

# COMP 414/514: Optimization – Algorithms, Complexity and Approximations

Lecture 5

# 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  $\kappa$ )

# 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  $\kappa$ )

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)

# 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

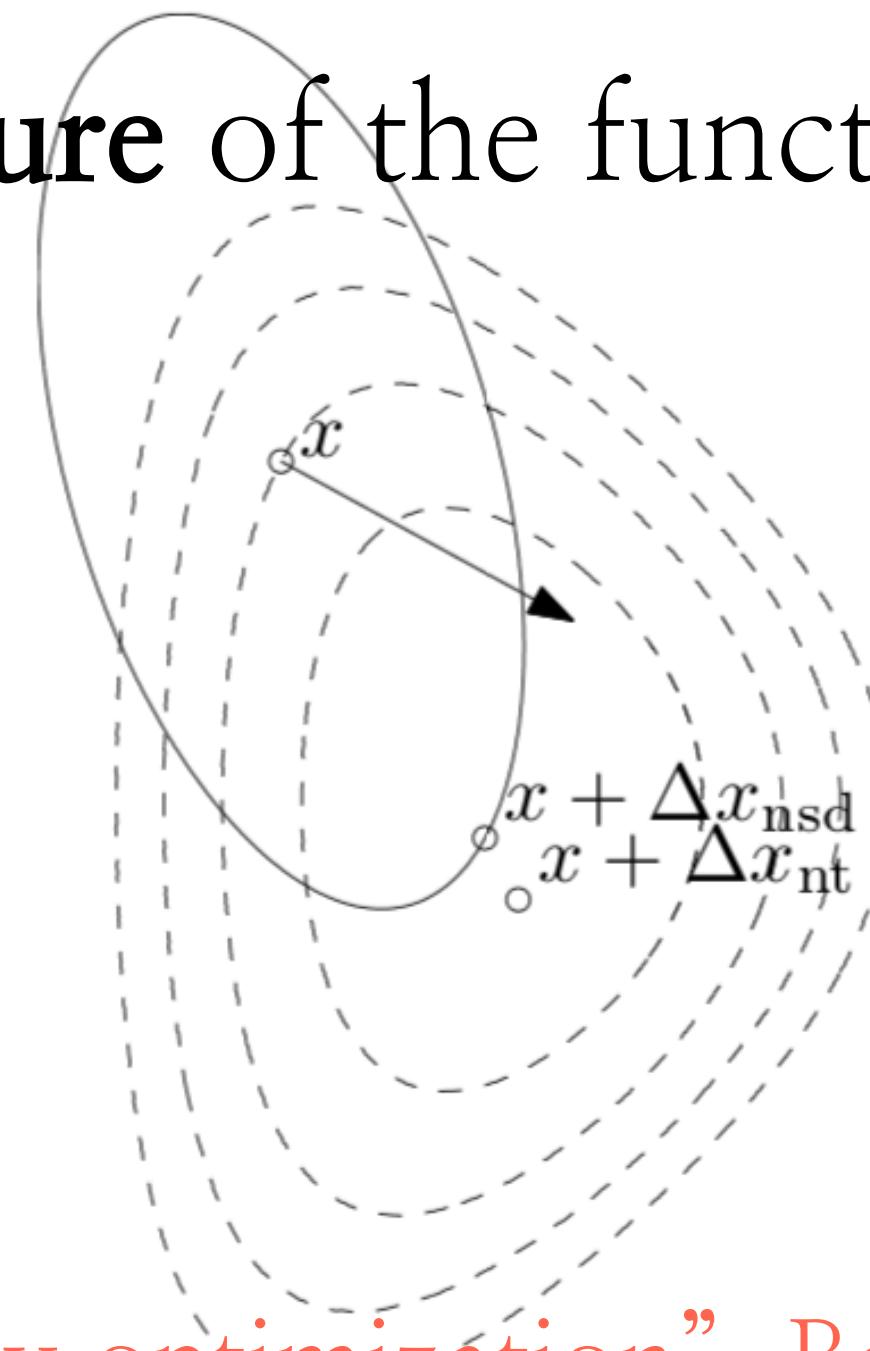
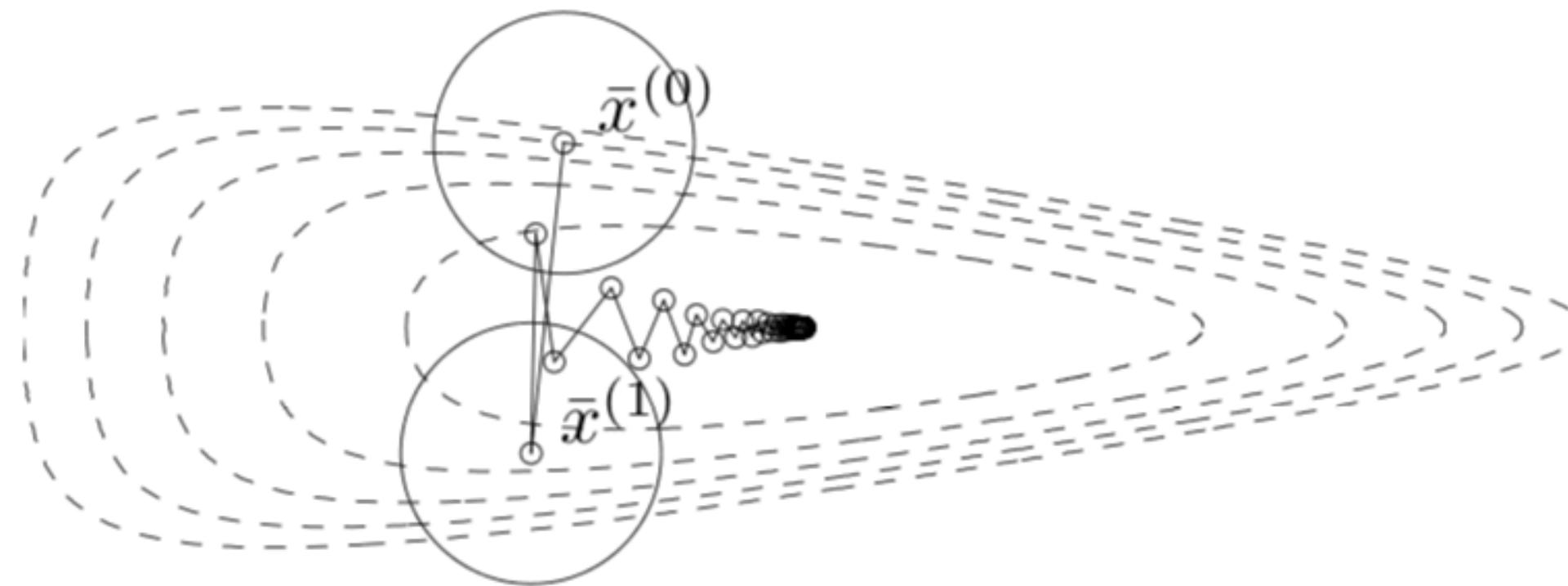
# The notorious Newton's method

Demo

# General comments of Newton's method

# General comments of Newton's method

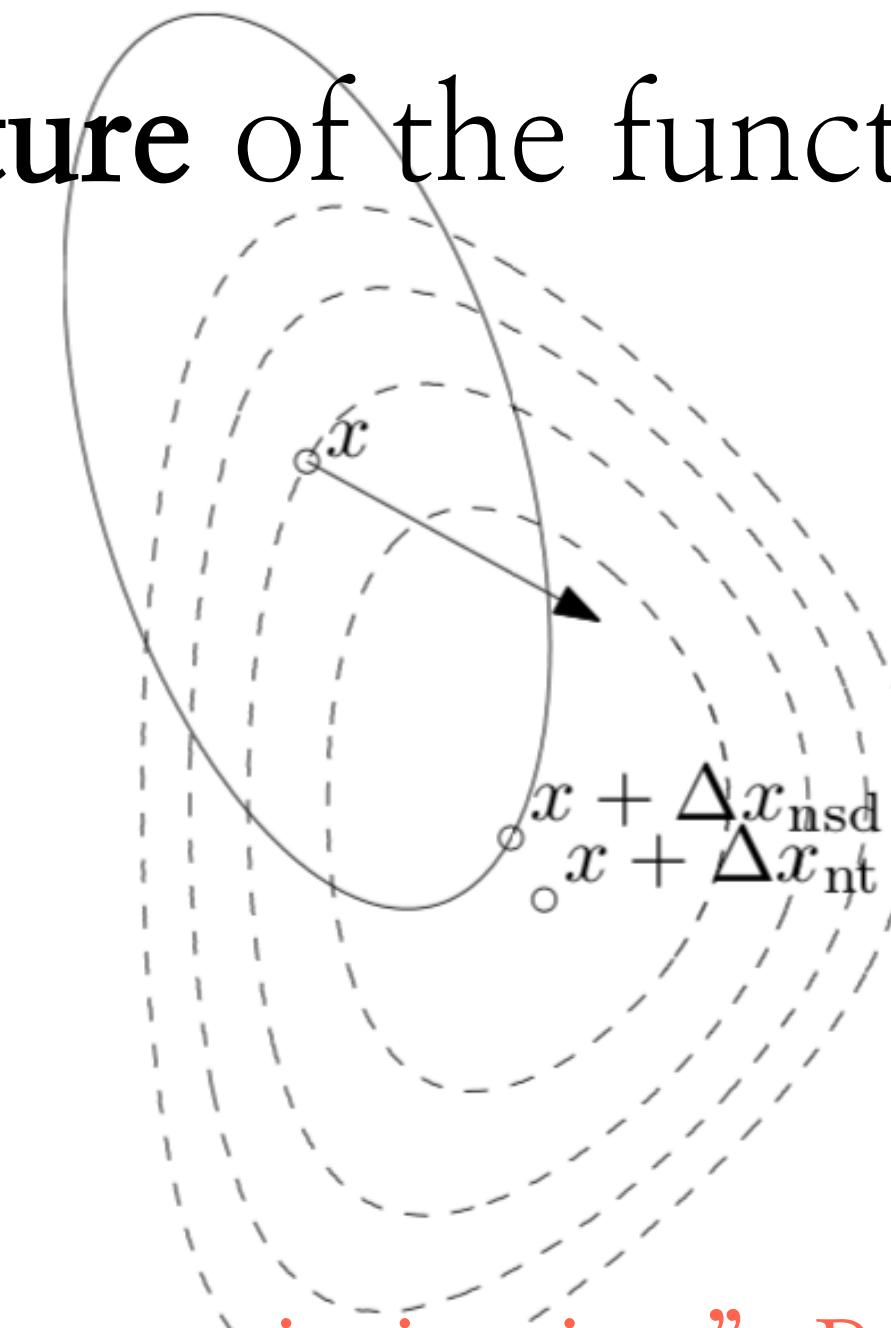
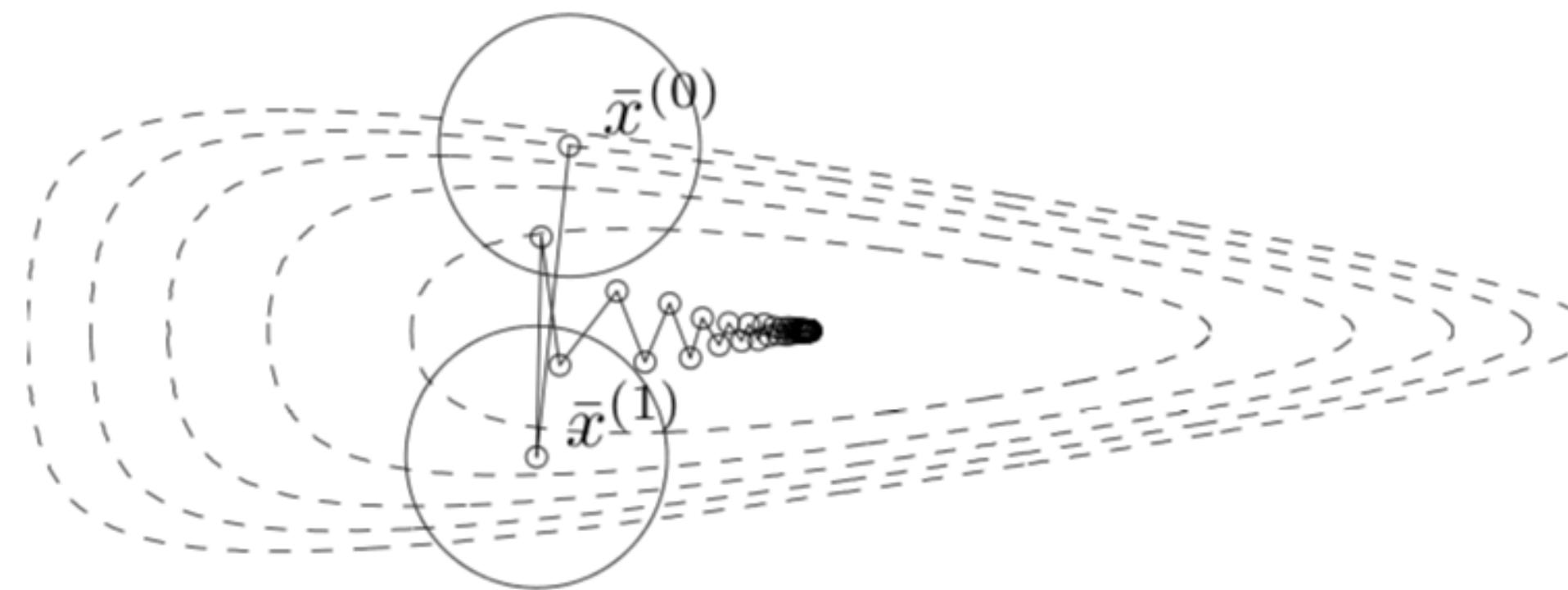
- Newton's method exploits the **local curvature** of the function



“Convex optimization”, Boyd and Vandenberghe

# General comments of Newton's method

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

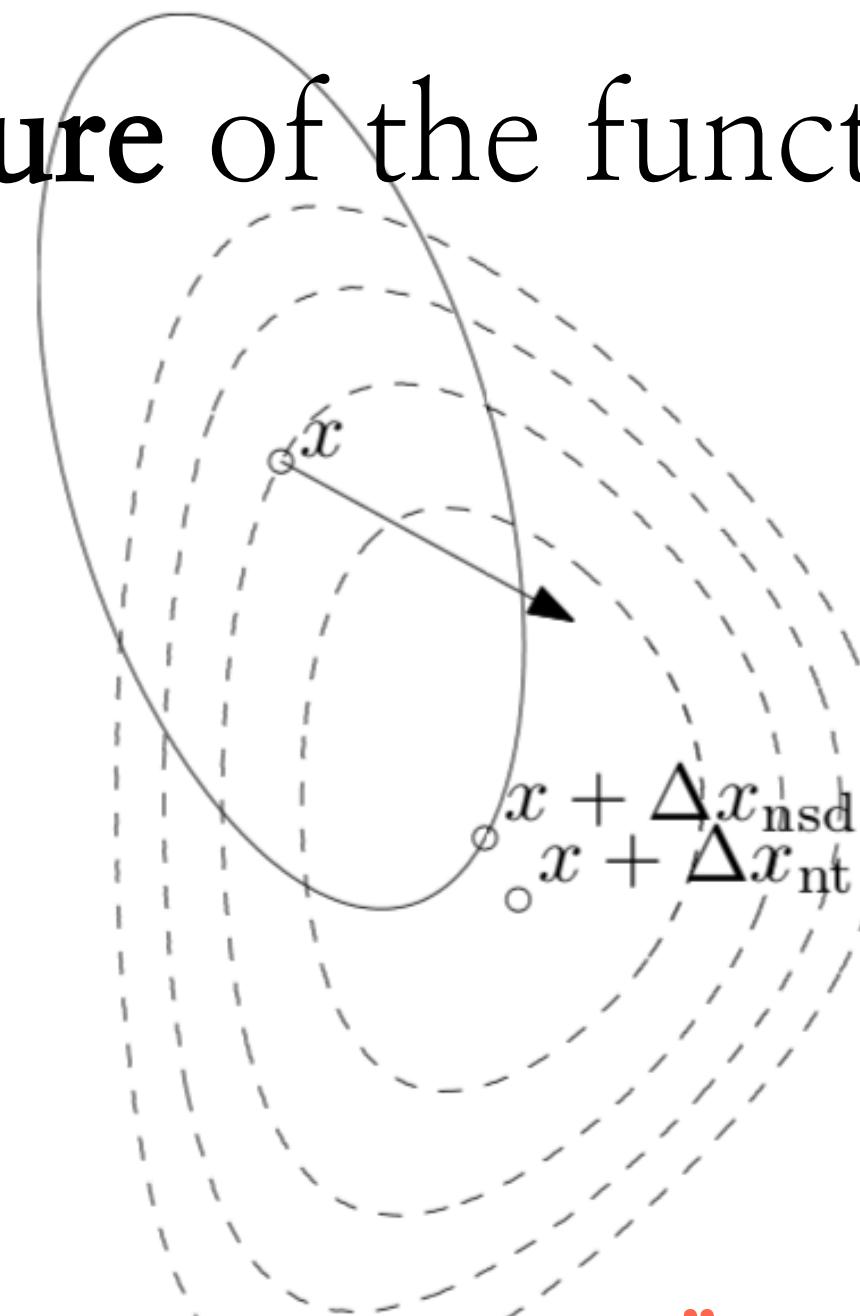
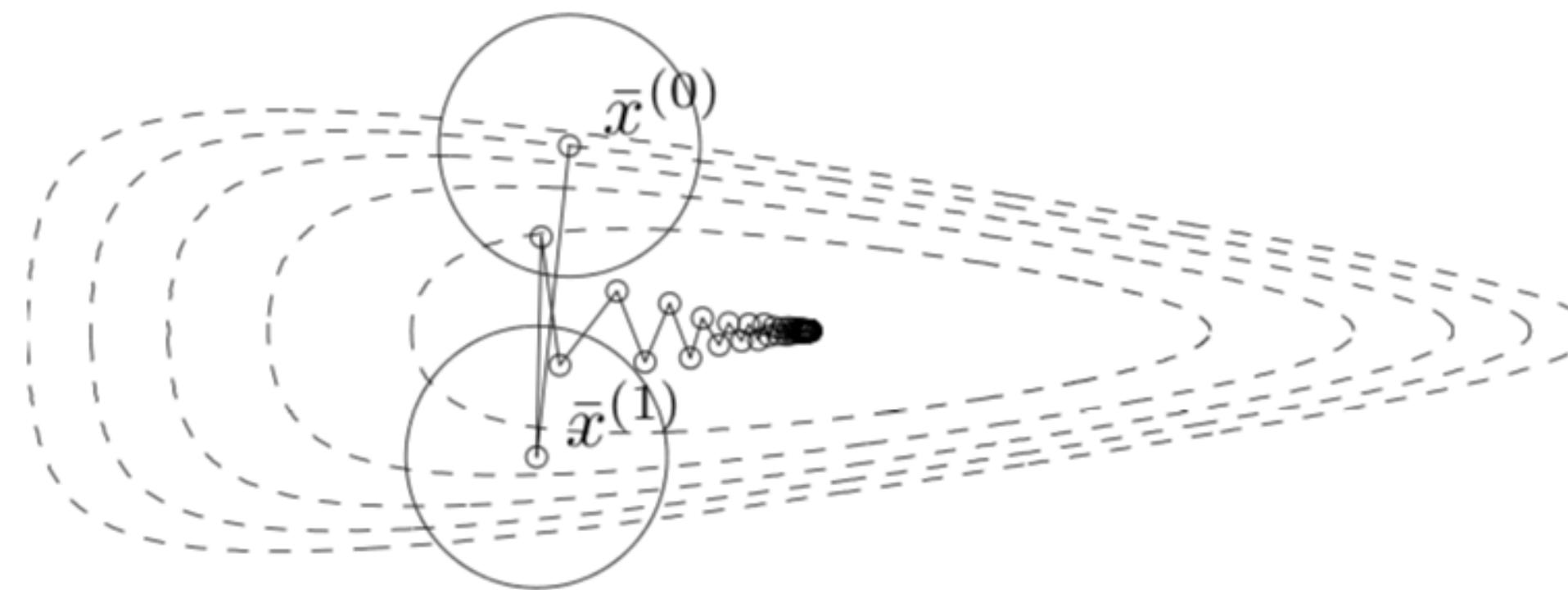


“Convex optimization”, Boyd and Vandenberghe

- Each iteration is **more computationally expensive**

# General comments of Newton's method

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

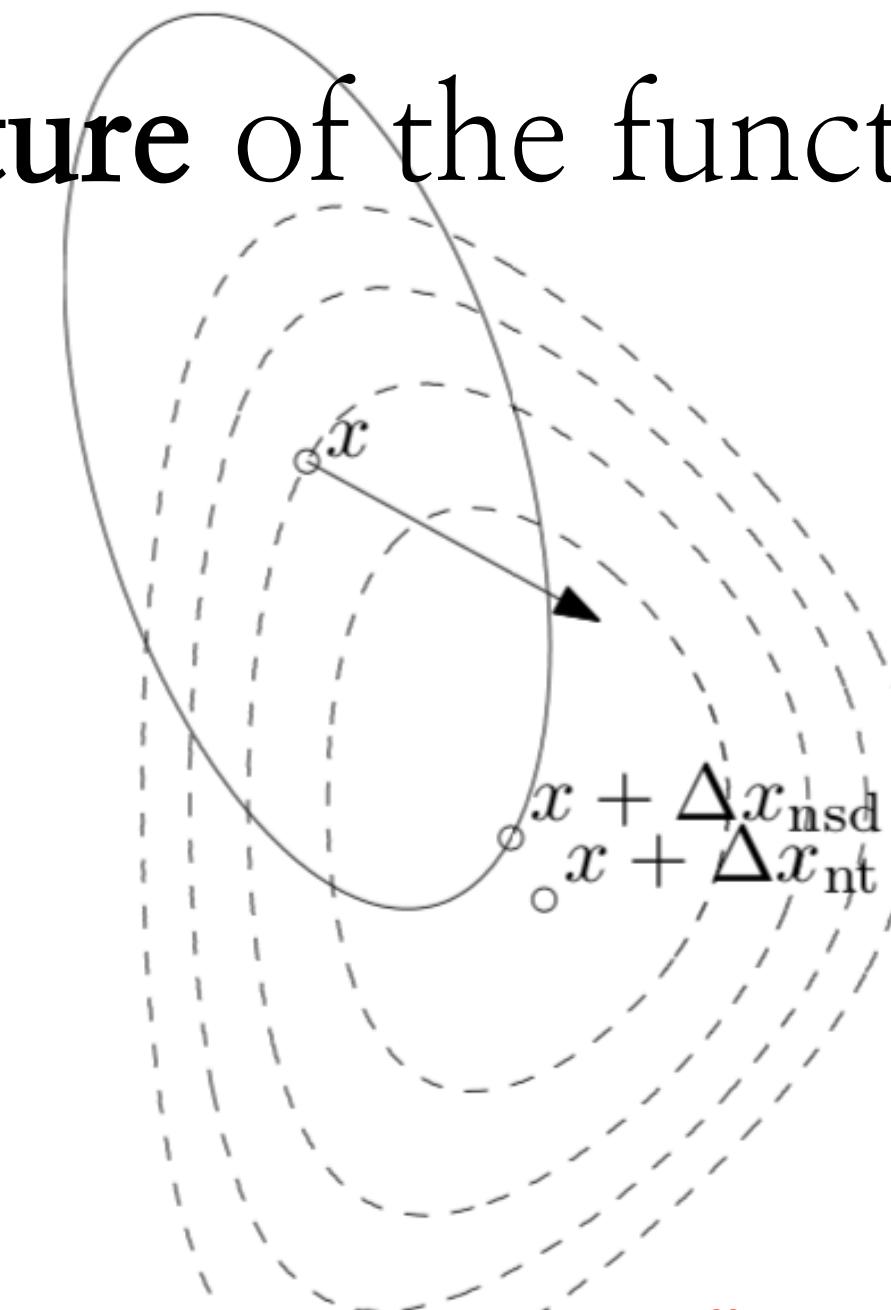
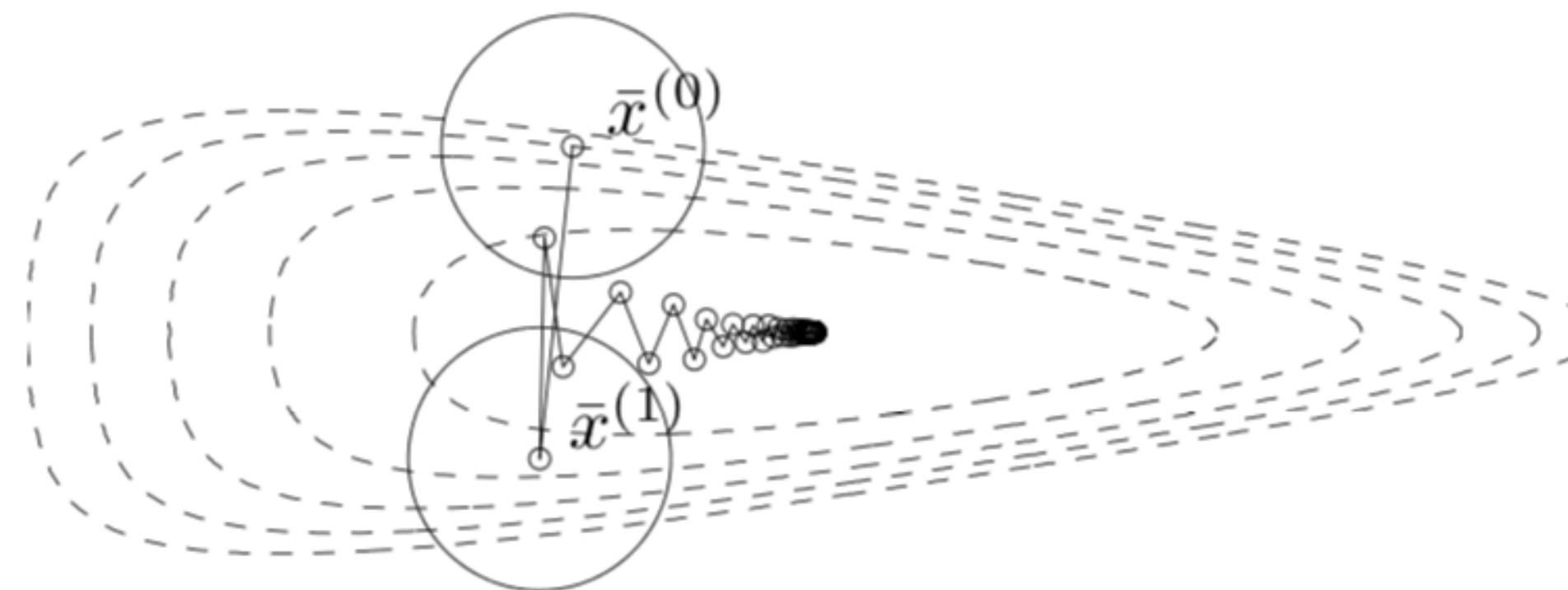


“Convex optimization”, Boyd and Vandenberghe

- 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

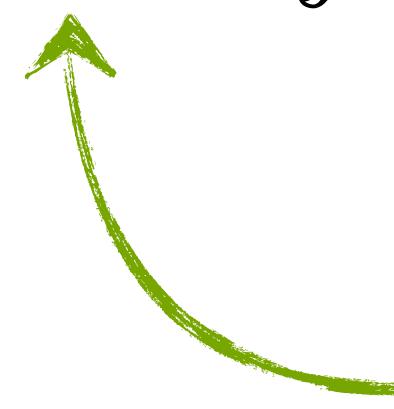
- Quasi–Newton methods

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

# Between gradient descent and Newton's method

- Quasi–Newton methods

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

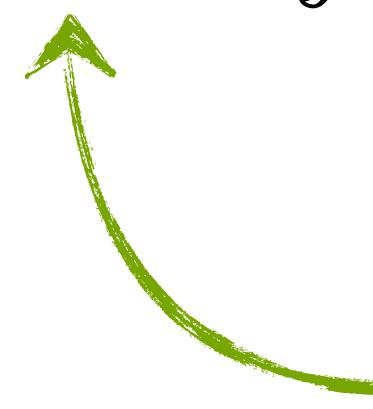


Approximation of the  
inverse Hessian

# Between gradient descent and Newton's method

- Quasi–Newton methods

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



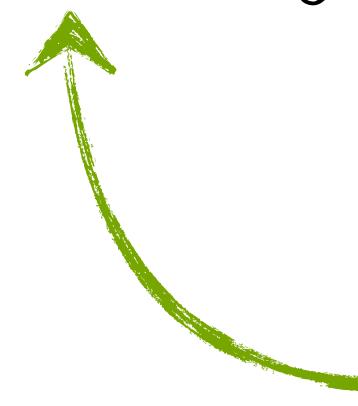
Approximation of the  
inverse Hessian

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

# Between gradient descent and Newton's method

- Quasi–Newton methods

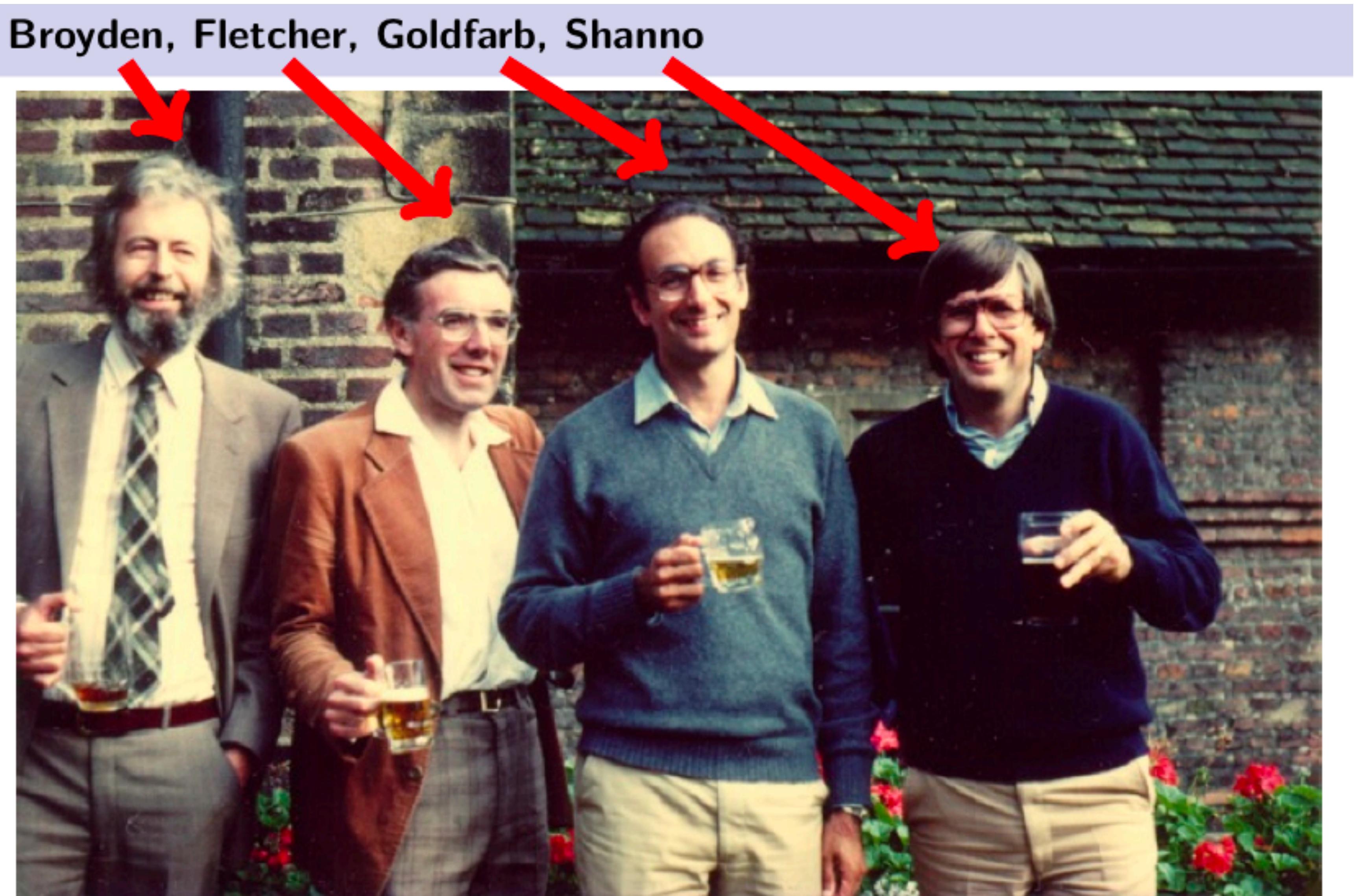
$$x_{t+1} = x_t - \eta B_t \nabla f(x_t)$$



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:

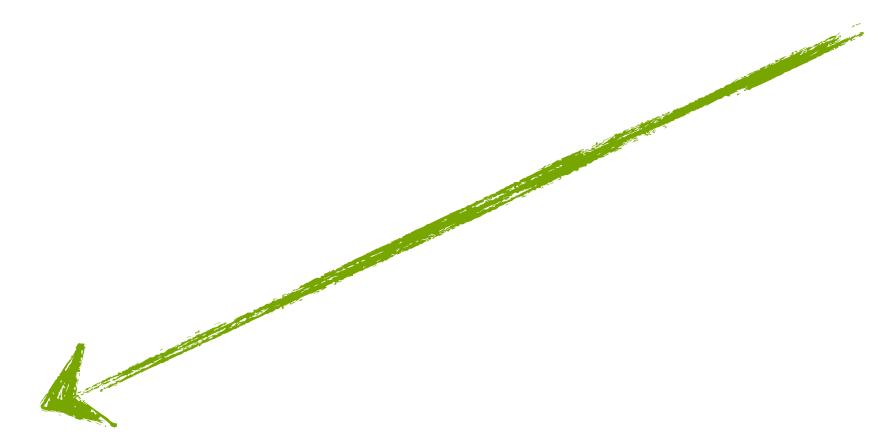
$$\begin{aligned} & \min_{B \succ 0} \|B - B_t\|_F^2 \\ \text{s.t. } & B = B^\top, \\ & \Delta x = B (\nabla f(x_t) - \nabla f(x_{t-1})) \end{aligned}$$

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

(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

(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

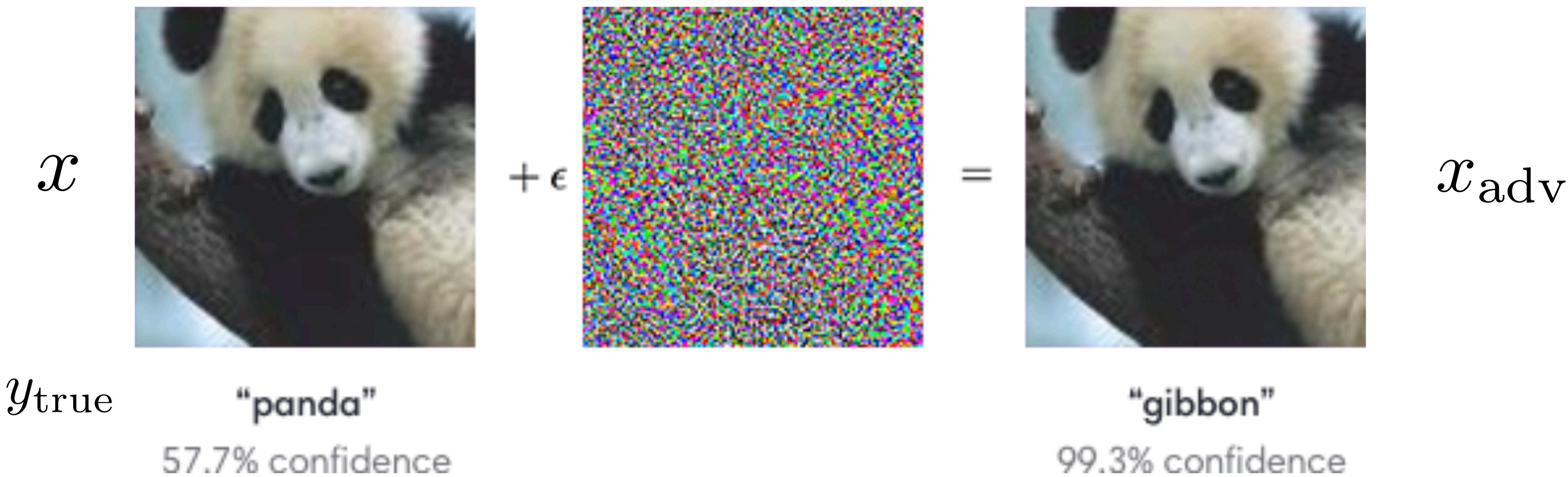
(A quick description)

- 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



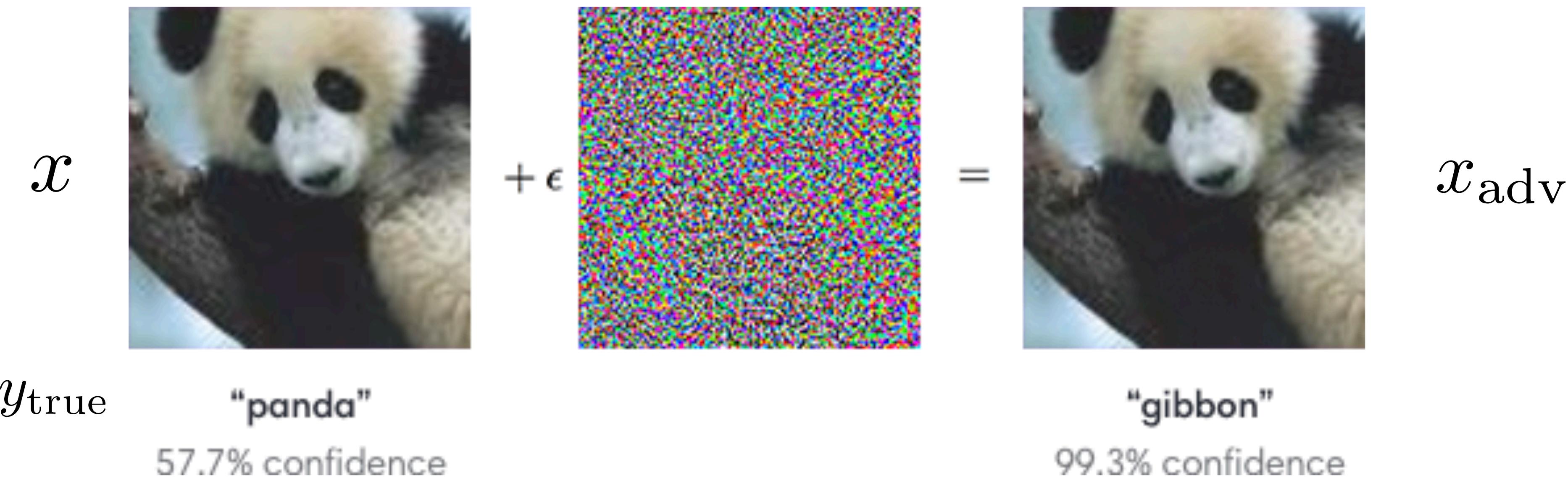
$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla f(x, y_{\text{true}}))$$

(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



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

We are looking into directions that move away from the minimum

(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  $\delta$ , 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**

---

# Conclusion

- We studied algorithms beyond gradient descent: Newton's method, quasi-Newton algorithms, derivative-free optimization, and natural gradient descent method
- Which one to use depends on the problem at hand (accuracy, complexity)
- While these methods match or even overcome the lower bounds, we have been “cheating” by exploiting exact or approximate second-order information

# Next lecture

- We will discuss a bit about acceleration and stochasticity in optimization