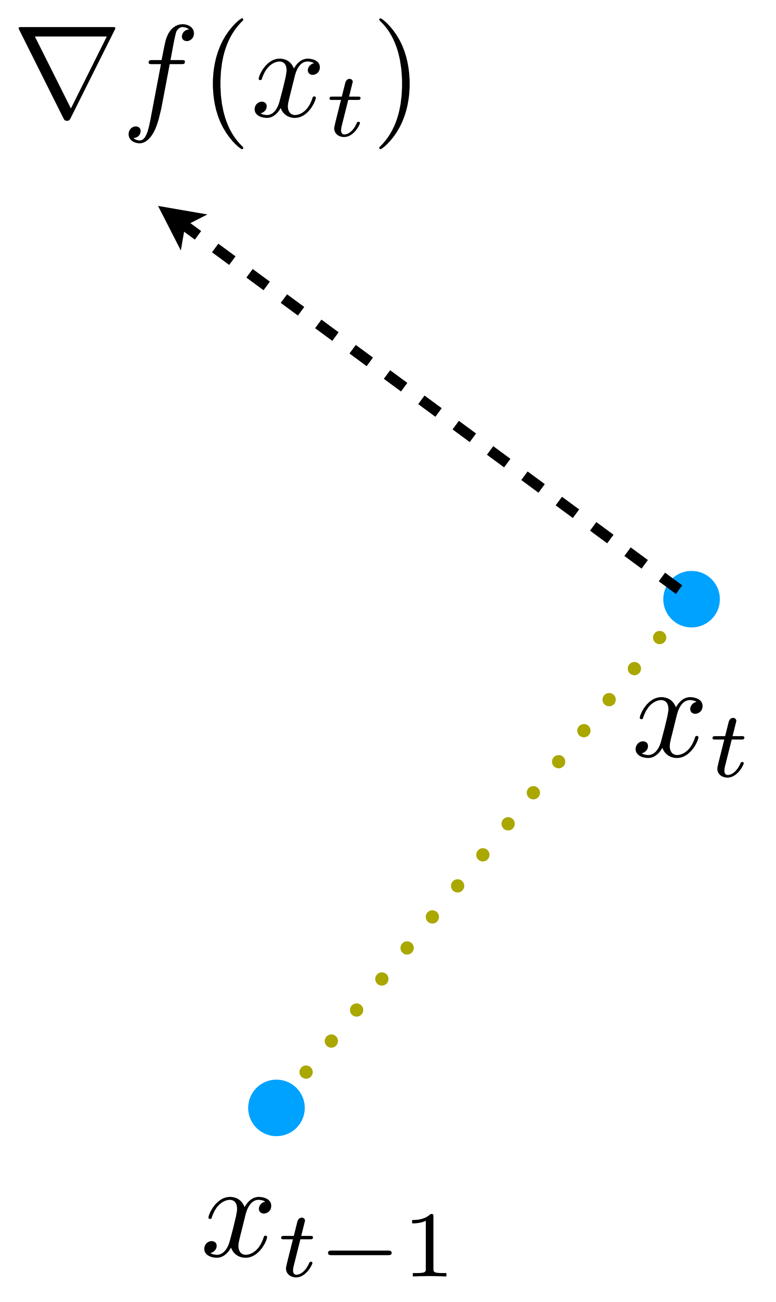
$$x_{t+1} = \underbrace{x_t - \eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$



Recall: Momentum acceleration

– Heavy ball method

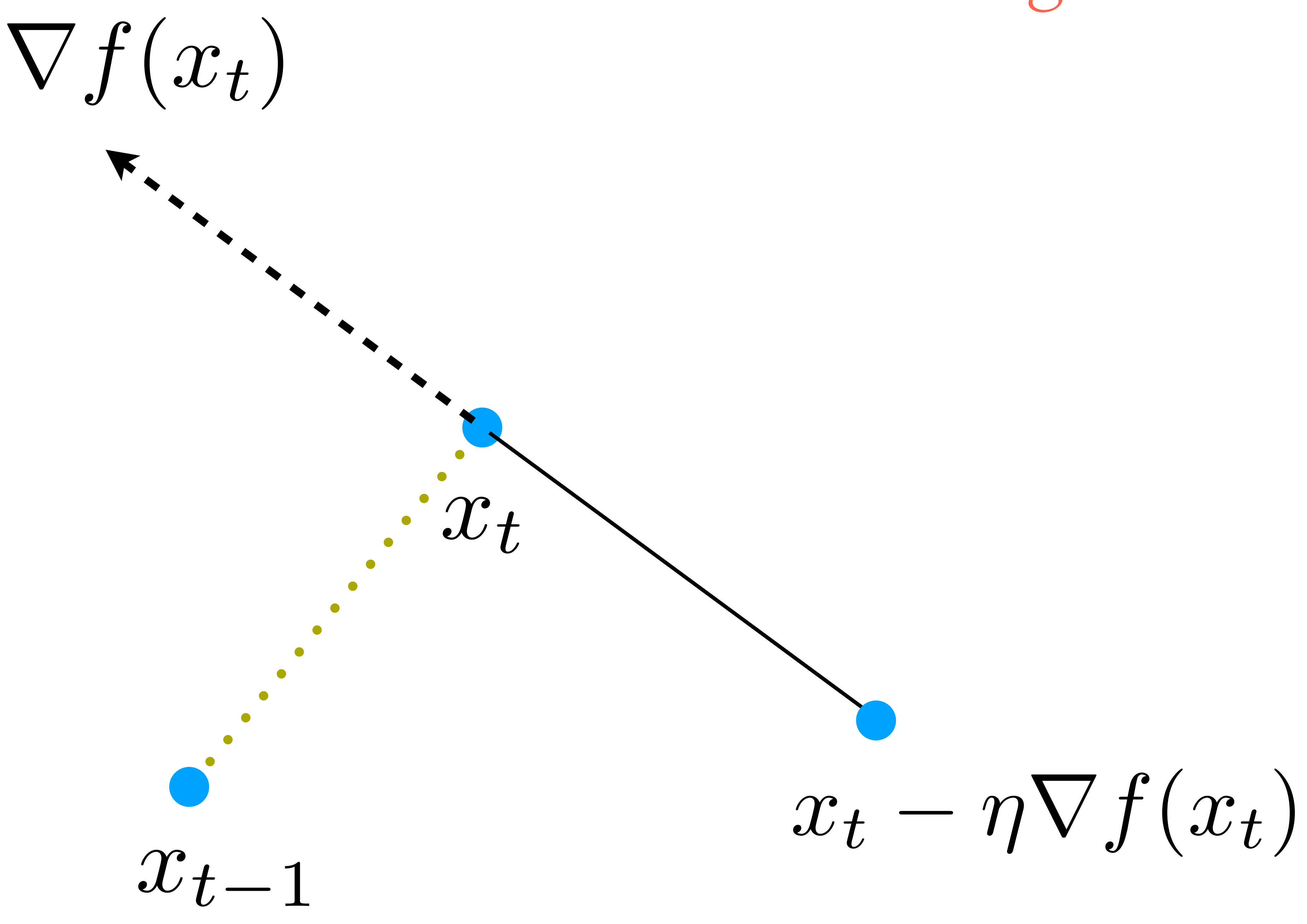
$$x_{t+1} = x_t \underbrace{- \eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$



Recall: Momentum acceleration

– Heavy ball method

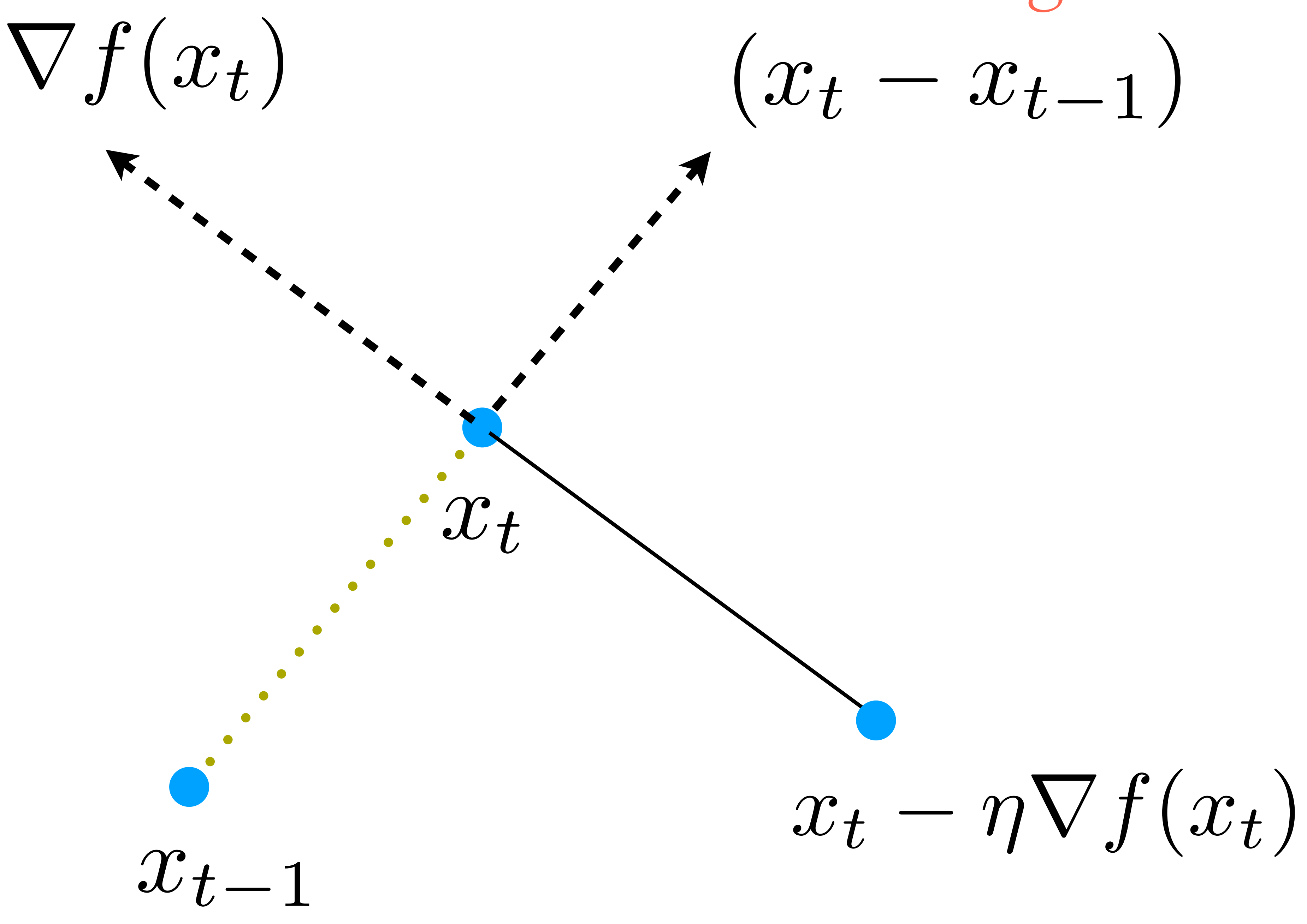
$$x_{t+1} = x_t - \underbrace{\eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$



Recall: Momentum acceleration

– Heavy ball method

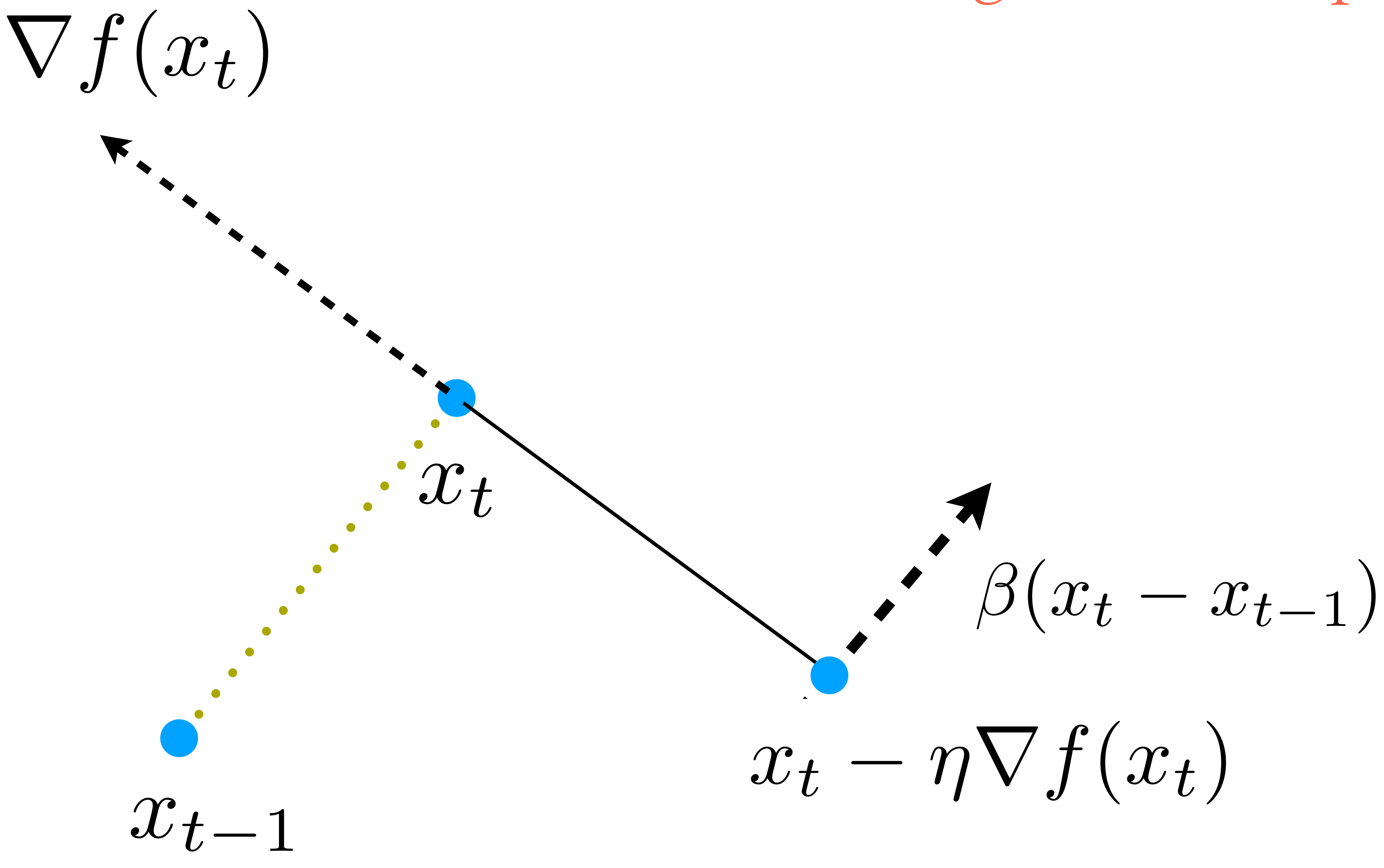
$$x_{t+1} = x_t \underbrace{- \eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$



Recall: Momentum acceleration

- Heavy ball method

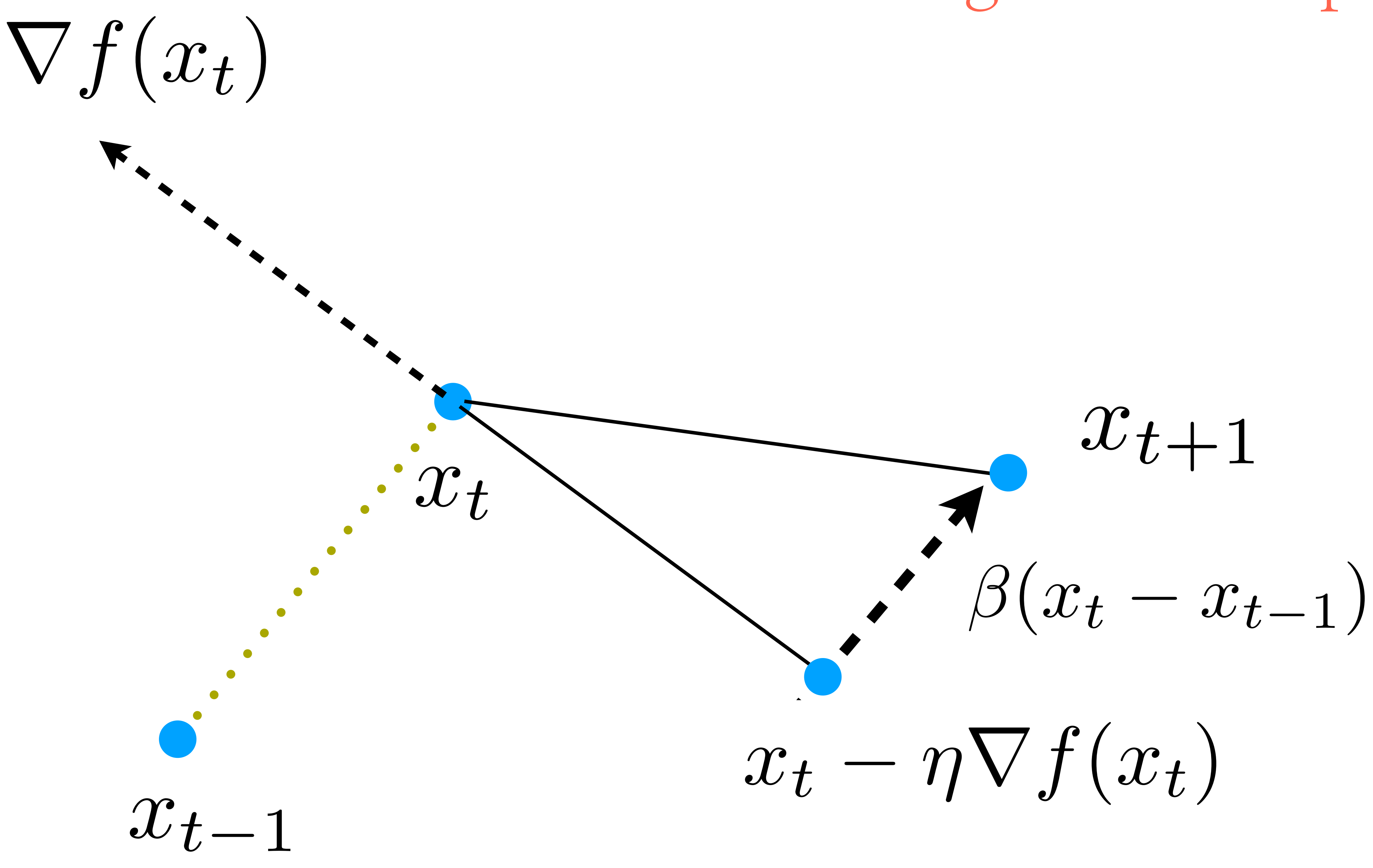
$$x_{t+1} = x_t \underbrace{- \eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$



Recall: Momentum acceleration

– Heavy ball method

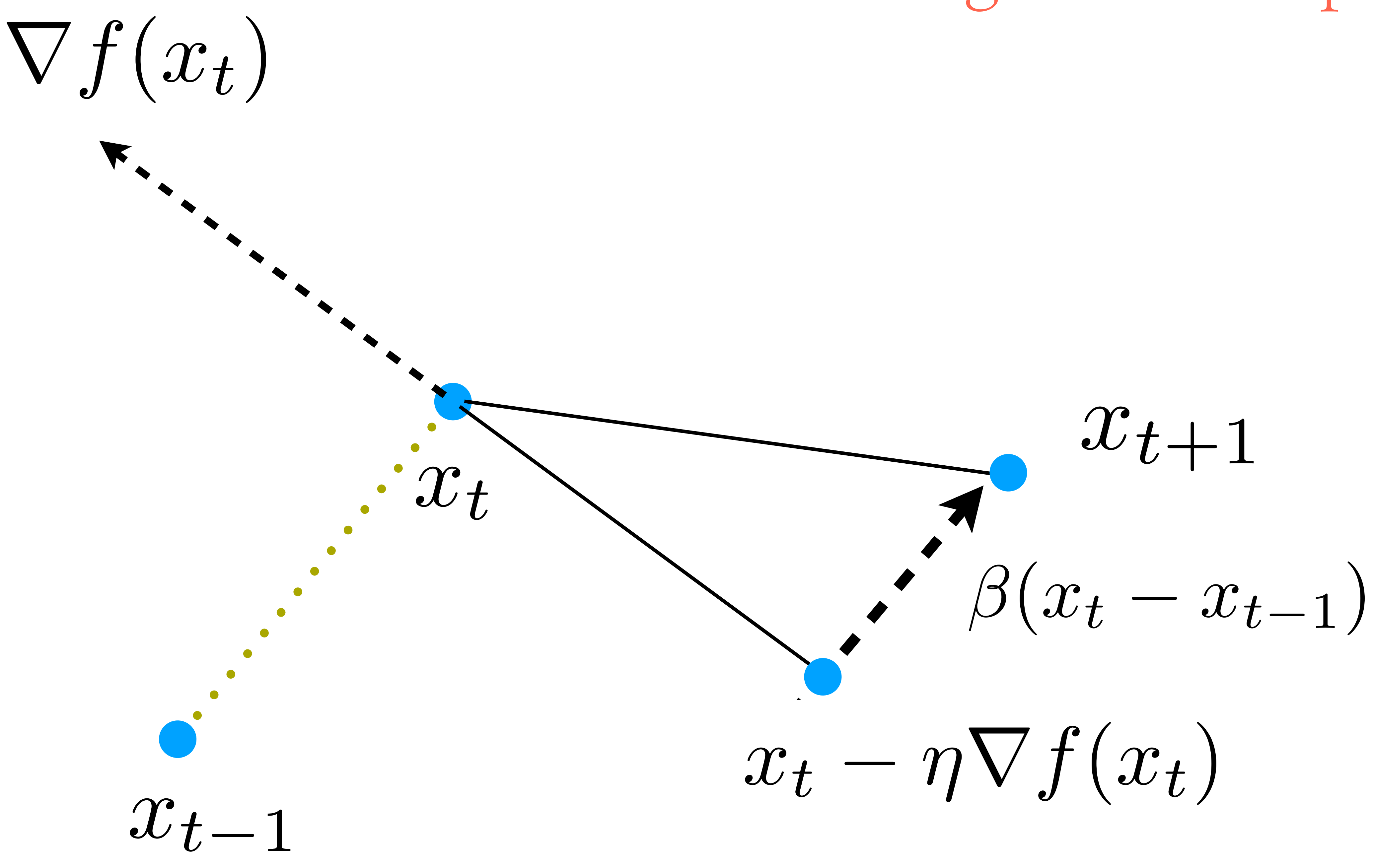
$$x_{t+1} = \underbrace{x_t - \eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$



Recall: Momentum acceleration

– Heavy ball method

$$x_{t+1} = \underbrace{x_t - \eta \nabla f(x_t)}_{\text{Standard gradient step}} + \underbrace{\beta(x_t - x_{t-1})}_{\text{Momentum step}}$$



Any analogy in the physical world?

– 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:

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

Guarantees of Heavy Ball method

Non−convex!

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

"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:

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

Recall: Momentum acceleration

- Nesterov's work: a collection of acceleration methods

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

Recall: Momentum acceleration

- Nesterov's work: a collection of acceleration methods

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

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

Recall: Momentum acceleration

– Nesterov's work: a collection of acceleration methods

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

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

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

Evaluate gradient at
current point

Recall: 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

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

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

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

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

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

Recall: 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

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

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

Nesterov's acceleration (1/2)

$$\boxed{\begin{aligned} \widetilde{x} &= x_t - \eta \nabla f(x_t + \beta(x_t - x_{t-1})) \\ x_{t+1} &= \widetilde{x} + \beta(x_t - x_{t-1}) \end{aligned}}$$

Recall: Momentum acceleration

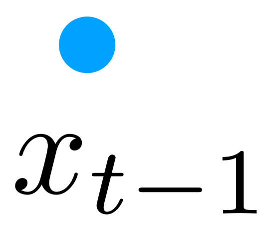
- Nesterov's work: most famous version

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

Recall: Momentum acceleration

- Nesterov's work: most famous version

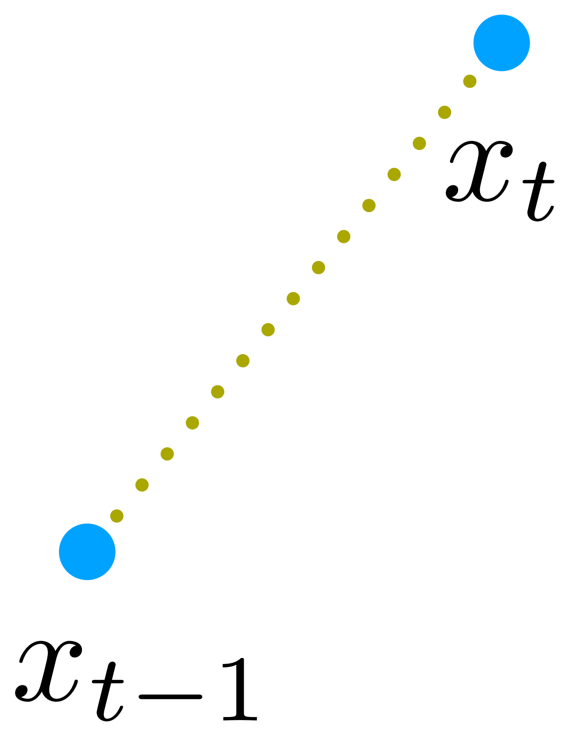
$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



Recall: Momentum acceleration

– Nesterov's work: most famous version

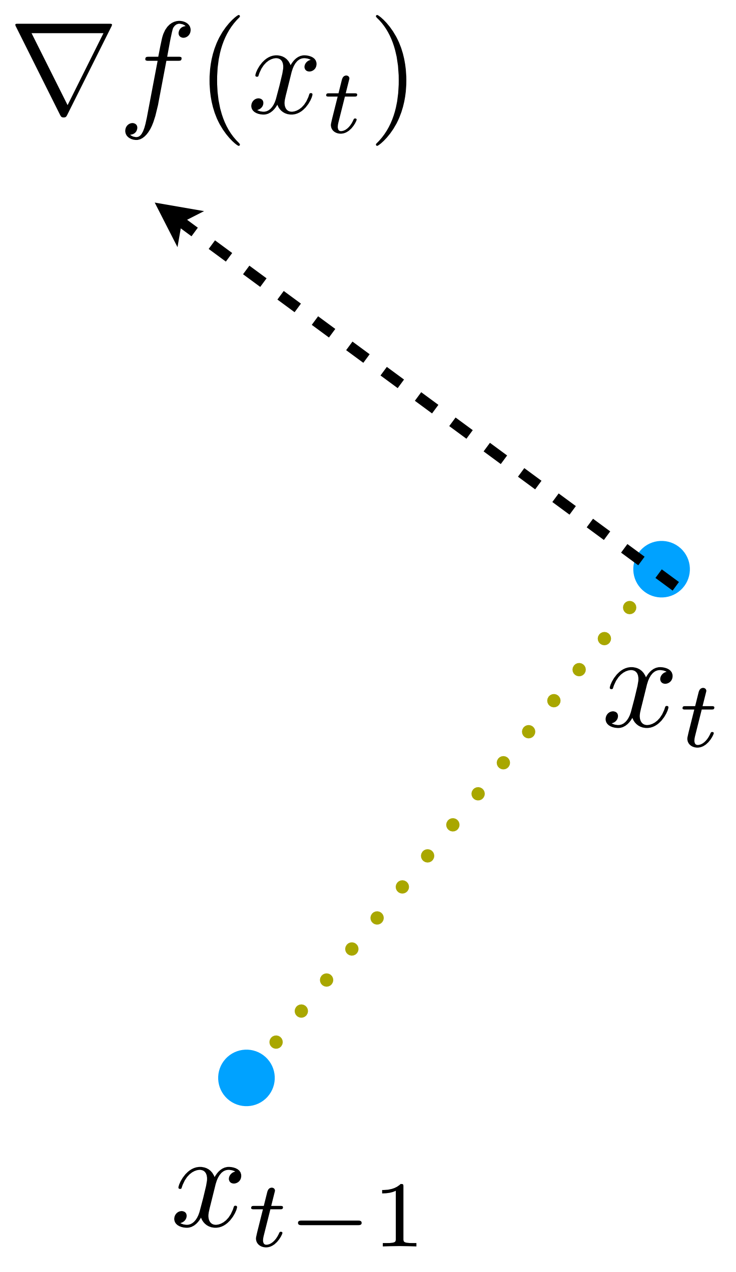
$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



Recall: Momentum acceleration

- Nesterov's work: most famous version

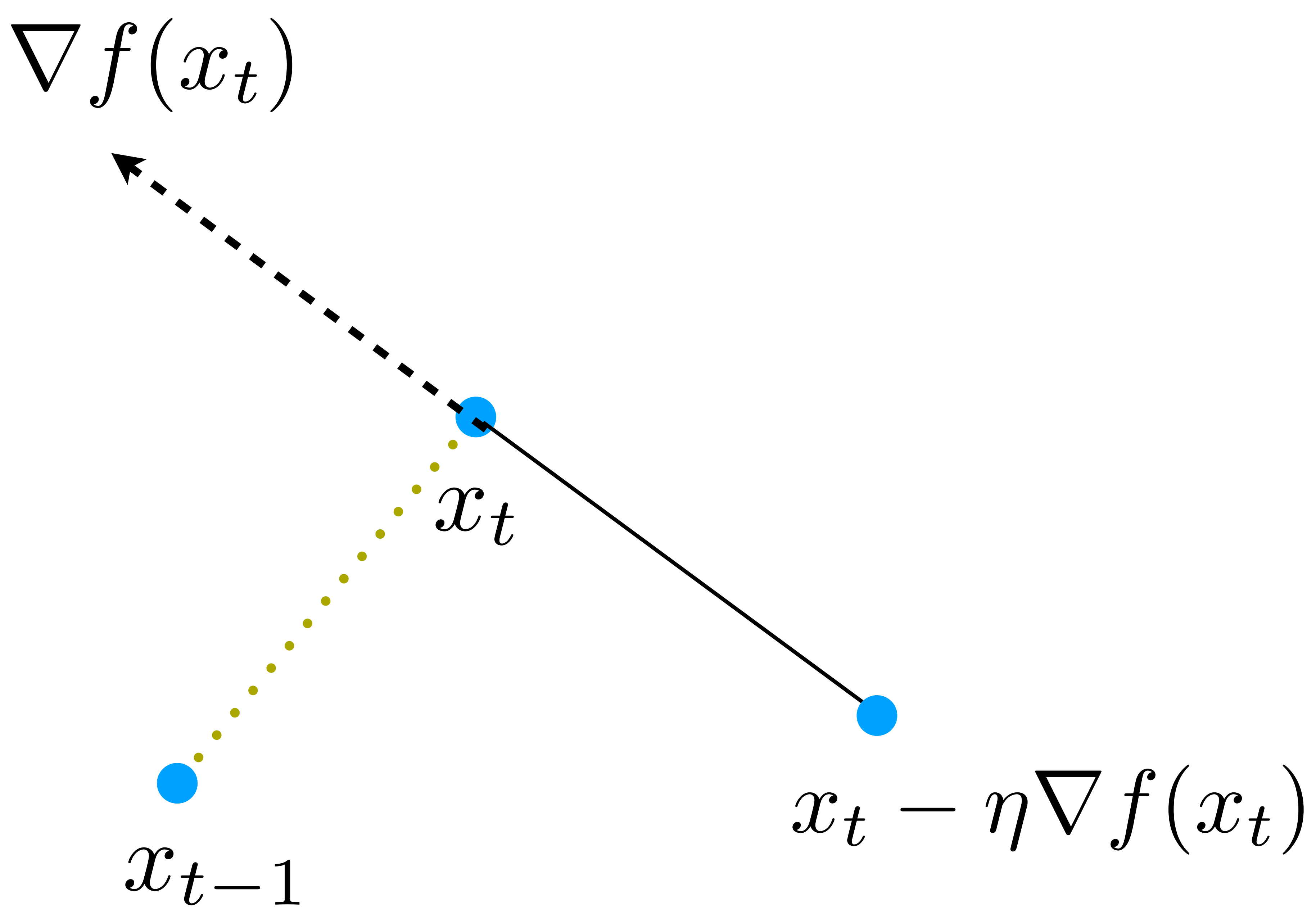
$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



Recall: Momentum acceleration

– Nesterov's work: most famous version

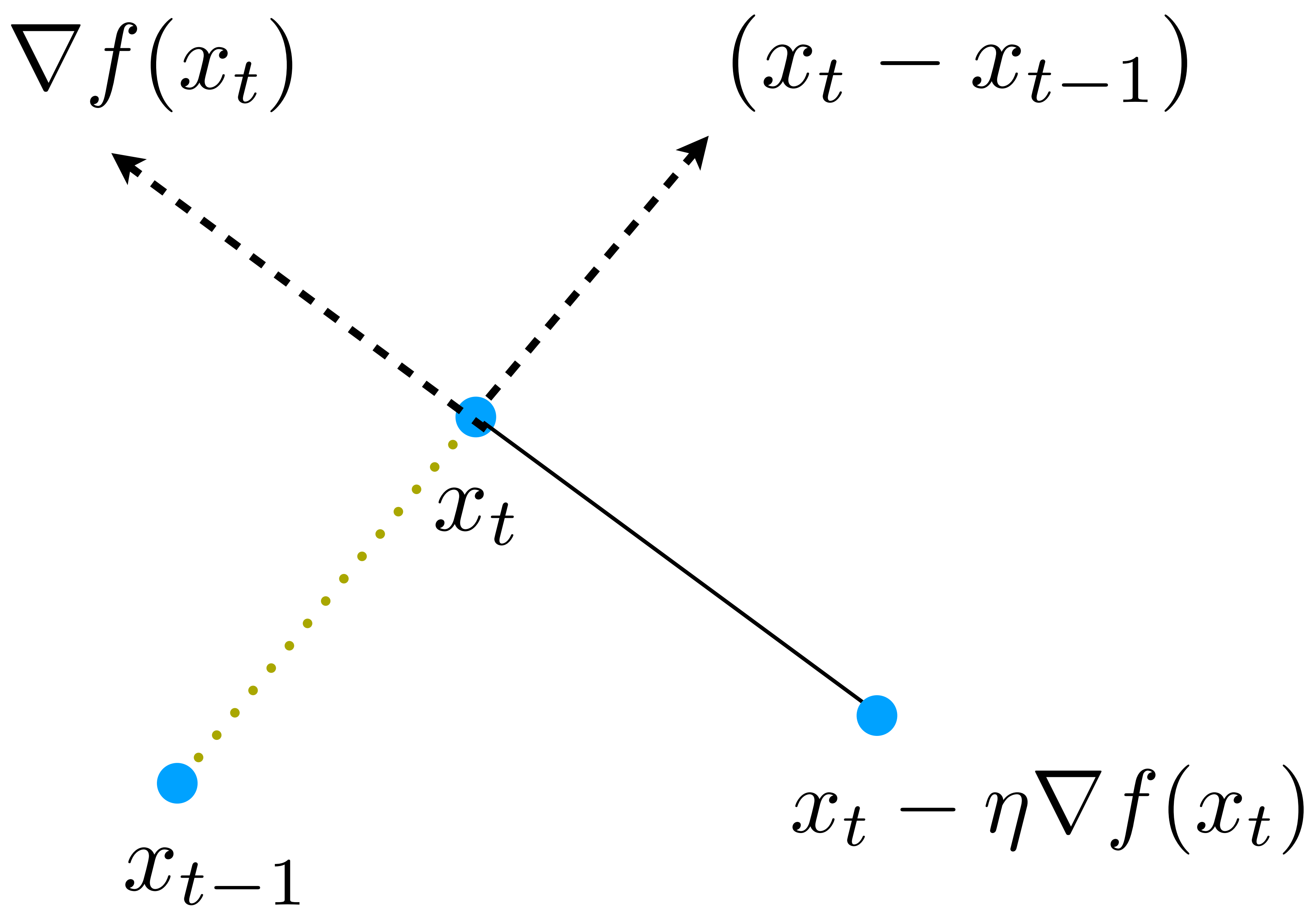
$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



Recall: Momentum acceleration

- Nesterov's work: most famous version

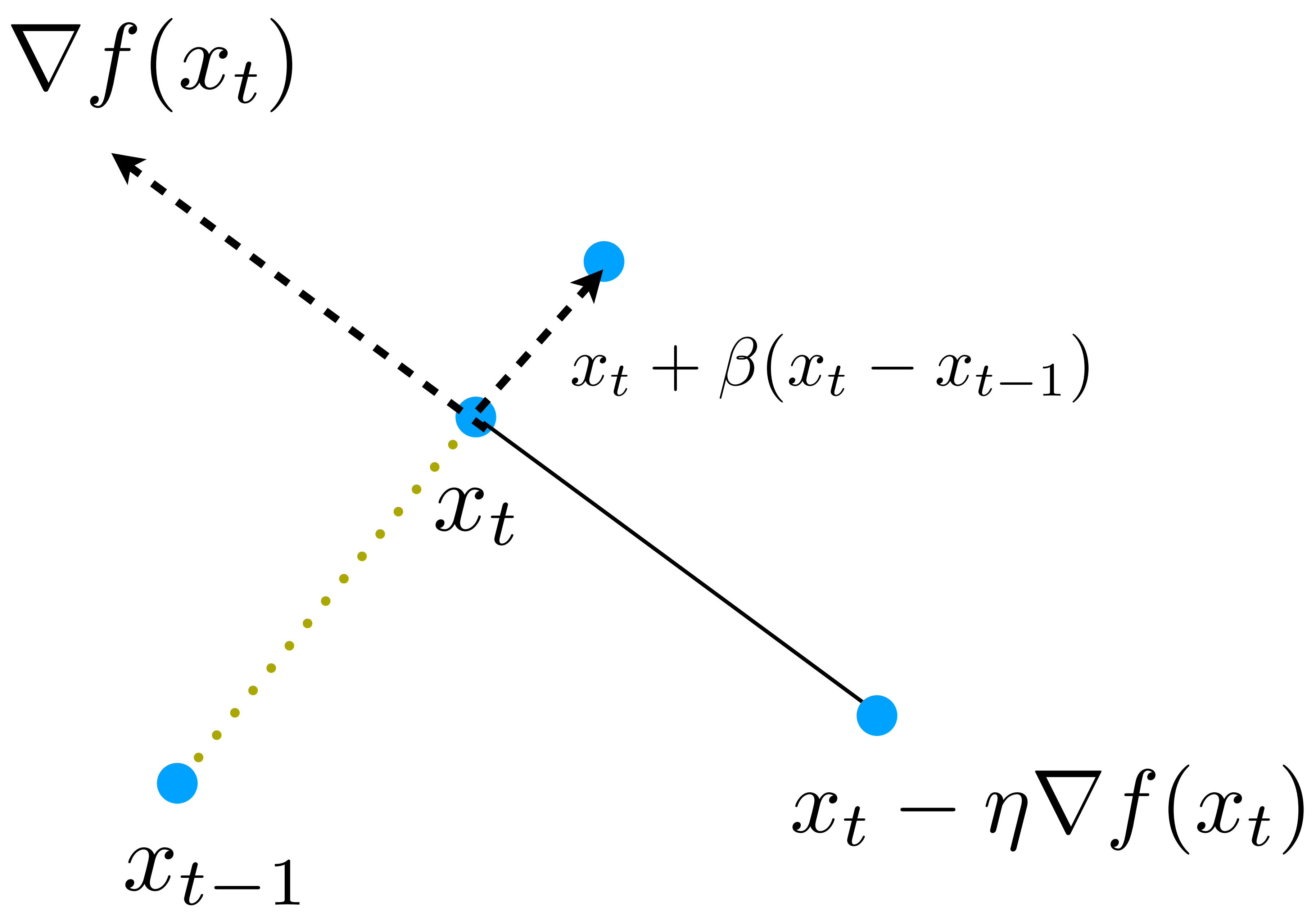
$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



Recall: Momentum acceleration

- Nesterov's work: most famous version

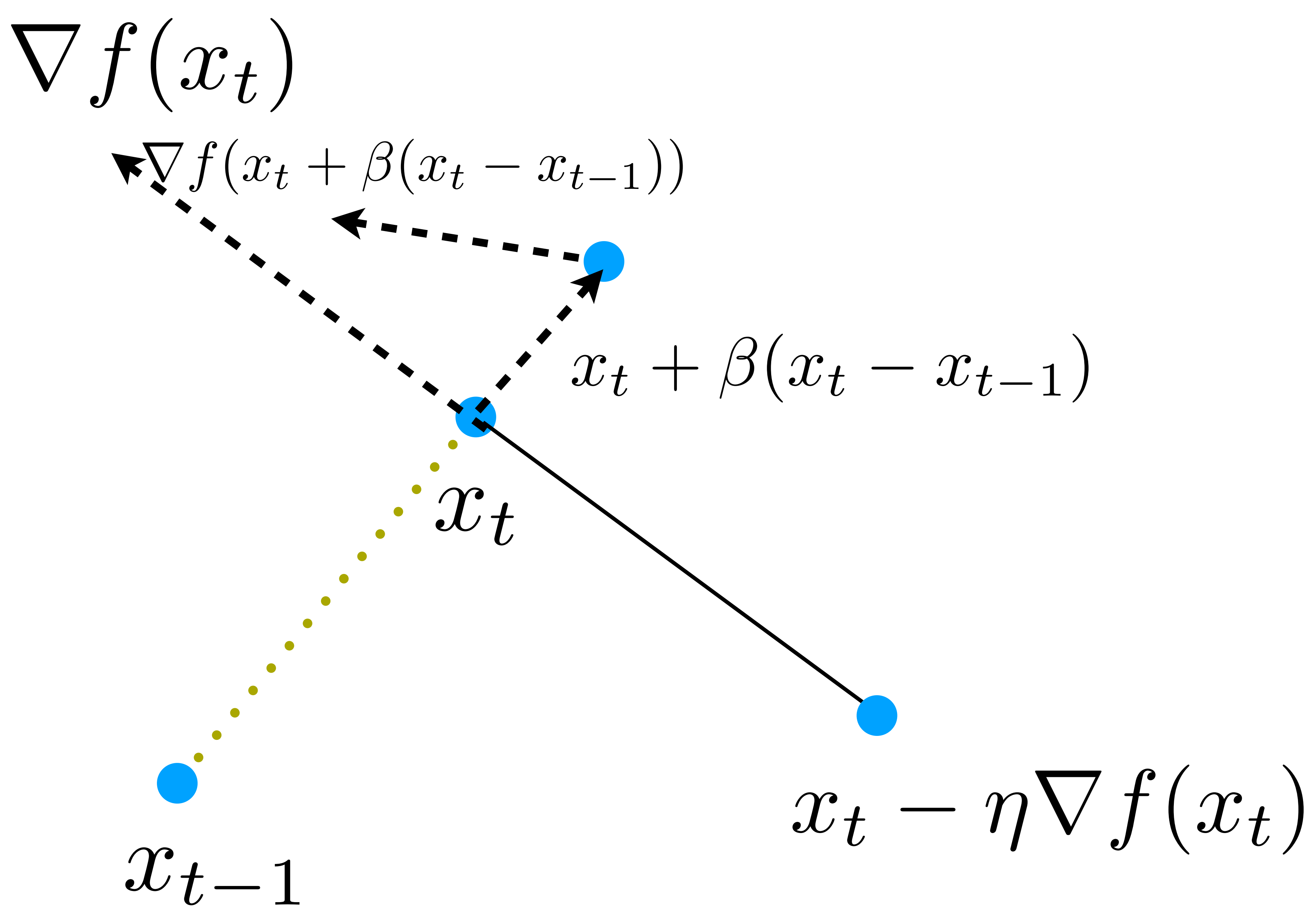
$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



Recall: Momentum acceleration

– Nesterov's work: most famous version

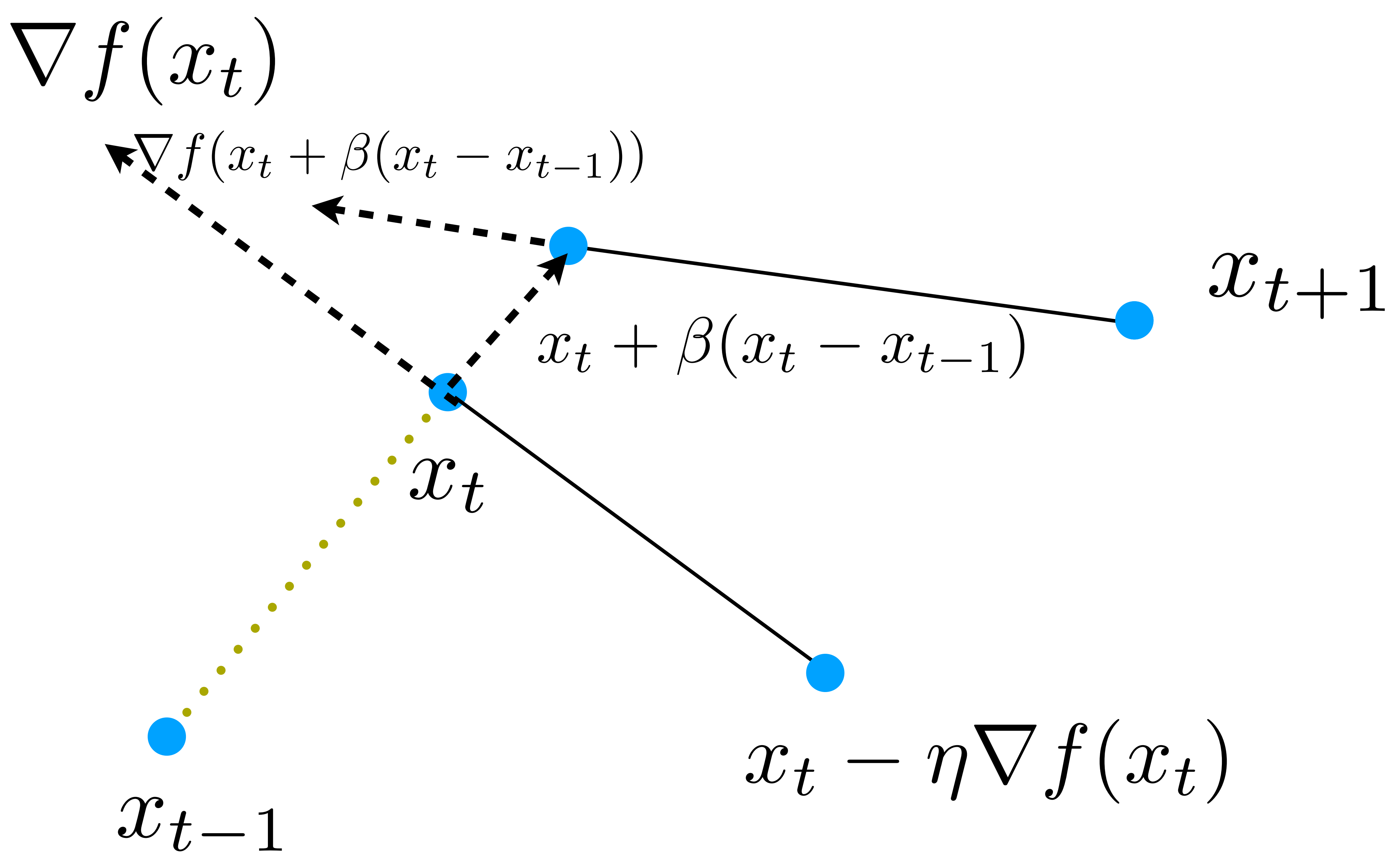
$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



Recall: Momentum acceleration

- Nesterov's work: most famous version

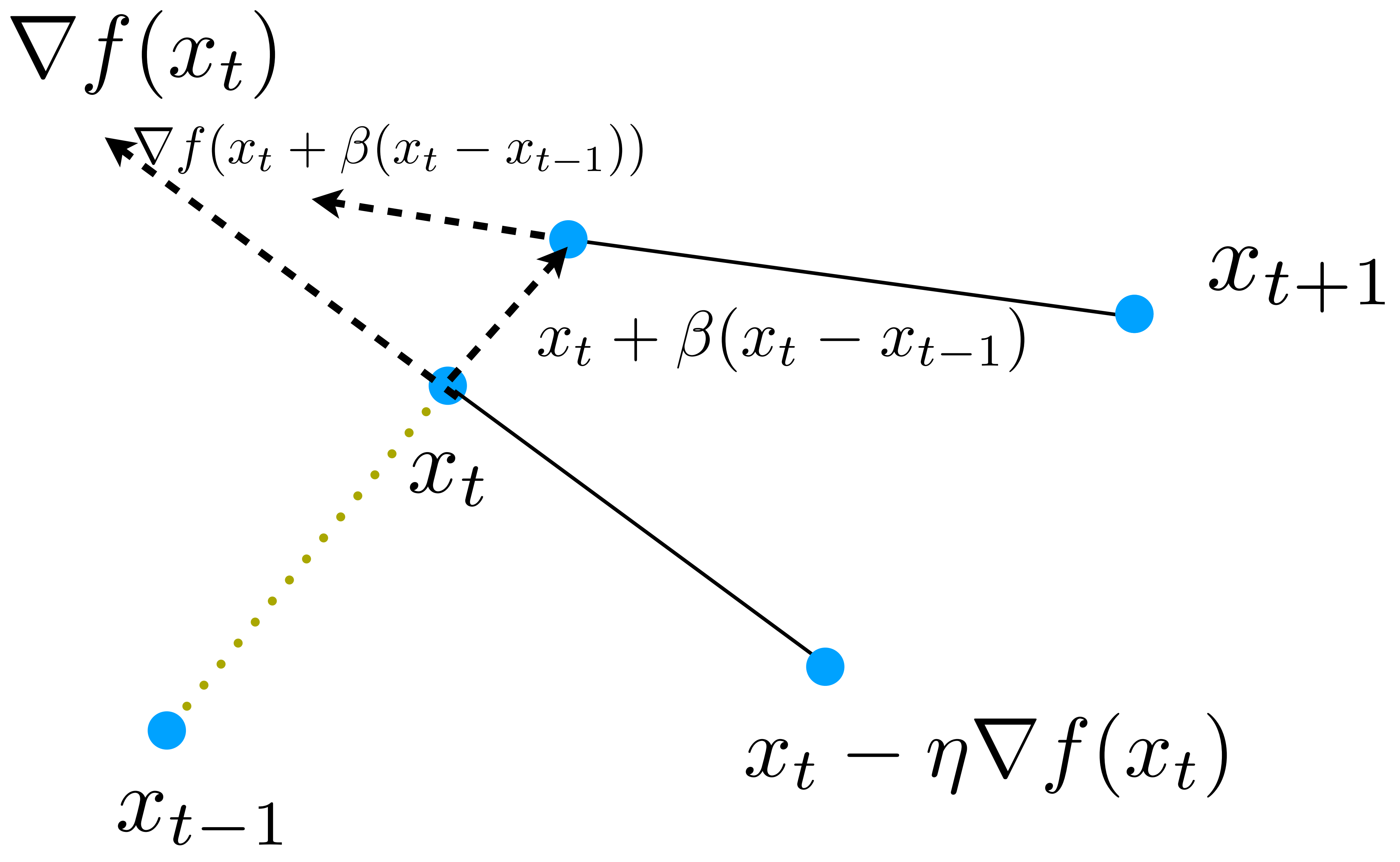
$$x_{t+1} = y_t - \eta \nabla f(y_t)$$
$$y_{t+1} = x_{t+1} + \beta(x_{t+1} - x_t)$$



Recall: Momentum acceleration

- Nesterov's work: most famous version

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

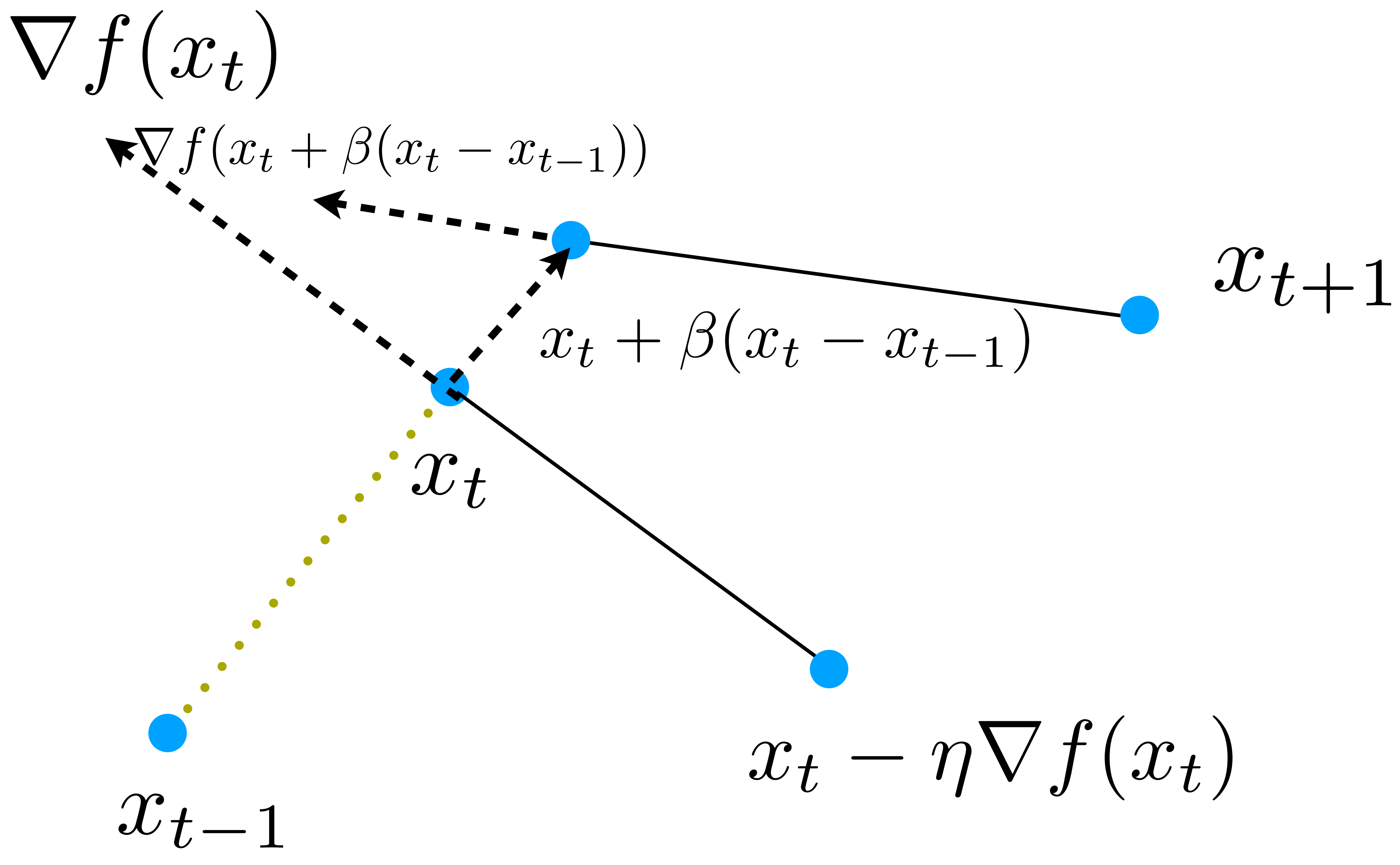


- Main difference: the point that we are calculating the gradient at.

Recall: 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

Recall: Momentum acceleration

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

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

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

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

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

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

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

3. $\beta = 0.9$

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

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

3. $\beta = 0.9$

One of the mysteries of optimization..

AdaGrad algorithm

(A Google algorithm that found application to "Large-scale distributed deep networks" paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

AdaGrad algorithm

(A Google algorithm that found application to "Large–scale distributed deep networks" paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

- AdaGrad adapts the initial step size for each of the components:
  - Associates small step sizes to frequently occurring features
  - Associates large step sizes to rate occurring features

AdaGrad algorithm

(A Google algorithm that found application to "Large-scale distributed deep networks" paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

- AdaGrad adapts the initial step size for each of the components:
  - Associates small step sizes to frequently occurring features
  - Associates large step sizes to rate occurring features

- What is the main idea? Consider $x_{t+1,i} = x_{t,i} - \eta \nabla f(x_t)_i$

Entrywise representation of GD

AdaGrad algorithm

(A Google algorithm that found application to "Large–scale distributed deep networks" paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t

- AdaGrad adapts the initial step size for each of the components:
  - Associates small step sizes to frequently occurring features
  - Associates large step sizes to rate occurring features

- What is the main idea? Consider $x_{t+1,i} = x_{t,i} - \eta \nabla f(x_t)_i$

Entrywise representation of GD

Then, practical version of AdaGrad does: $x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{B_{t,ii} + \epsilon}} \cdot \nabla f_{i_t}(x_t)_i$

AdaGrad algorithm

(A Google algorithm that found application to "Large–scale distributed deep networks" paper)

- Algorithms so far assume a common (and often fixed) step size for all components of x_t
- AdaGrad adapts the initial step size for each of the components:
  - Associates small step sizes to frequently occurring features
  - Associates large step sizes to rate occurring features
- What is the main idea? Consider $x_{t+1,i} = x_{t,i} - \eta \nabla f(x_t)_i$

Entrywise representation of GD

Then, practical version of AdaGrad does: $x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{\boxed{B_{t,ii}} + \epsilon}} \cdot \nabla f_{i_t}(x_t)_i$

What is this quantity?

AdaGrad algorithm

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

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

Recall: Preconditioning algorithms (BFGS, SR1) in lecture 3

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

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

Recall: Preconditioning algorithms (BFGS, SR1) in lecture 3

where

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

"Square root of the sum of gradient outer products, till current iteration"

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

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

Recall: Preconditioning algorithms (BFGS, SR1) in lecture 3

where

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

"Square root of the sum of gradient outer products, till current iteration"

Full matrix AdaGrad

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

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

Recall: Preconditioning algorithms (BFGS, SR1) in lecture 3

where

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

"Square root of the sum of gradient outer products, till current iteration"

Full matrix AdaGrad

- Compare this to the simpler (and practical version)

$$x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{B_{t,ii} + \epsilon}} \cdot \nabla f_{i_t}(x_t)_i$$

AdaGrad algorithm

- AdaGrad is just another preconditioning algorithm:

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

Recall: Preconditioning algorithms (BFGS, SR1) in lecture 3

where

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

"Square root of the sum of gradient outer products, till current iteration"

Full matrix AdaGrad

- Compare this to the simpler (and practical version)

$$x_{t+1,i} = x_{t,i} - \frac{\eta}{\sqrt{B_{t,ii} + \boxed{\epsilon}}} \cdot \nabla f_{i_t}(x_t)_i$$

Avoids division with zero

AdaGrad algorithm

- "What is the intuition behind the form of B_t ?"

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

Relates to the **Fisher Information matrix** (which is related to the expected Hessian) – outside our scope

AdaGrad algorithm

- "What is the intuition behind the form of B_t ?"

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

Relates to the **Fisher Information matrix** (which is related to the expected Hessian) – outside our scope

- "What is the connection between full and diagonal preconditioner?"

Whiteboard

AdaGrad algorithm

– "What is the intuition behind the form of B_t ?"

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

Relates to the **Fisher Information matrix** (which is related to the expected Hessian) – outside our scope

– "What is the connection between full and diagonal preconditioner?"

Whiteboard

– "What are some properties of AdaGrad?"

1. Step size is automatically set – default values for initial step size is $\eta = 0.01$
2. The original version keeps accumulating squared gradients, which makes resulting step sizes really small.

AdaGrad algorithm

- ``What is the intuition behind the form of $B_t$ ?``

$$B_t = \left(\sum_{j=1}^{t} \nabla f_{i_j}(x_j) \cdot \nabla f_{i_j}(x_j)^\top \right)^{1/2}$$

Relates to the **Fisher Information matrix** (which is related to the expected Hessian) – outside our scope

- ``What is the connection between full and diagonal preconditioner?``

Whiteboard

- ``What are some properties of AdaGrad?``

1. Step size is automatically set – default values for initial step size is $\eta = 0.01$
2. The original version keeps accumulating squared gradients, which makes resulting step sizes really small.

- ``Are there guarantees for AdaGrad?``
- Yes, in the convex case, using regret bounds – see Literature section

AdaGrad pseudocode

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

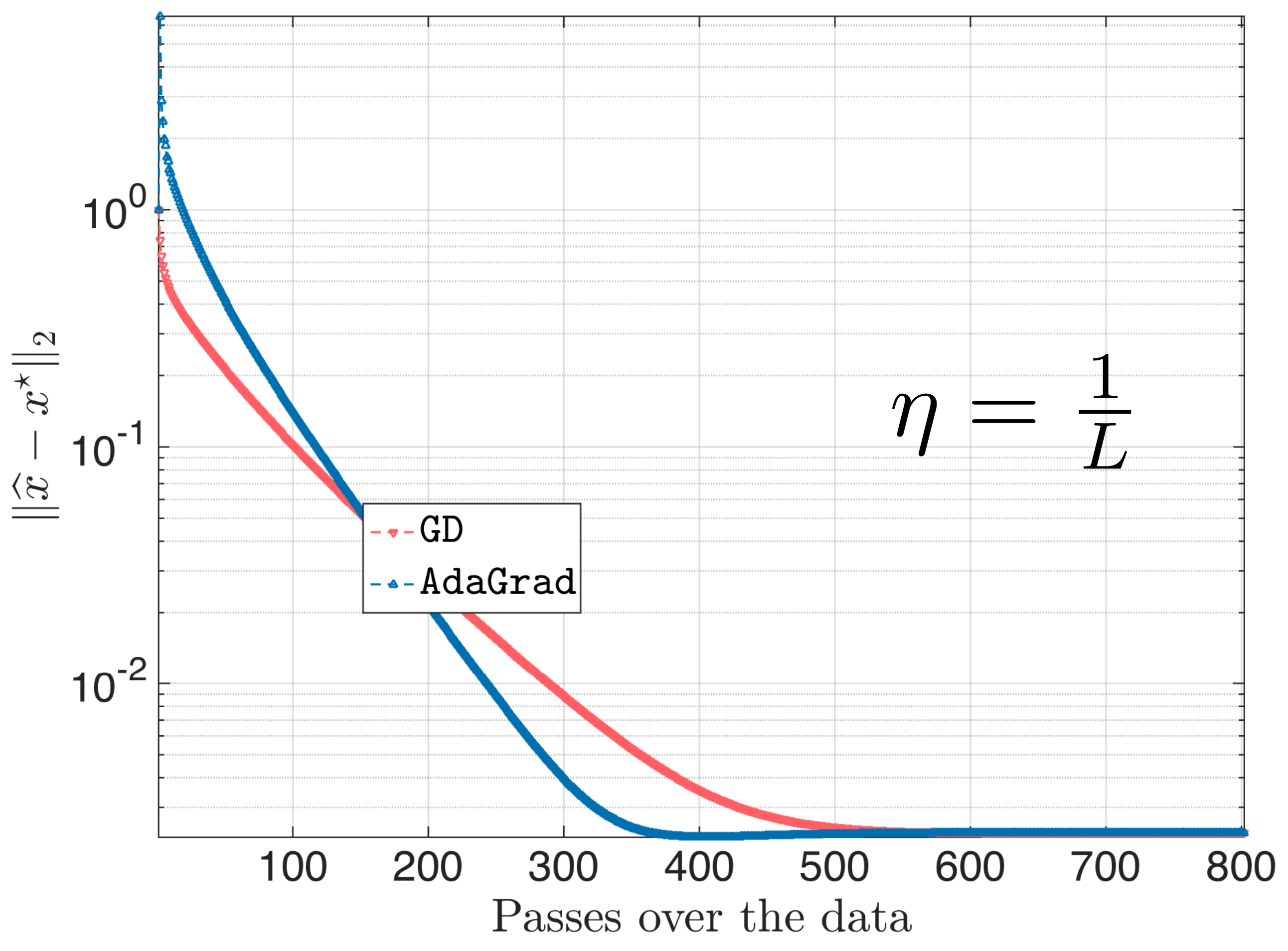
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

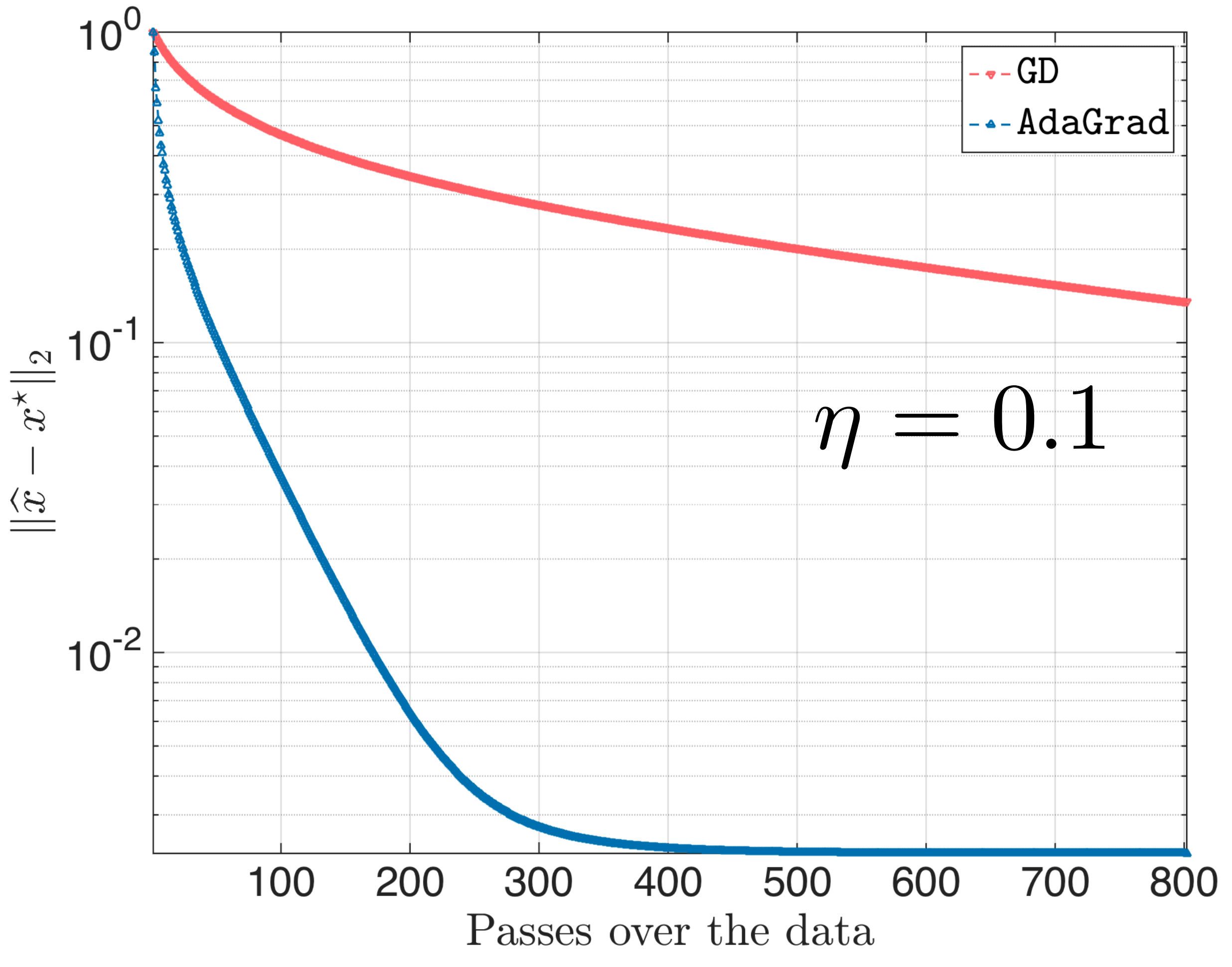
end while

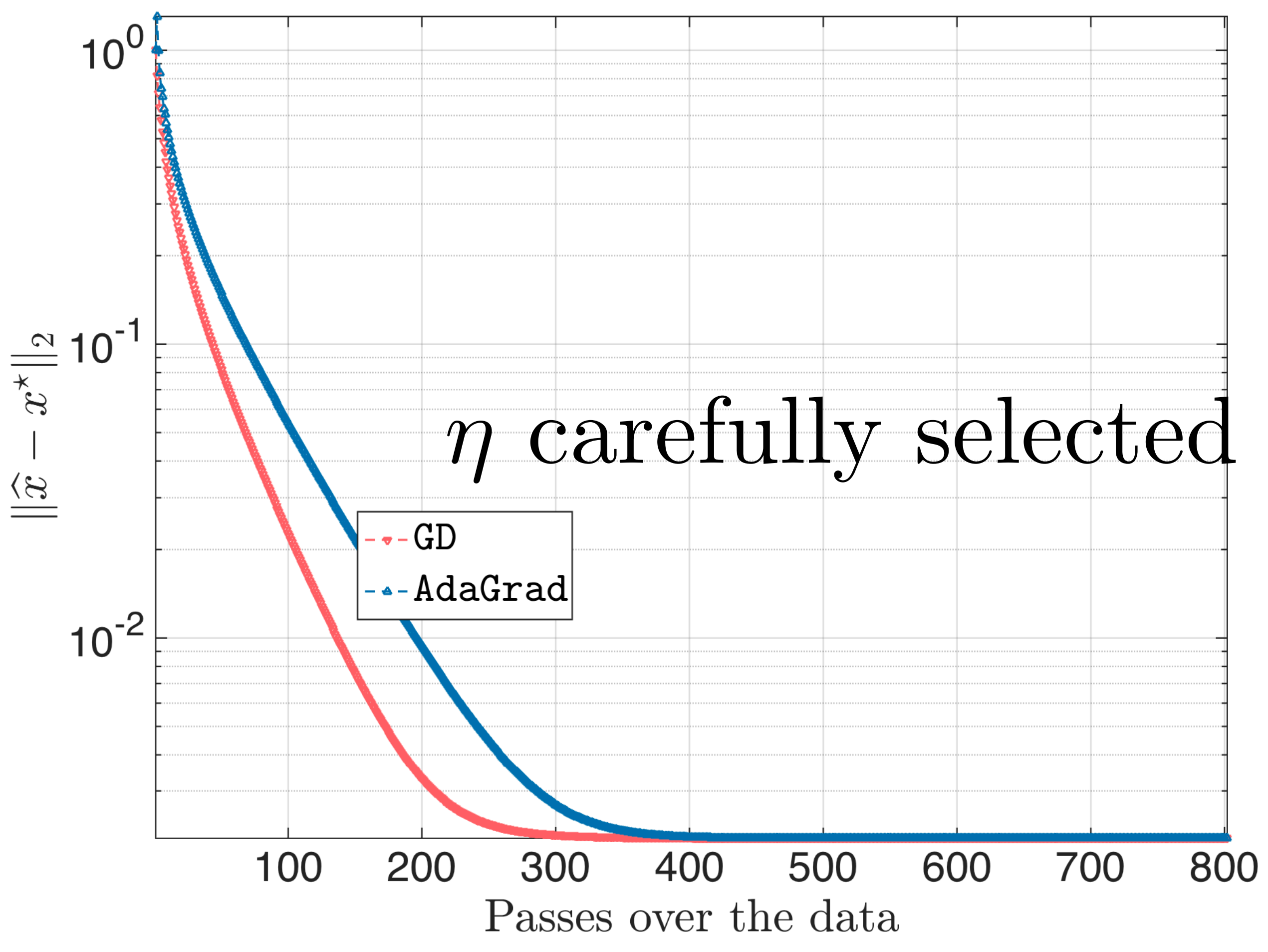
AdaGrad in practice

AdaGrad in practice

Well-conditioned linear regression

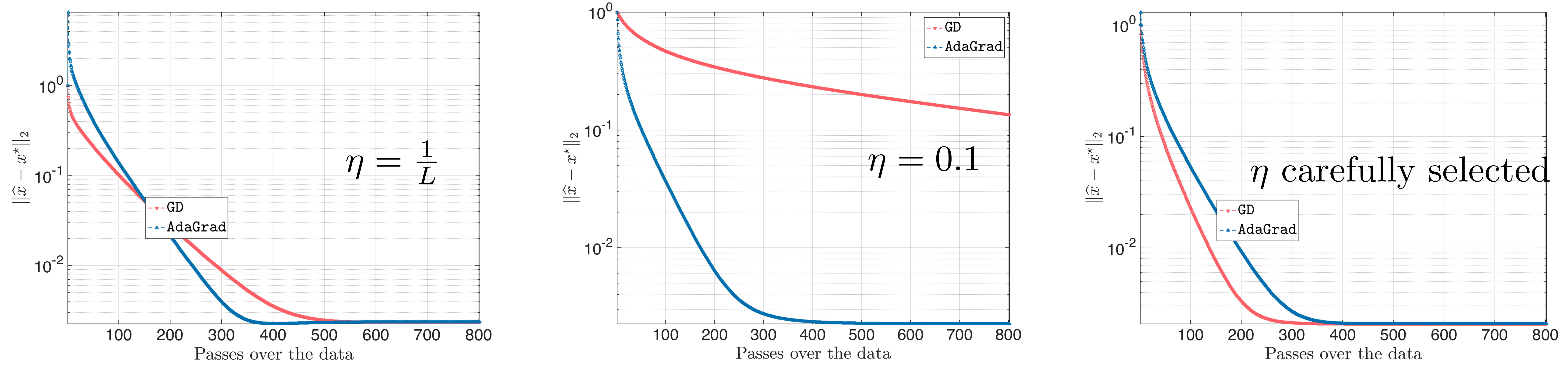




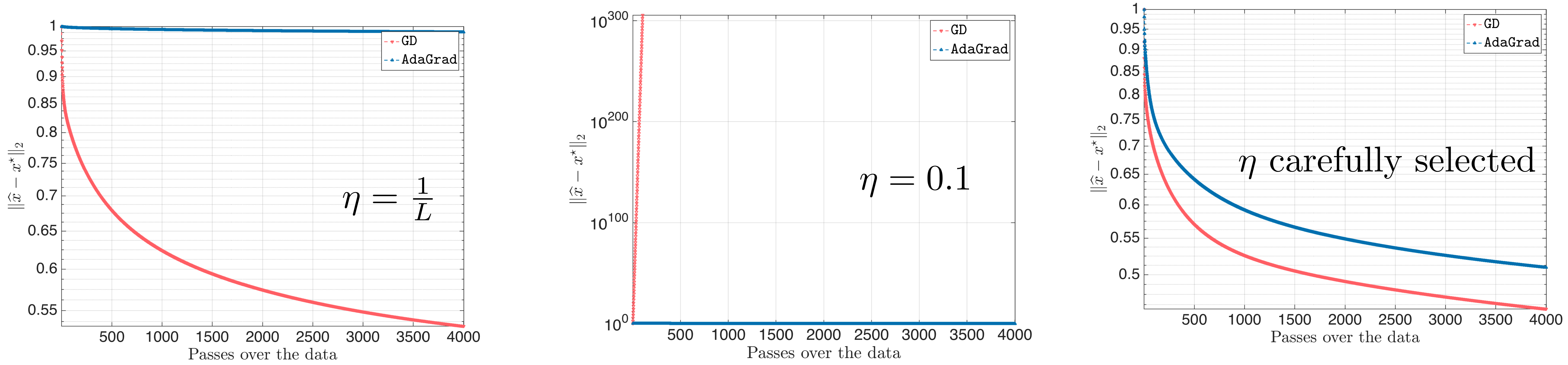


AdaGrad in practice

Well−conditioned linear regression



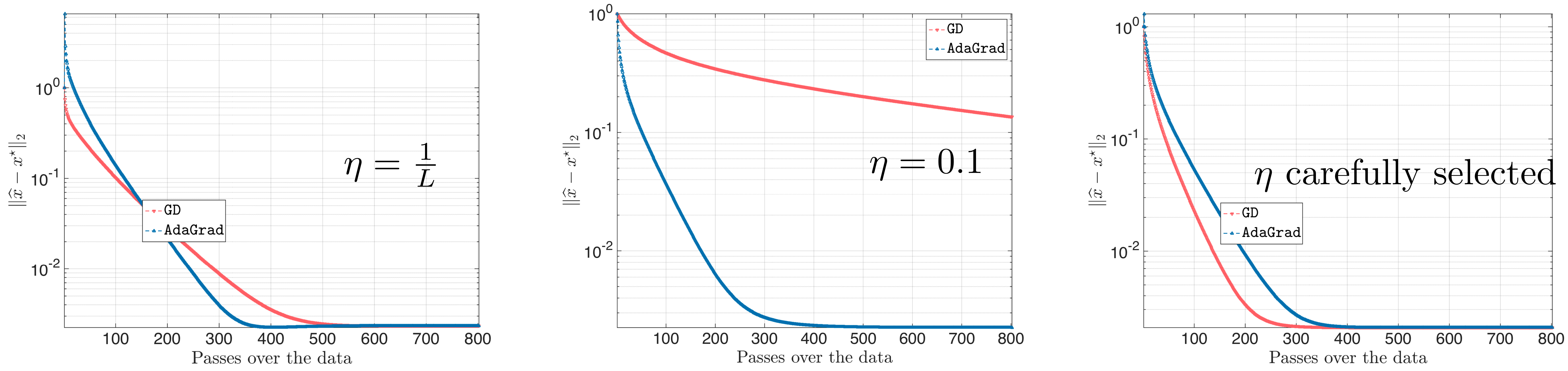
Ill−conditioned linear regression



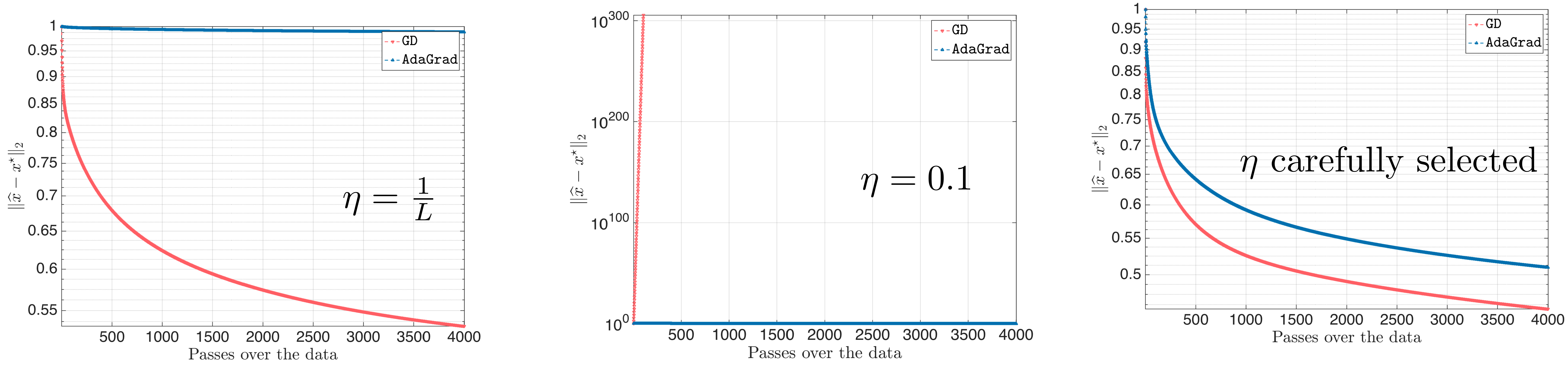
AdaGrad in practice

(Similar performance in logistic regression)

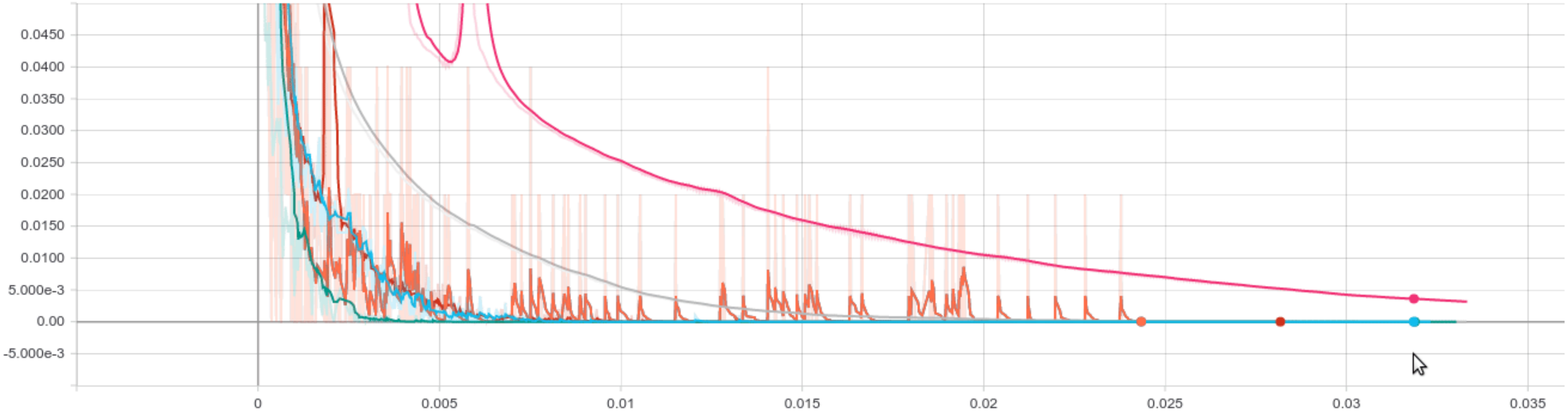
Well-conditioned linear regression



Ill-conditioned linear regression

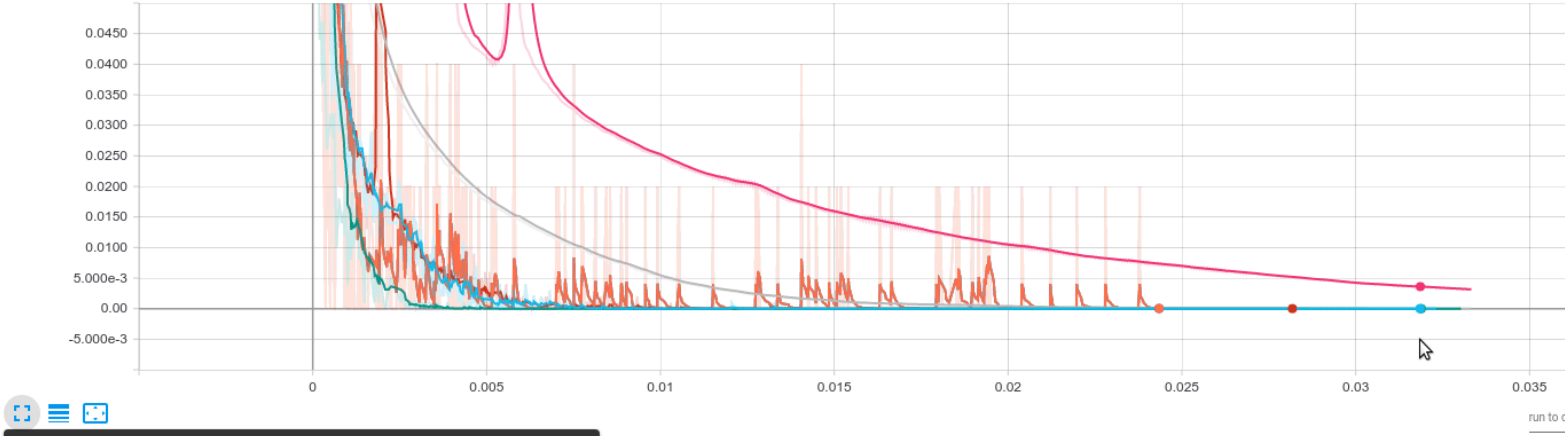


AdaGrad in practice



Name	Smoothed	Value	Step	Time	Relative
AdaGrad2	-1.1921e-7	-1.1921e-7	5.381k	Sun Dec 10, 15:17:05	1m 54s
AdaGrad3	3.6024e-3	3.5634e-3	764.0	Sun Dec 10, 15:19:02	1m 54s
Nesterov2	-1.1921e-7	-1.1921e-7	5.303k	Sun Dec 10, 15:28:30	1m 54s
Nesterov3	-2.3665e-7	-2.3842e-7	764.0	Sun Dec 10, 15:30:29	1m 54s
SGD1	2.1986e-5	0.000	5.486k	Sun Dec 10, 15:01:31	1m 27s
SGD1/.	2.1986e-5	0.000	5.486k	Sun Dec 10, 15:01:31	1m 27s
SGD2	-1.1921e-7	-1.1921e-7	5.486k	Sun Dec 10, 15:03:13	1m 41s

AdaGrad in practice



	Name	Smoothed	Value	Step	Time	Relative
	AdaGrad2	-1.1921e-7	-1.1921e-7	5.381k	Sun Dec 10, 15:17:05	1m 54s
	AdaGrad3	3.6024e-3	3.5634e-3	764.0	Sun Dec 10, 15:19:02	1m 54s
	Nesterov2	-1.1921e-7	-1.1921e-7	5.303k	Sun Dec 10, 15:28:30	1m 54s
	Nesterov3	-2.3665e-7	-2.3842e-7	764.0	Sun Dec 10, 15:30:29	1m 54s
	SGD1	2.1986e-5	0.000	5.486k	Sun Dec 10, 15:01:31	1m 27s
	SGD1/.	2.1986e-5	0.000	5.486k	Sun Dec 10, 15:01:31	1m 27s
	SGD2	-1.1921e-7	-1.1921e-7	5.486k	Sun Dec 10, 15:03:13	1m 41s

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

  + Diagonal AdaGrad rule: $\mathrm{diag}(B_t) = \mathrm{diag}(B_{t-1}) + \mathrm{diag}\left(\nabla f_{i_t}(x_t) \circ \nabla f_{i_t}(x_t)\right)$

Removing extended gradient accumulation: **RMSprop** algorithm

– Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

+ Diagonal AdaGrad rule: $\underbrace{\mathrm{diag}(B_t)}_{E[g^2]_t} = \underbrace{\mathrm{diag}(B_{t-1})}_{E[g^2]_{t-1}} + \underbrace{\mathrm{diag}\left(\nabla f_{i_t}(x_t) \circ \nabla f_{i_t}(x_t)\right)}_{g_t^2}$

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation

  + Diagonal AdaGrad rule: $\underbrace{\text{diag}(B_t)}_{E[g^2]_t} = \underbrace{\text{diag}(B_{t-1})}_{E[g^2]_{t-1}} + \underbrace{\text{diag}\left(\nabla f_{i_t}(x_t) \circ \nabla f_{i_t}(x_t)\right)}_{g_t^2}$

  + RMSprop rule: $E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$

“We always give weight 0.1 to the new information”

Removing extended gradient accumulation: **RMSprop** algorithm

- Idea: keep AdaGrad as it is; except, use a weighted moving average for gradient accumulation
  - Diagonal AdaGrad rule: $\underbrace{\text{diag}(B_t)}_{E[g^2]_t} = \underbrace{\text{diag}(B_{t-1})}_{E[g^2]_{t-1}} + \underbrace{\text{diag}\left(\nabla f_{i_t}(x_t) \circ \nabla f_{i_t}(x_t)\right)}_{g_t^2}$
  - RMSprop rule: $E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$

``We always give weight 0.1 to the new information``

- Algorithm:

$$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$
$$x_{t+1} = x_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla f_{i_t}(x_t)$$

Introducing exponentially weighted averages

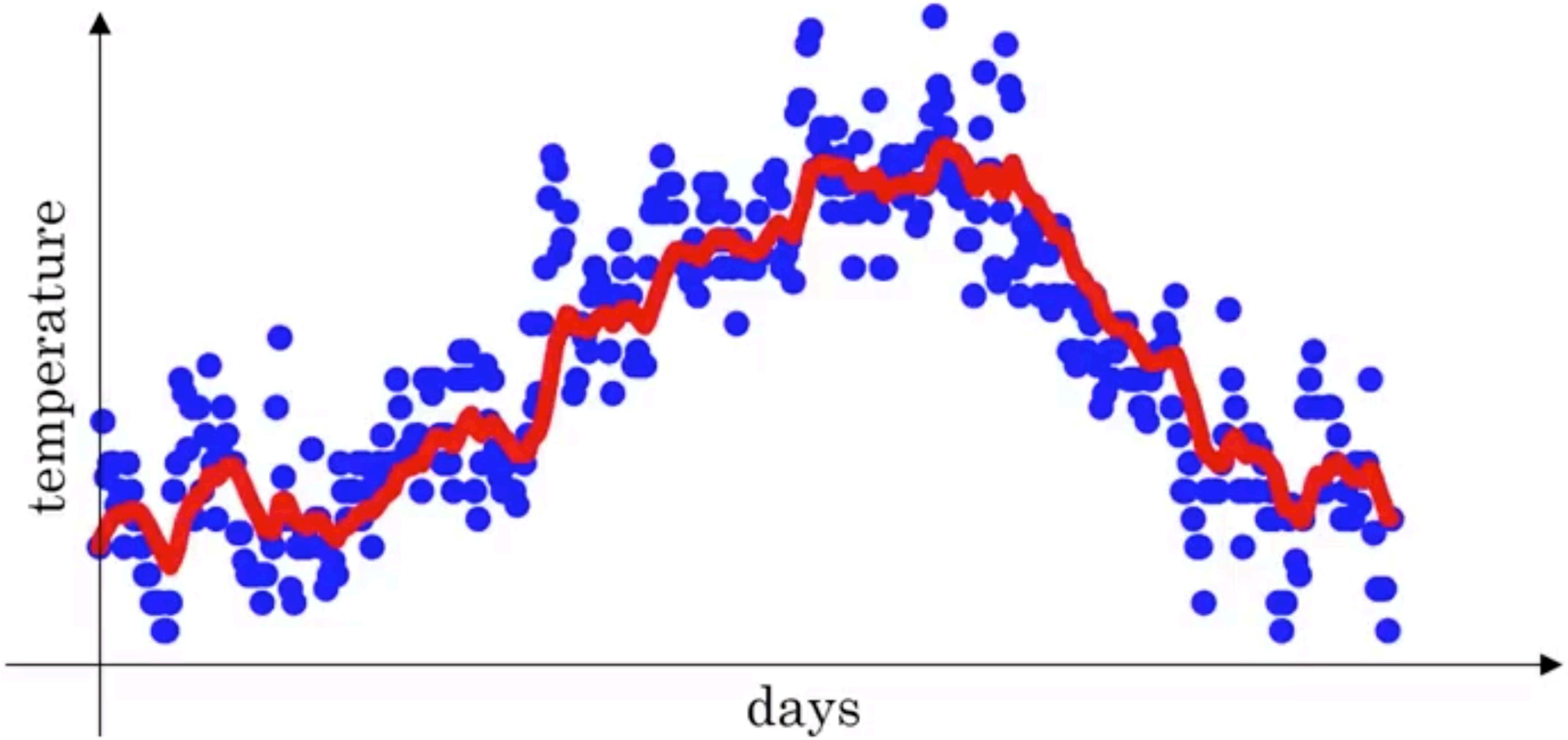
(Adapted from Ng's lectures)

- Toy example: temperature values over a year

Introducing exponentially weighted averages

(Adapted from Ng's lectures)

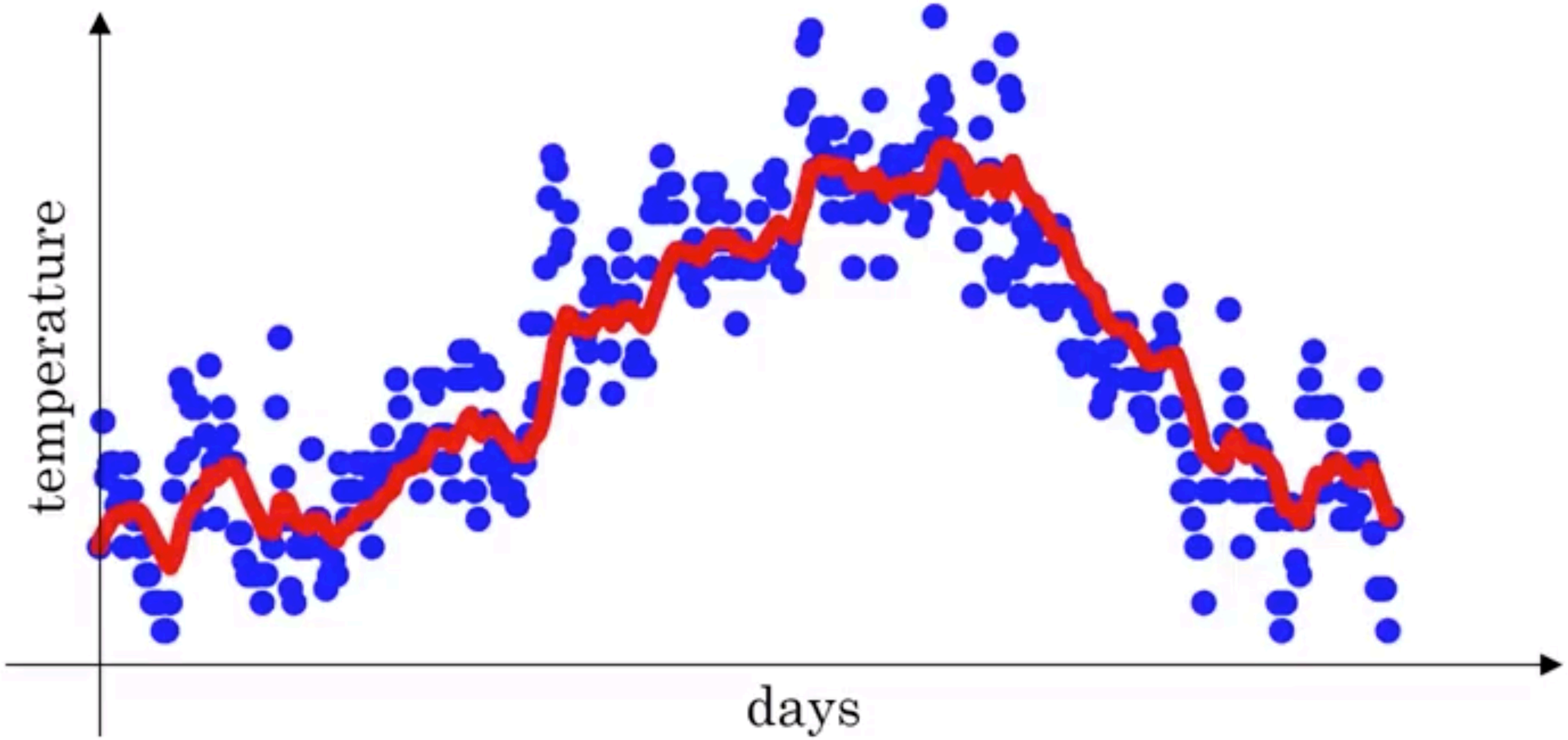
- Toy example: temperature values over a year



Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year



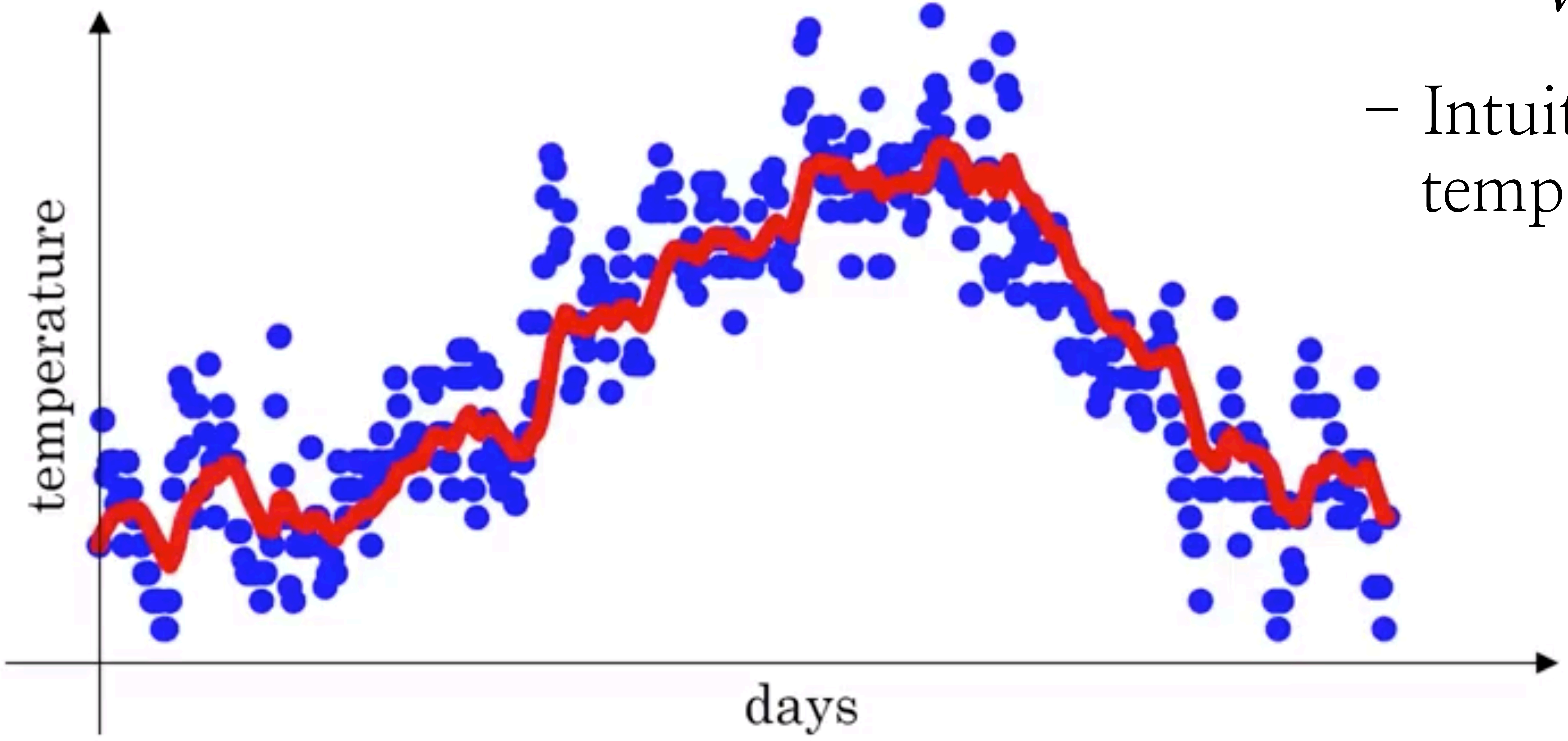
- Computing trends: local averages and how they evolve

$$V_0 = 0$$
$$V_1 = 0.9V_0 + 0.1\theta_1$$
$$V_2 = 0.9V_1 + 0.1\theta_2$$
$$\vdots$$
$$V_t = 0.9V_{t-1} + 0.1\theta_t$$

Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year



- General formula:

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

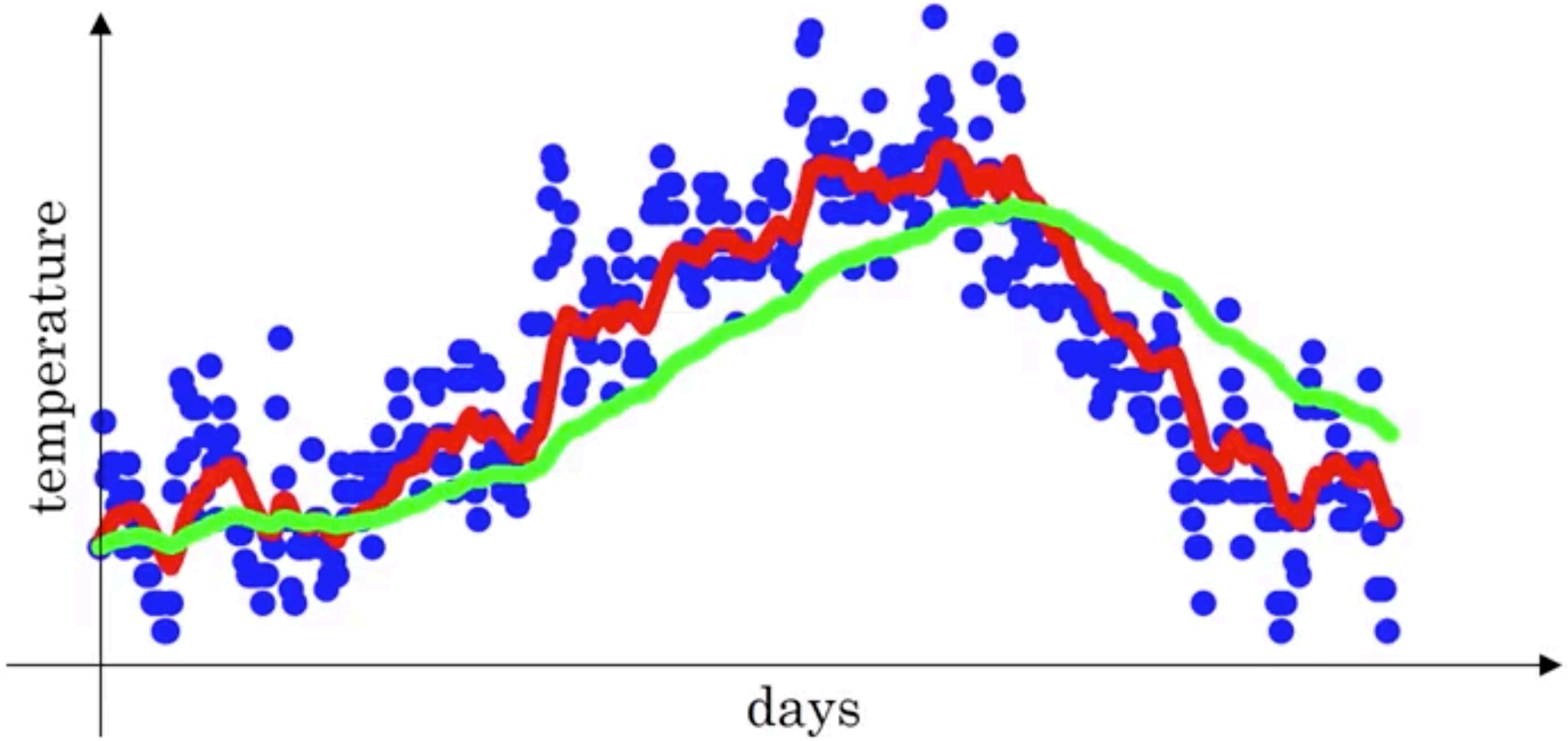
- Intuition: V_t approximates temperature over

$$\approx \frac{1}{1-\beta} \text{ days}$$

Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year



- Examples:

$\beta = 0.9 \rightarrow \quad \approx 10$ days

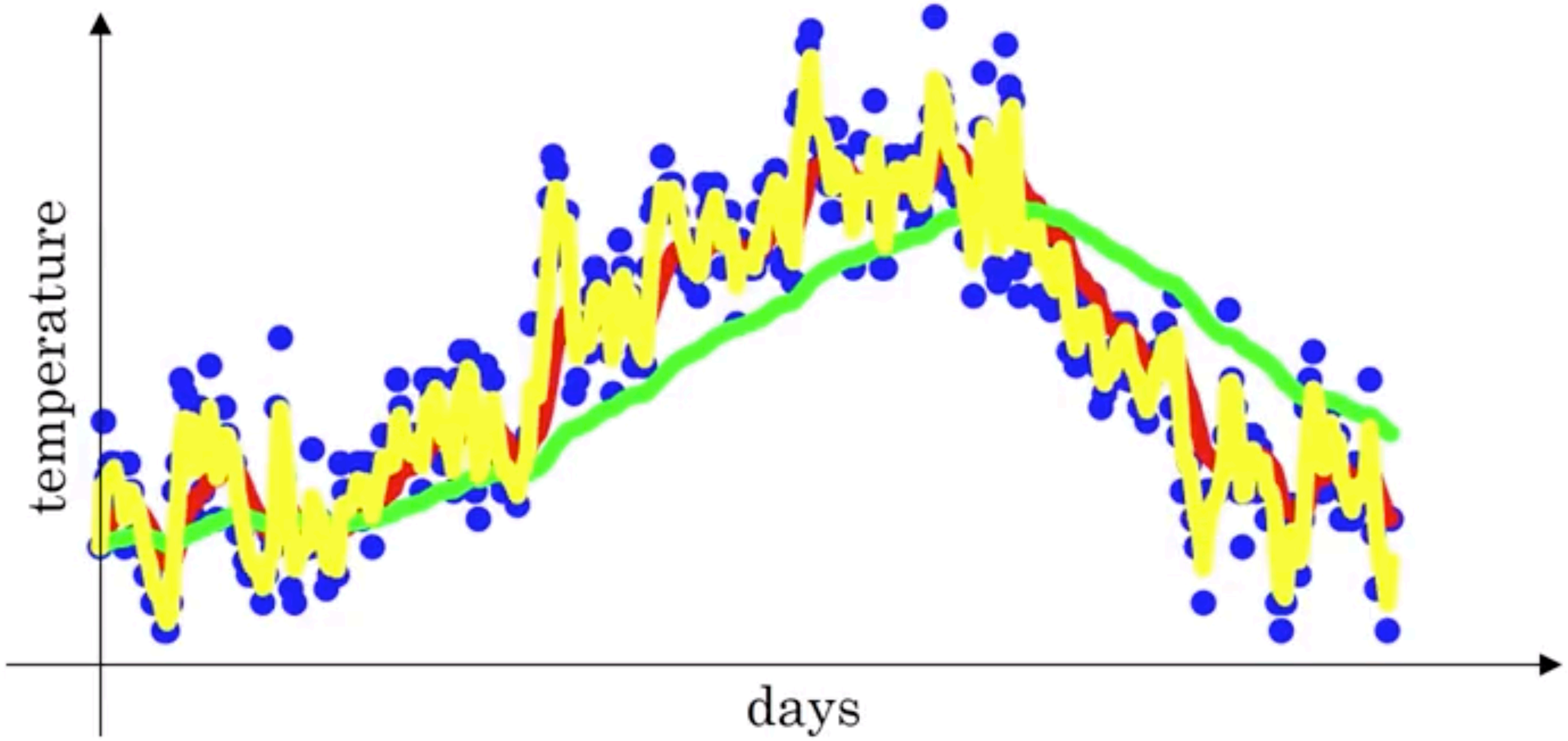
$\beta = 0.98 \rightarrow \quad \approx 50$ days

$\beta = 0.5 \rightarrow \quad \approx 2$ days

Introducing exponentially weighted averages

(Adapted from Ng's lectures)

- Toy example: temperature values over a year



- Examples:

  $\beta = 0.9 \rightarrow \quad \approx 10$ days

  $\beta = 0.98 \rightarrow \quad \approx 50$ days

  $\beta = 0.5 \rightarrow \quad \approx 2$ days

Going beyond RMSprop: **Adam** algorithm

– Idea: Use weighted moving average in gradient also:

Going beyond RMSprop: **Adam** algorithm

– Idea: Use weighted moving average in gradient also:

+ **RMSprop** rule: $E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$

Going beyond RMSprop: **Adam** algorithm

- Idea: Use weighted moving average in gradient also:
  + **RMSprop** rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$
  + **Adam** rule: $$E[g^2]_t = \beta_2 \cdot E[g^2]_{t-1} + (1 - \beta_2) \cdot g_t^2$$

"Moving averages are essentially about averaging many previous values in order to become independent of local fluctuations and focus on the overall trend"

and

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla f_{i_t}(x_t)$$

Going beyond RMSprop: **Adam** algorithm

– Idea: Use weighted moving average in gradient also:

+ **RMSprop** rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

+ **Adam** rule: $$E[g^2]_t = \beta_2 \cdot E[g^2]_{t-1} + (1-\beta_2) \cdot g_t^2$$

"Moving averages are essentially about averaging many previous values in order to become independent of local fluctuations and focus on the overall trend"

and

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \nabla f_{i_t}(x_t)$$

Further: $$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \widehat{v}_t = \frac{E[g^2]_t}{1-\beta_2^t}$$

Going beyond RMSprop: **Adam** algorithm

- Idea: Use weighted moving average in gradient also:

  + **RMSprop** rule: $$E[g^2]_t = \frac{9}{10} \cdot E[g^2]_{t-1} + \frac{1}{10} \cdot g_t^2$$

  + **Adam** rule: $$E[g^2]_t = \beta_2 \cdot E[g^2]_{t-1} + (1 - \beta_2) \cdot g_t^2$$

"Moving averages are essentially about averaging many previous values in order to become independent of local fluctuations and focus on the overall trend"

and

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla f_{i_t}(x_t)$$

Further: $$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \widehat{v}_t = \frac{E[g^2]_t}{1 - \beta_2^t}$$

- Algorithm: $$x_{t+1} = x_t - \frac{\eta}{\sqrt{\widehat{v}_t} + \epsilon} \cdot \widehat{m}_t$$

$\beta_1 = 0.9, \;\; \beta_2 = 0.999$

Bias correction in weighted averages

(Adapted from Ng's lectures)

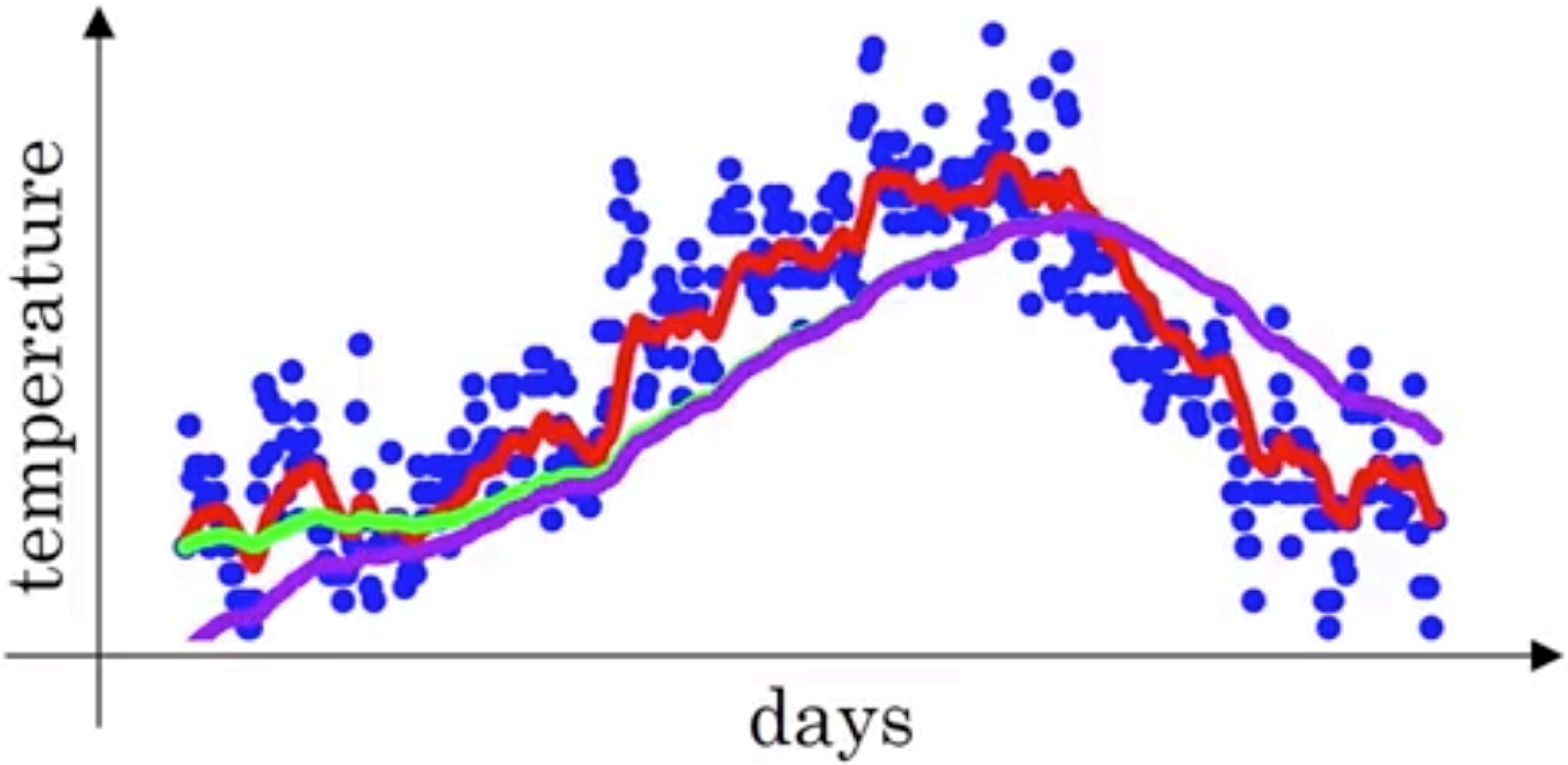
- How to explain these ``weird`` denominators?

Bias correction in weighted averages

(Adapted from Ng's lectures)

- How to explain these "weird" denominators?

$$V_t = \beta V_{t-1} + (1 - \beta)\theta_t$$



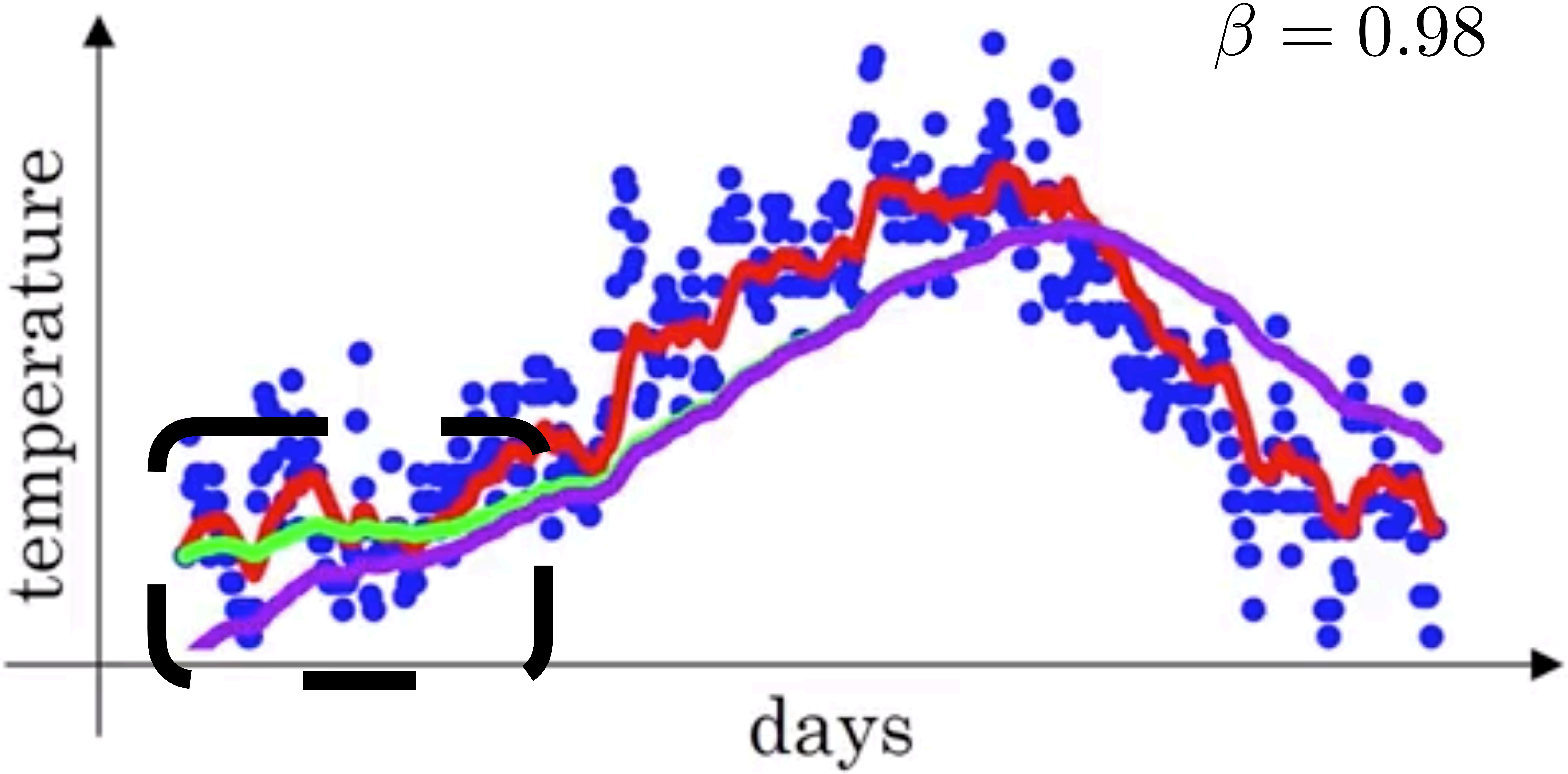
Bias correction in weighted averages

(Adapted from Ng's lectures)

- How to explain these ``weird`` denominators?

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$\beta = 0.98$

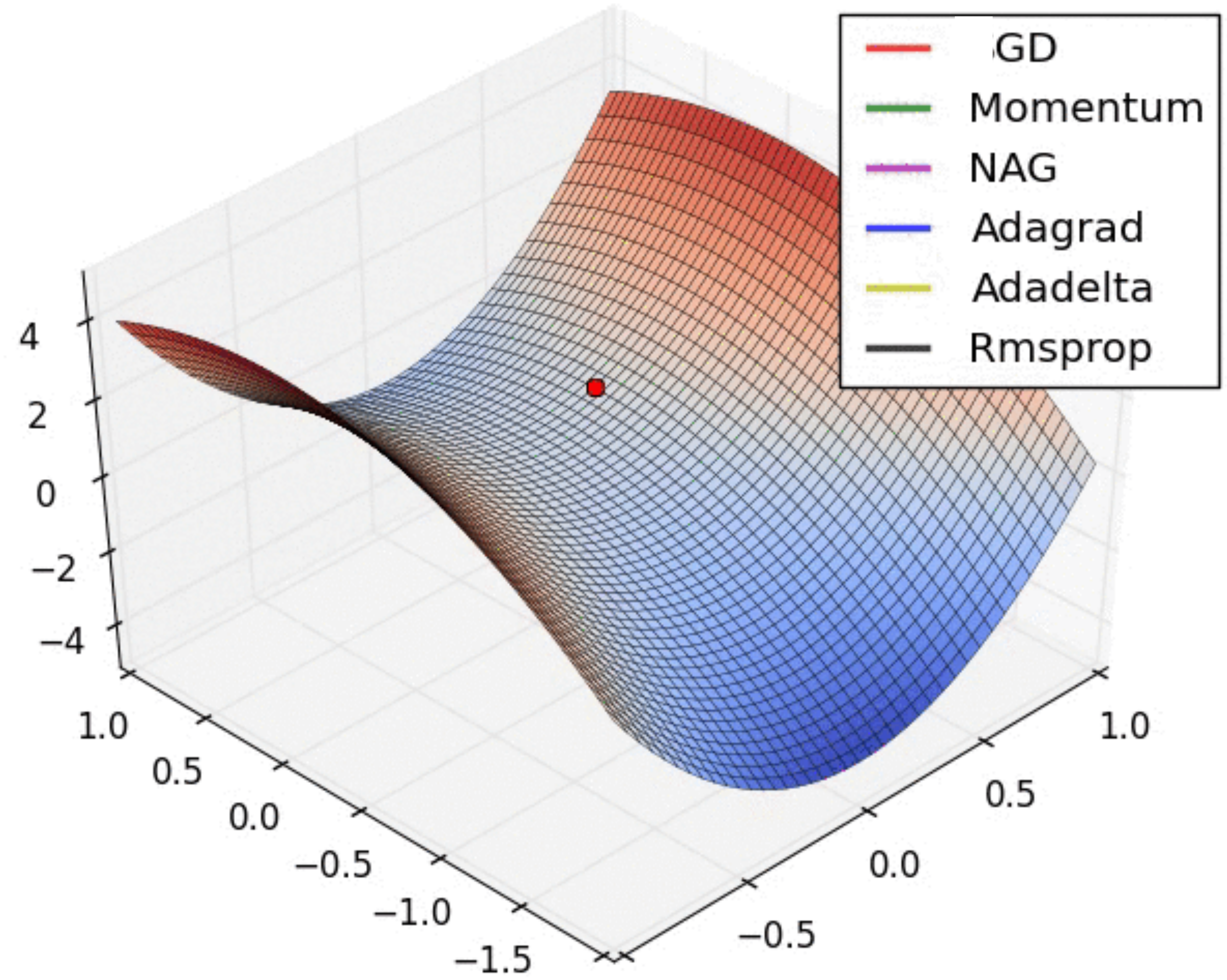


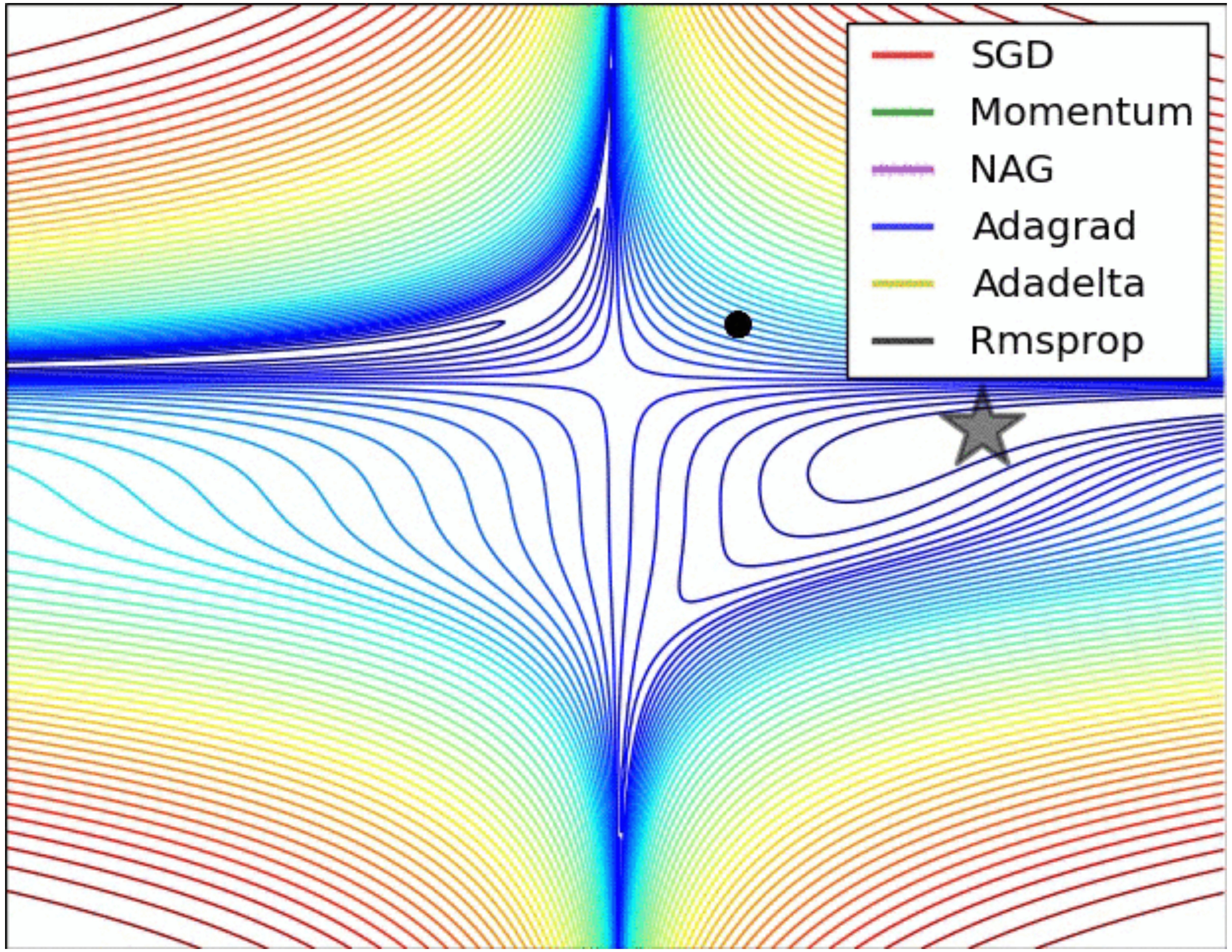
Other algorithms and sources

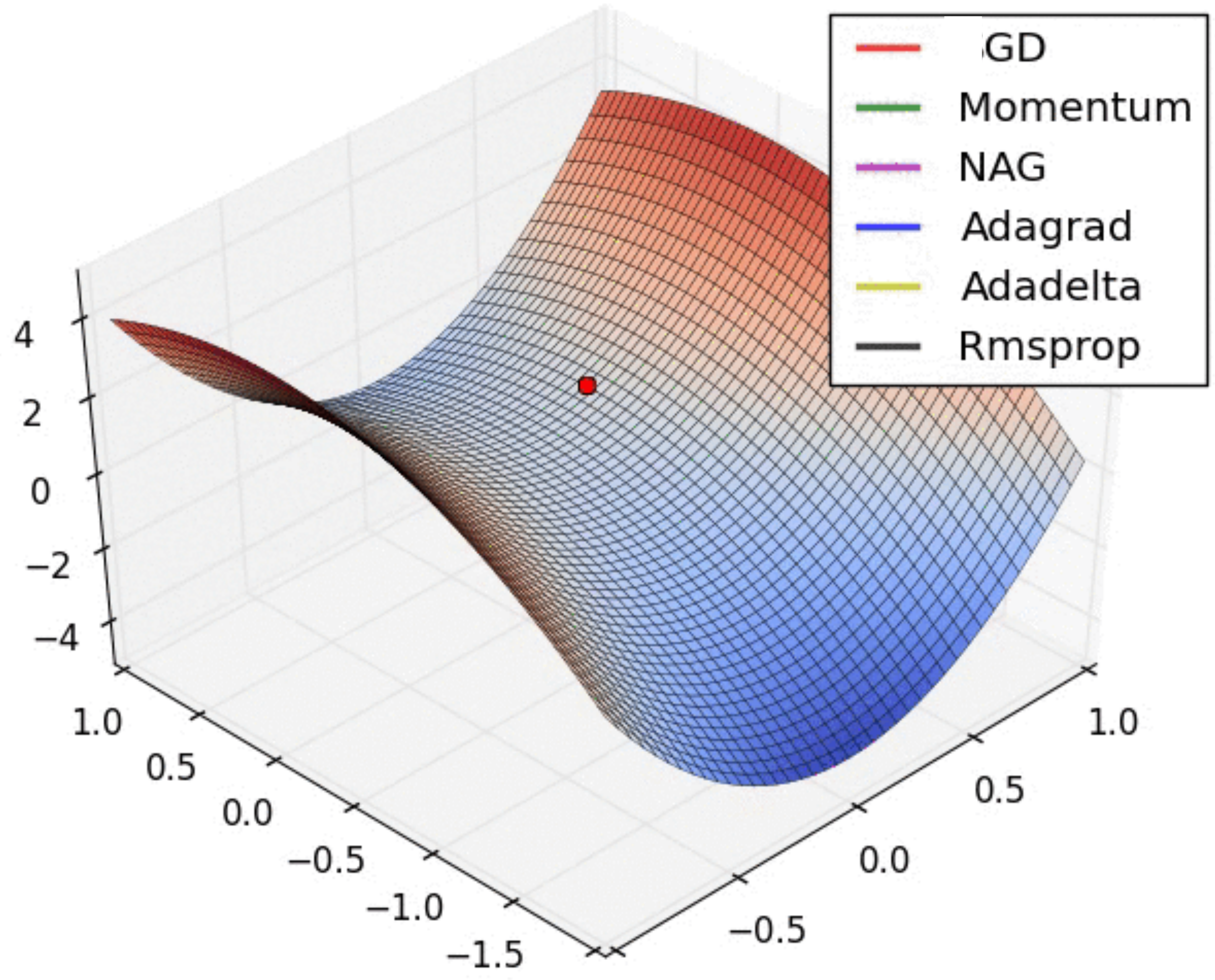
- Not a complete list: **AdaMax, Nadam, AMSGrad**, ..
- A nice blog post on the matter:

http://ruder.io/optimizing-gradient-descent/

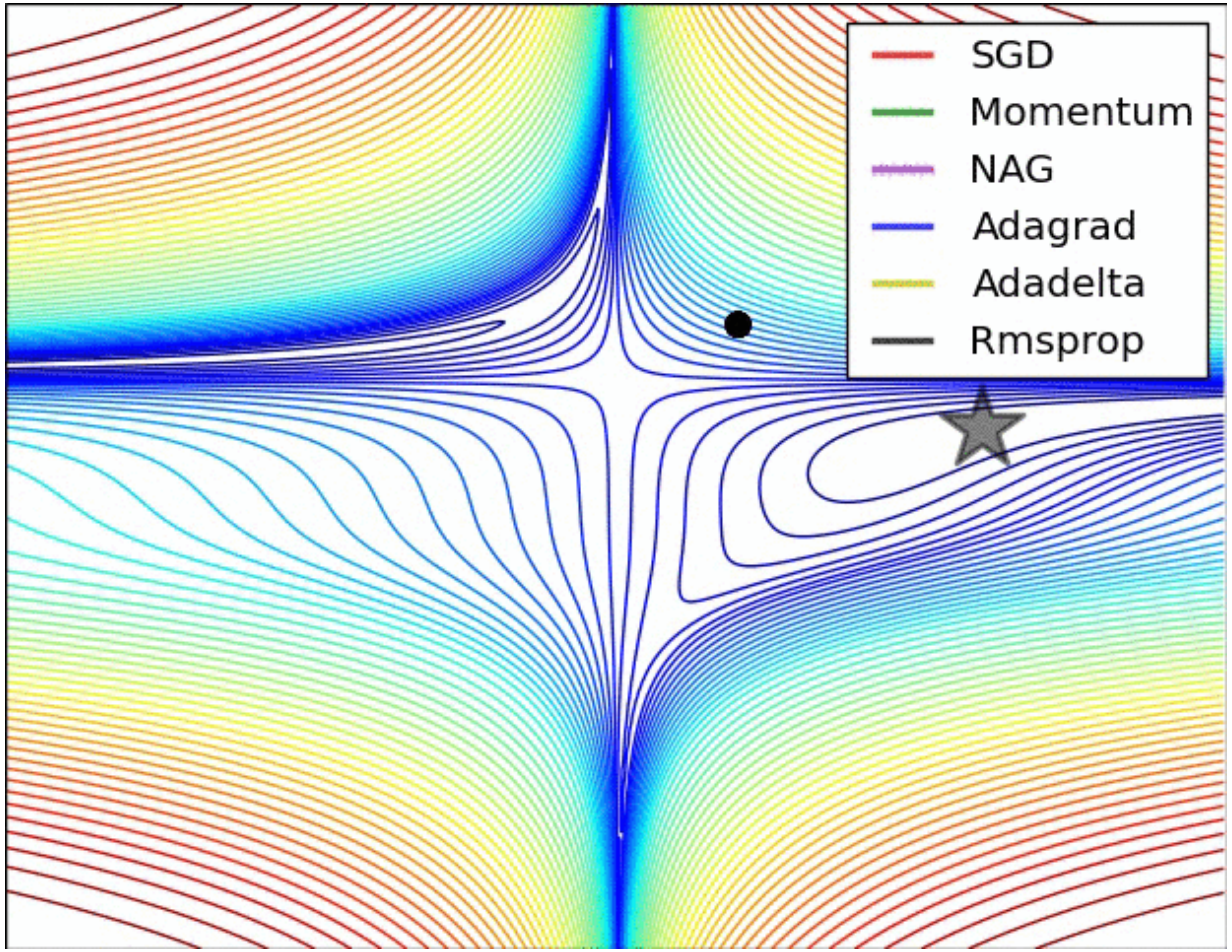
- Choosing the right algorithm: there is no consensus about it (see next slides)
- A visualization of their performance in toy examples:


GD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

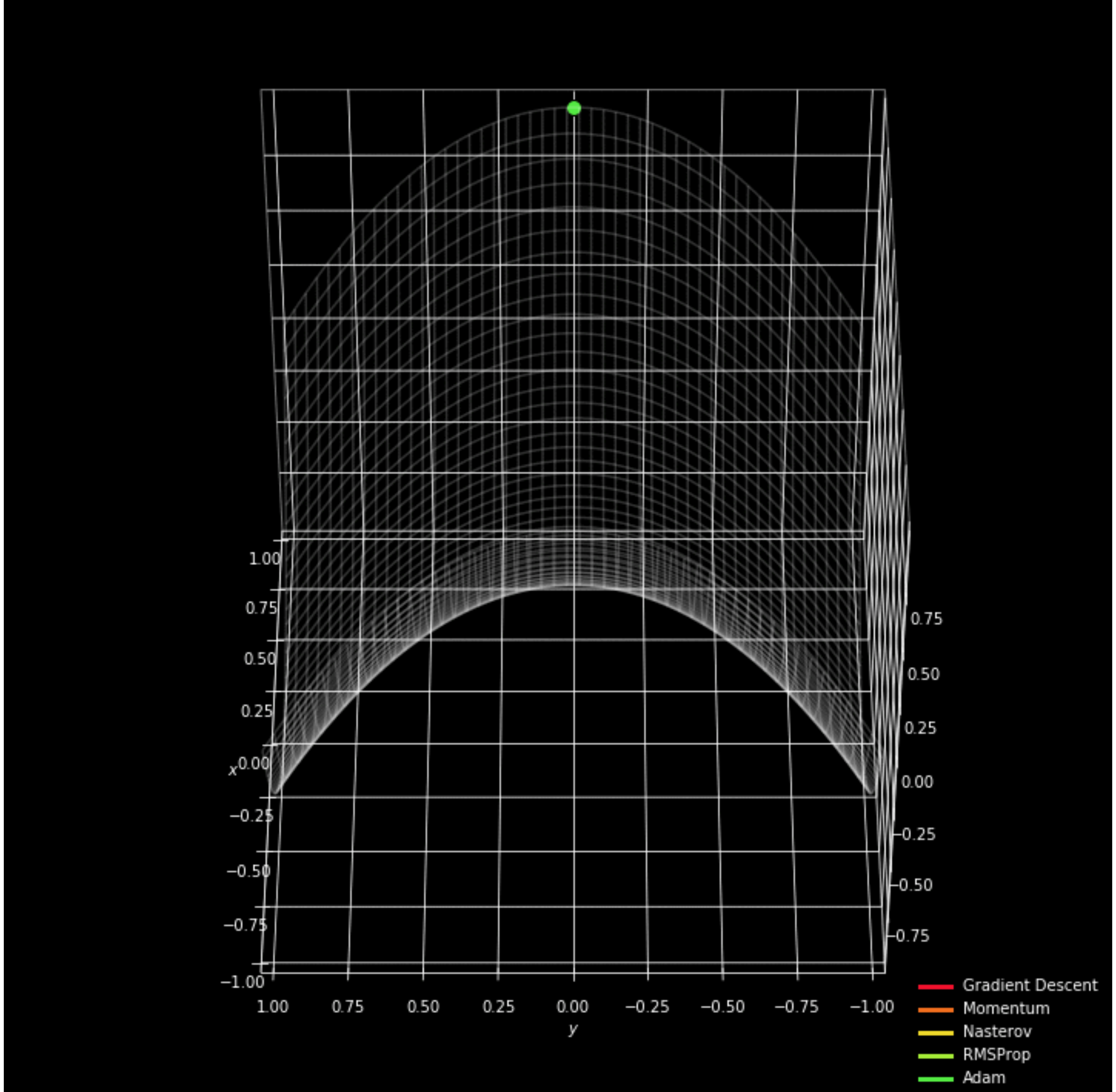

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

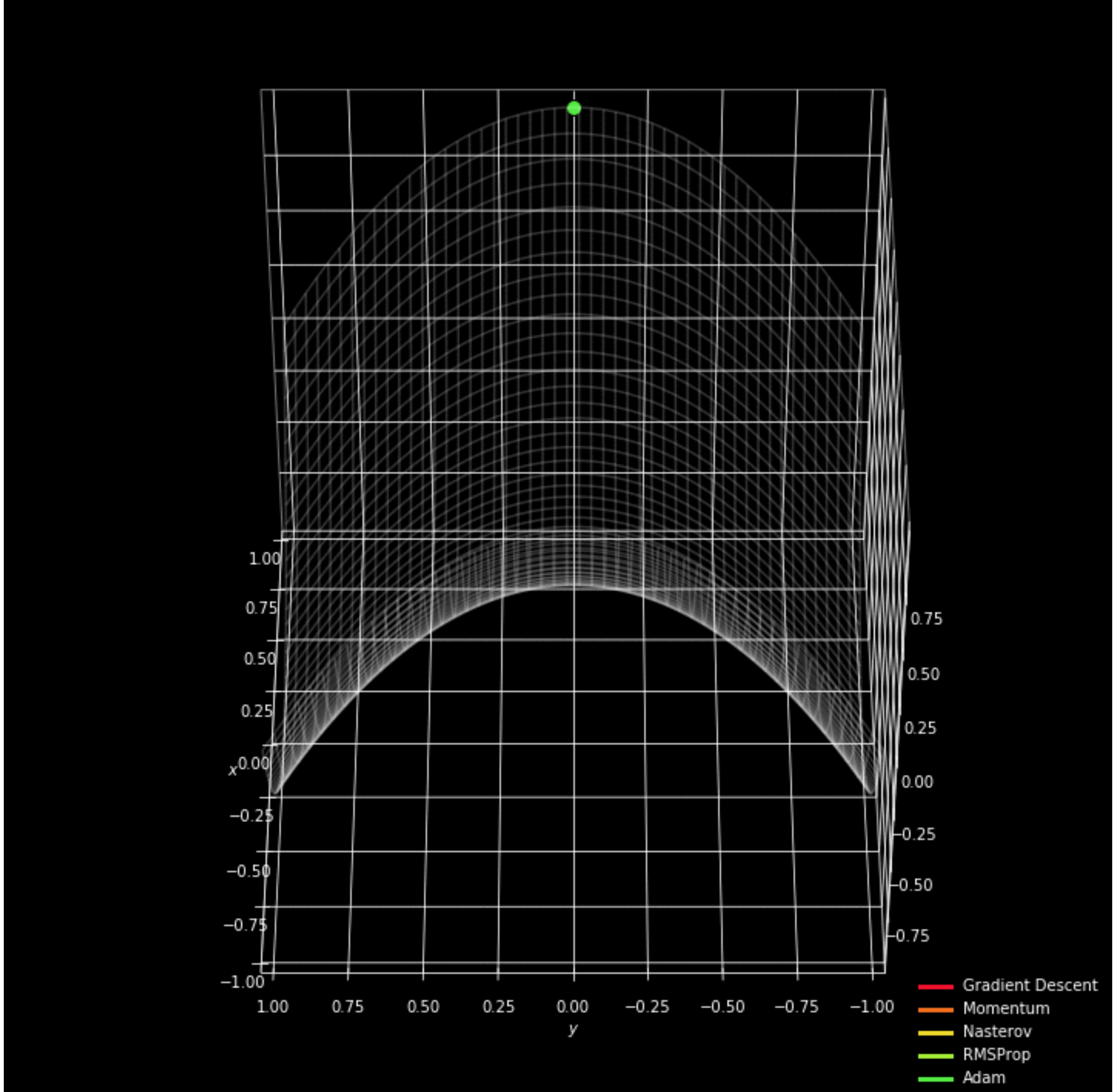


GD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5



SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


1.00
0.75
0.50
0.25
x
0.00
−0.25
−0.50
−0.75
−1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Gradient Descent
Momentum
Nasterov
RMSProp
Adam



x
y
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Gradient Descent
Momentum
Nasterov
RMSProp
Adam

Other algorithms and sources

- Not a complete list: **AdaMax, Nadam, AMSGrad**, ..
- A nice blog post on the matter:

  http://ruder.io/optimizing-gradient-descent/

- Choosing the right algorithm: there is no consensus about it (see next slides)
- A visualization of their performance in toy examples:
- Bonus discussion: **The marginal value of adaptive methods**

Other algorithms and sources

- Not a complete list: **AdaMax, Nadam, AMSGrad**, ..
- A nice blog post on the matter:

http://ruder.io/optimizing-gradient-descent/

- Choosing the right algorithm: there is no consensus about it (see next slides)
- A visualization of their performance in toy examples:
- Bonus discussion: **The marginal value of adaptive methods**

(Switch presentations)

Conclusion

- There are various algorithms for modern machine learning
- The most successful of them are gradient based; however, there are variations that make difference in practice (acceleration helps, adaptive learning rates work for most applications, etc).
- Which algorithm to use depends on the problem and the resources at hand
- These topics are highly attractive (research-wise): the idea is to devise new algorithms that achieve practical acceleration (with minimal tuning effort)