

IFT6390

Fondements de l'apprentissage machine

**Optimization**

**Basic principles and techniques**

Professor: Ioannis Mitliagkas

Slides: Pascal Vincent

# Optimization?

- The **training phase** of a learning algorithm often involves **optimization**
- That is, finding the **values of the parameters**  $\theta$  of the function  $f_\theta$  that **minimize** (or maximize) some **objective**  $J(\theta)$
- Objective example: empirical risk (or «mean error») on the training data:

$$J(\theta) = \hat{R}(f_\theta, D_n)$$

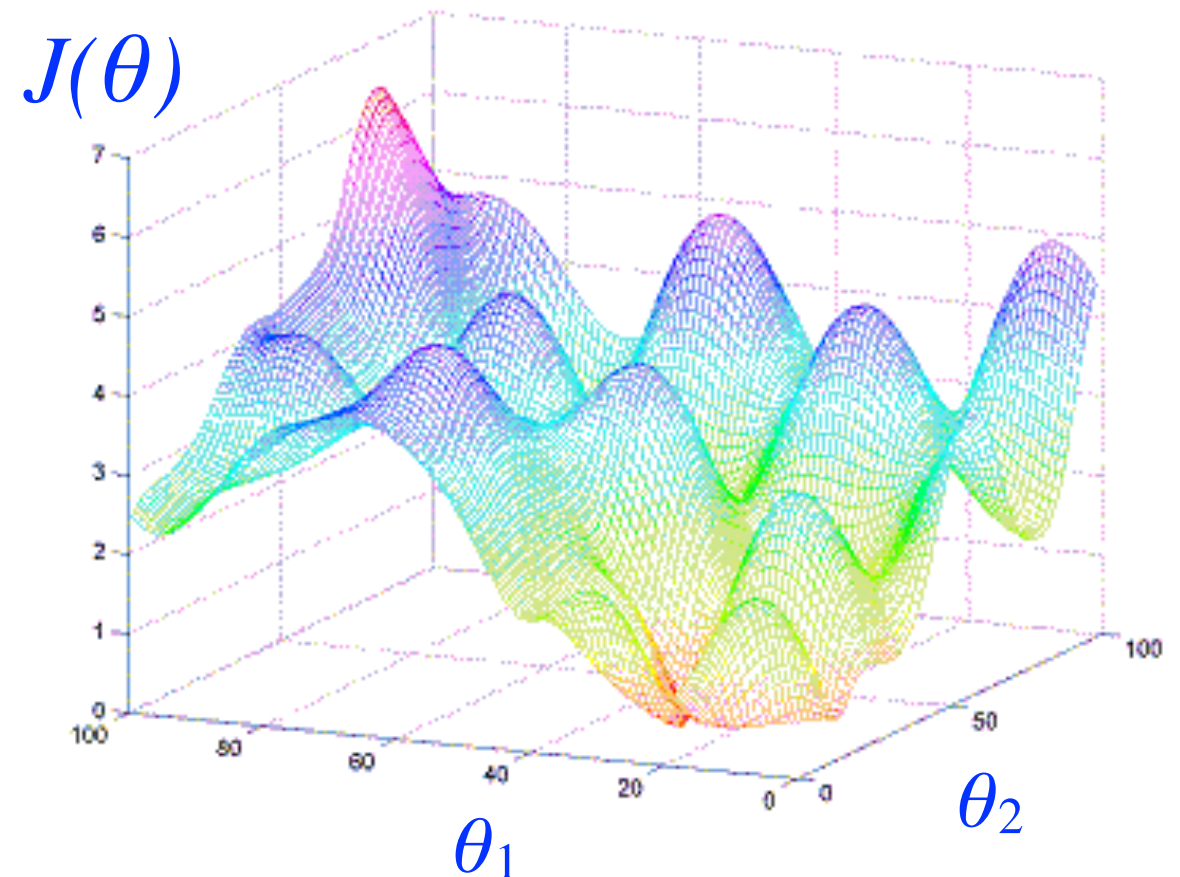
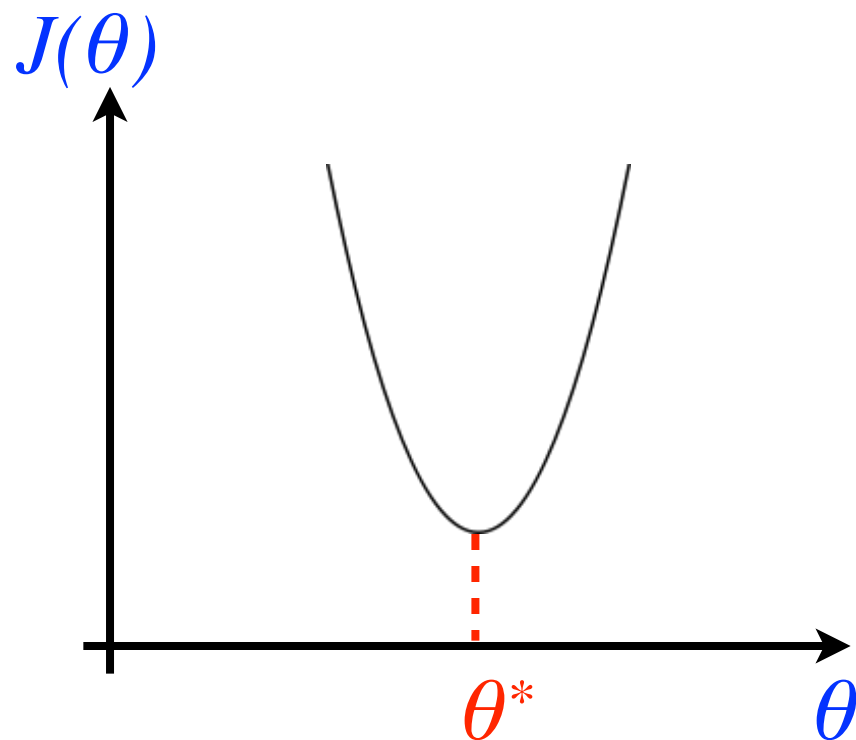
# The optimization problem

$$\theta^* = \arg \min_{\theta} J(\theta)$$

Optimal value of  
the parameters

How to  
find it ???

Landscape of the loss function:

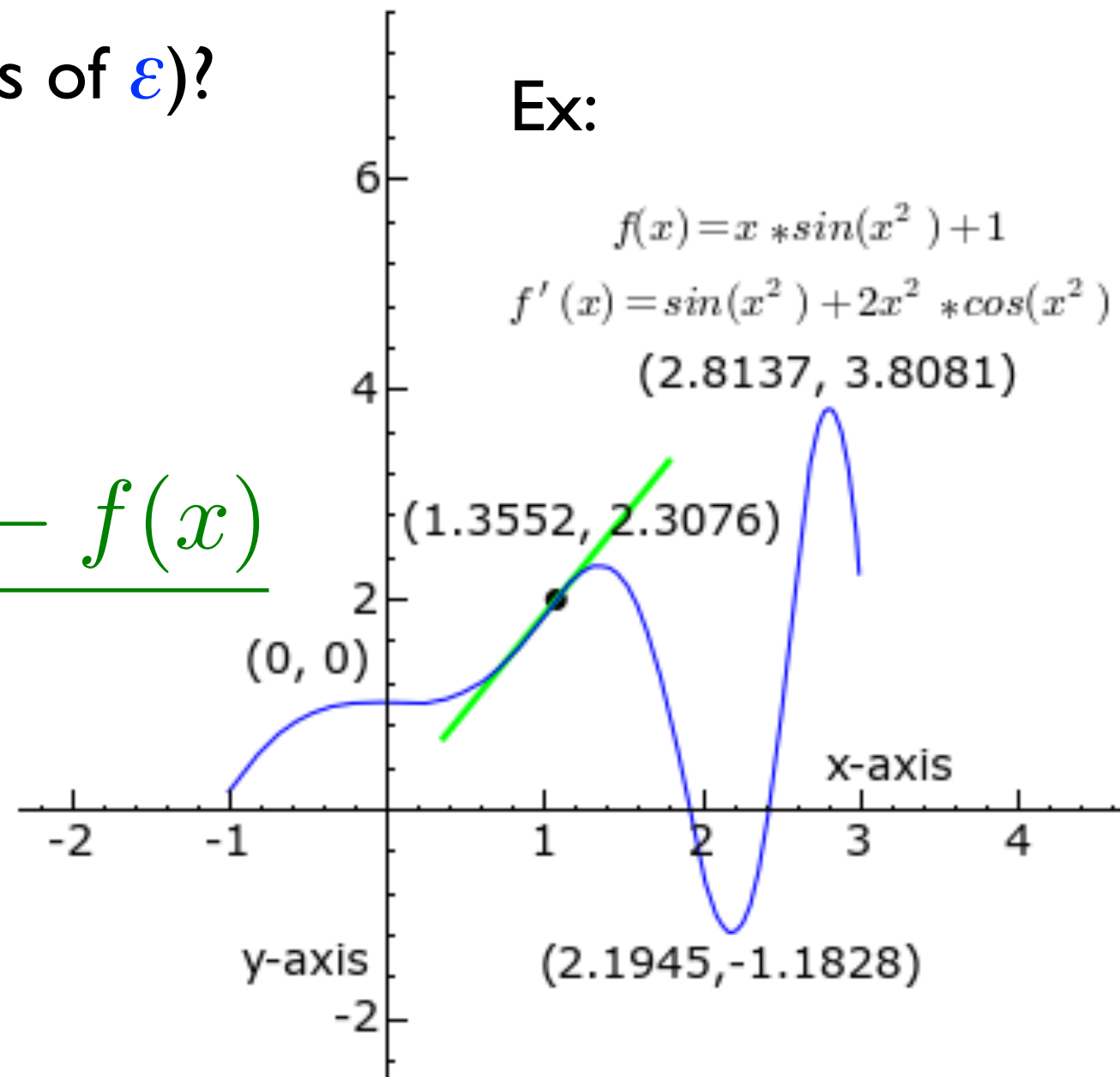


# Reminder: derivative

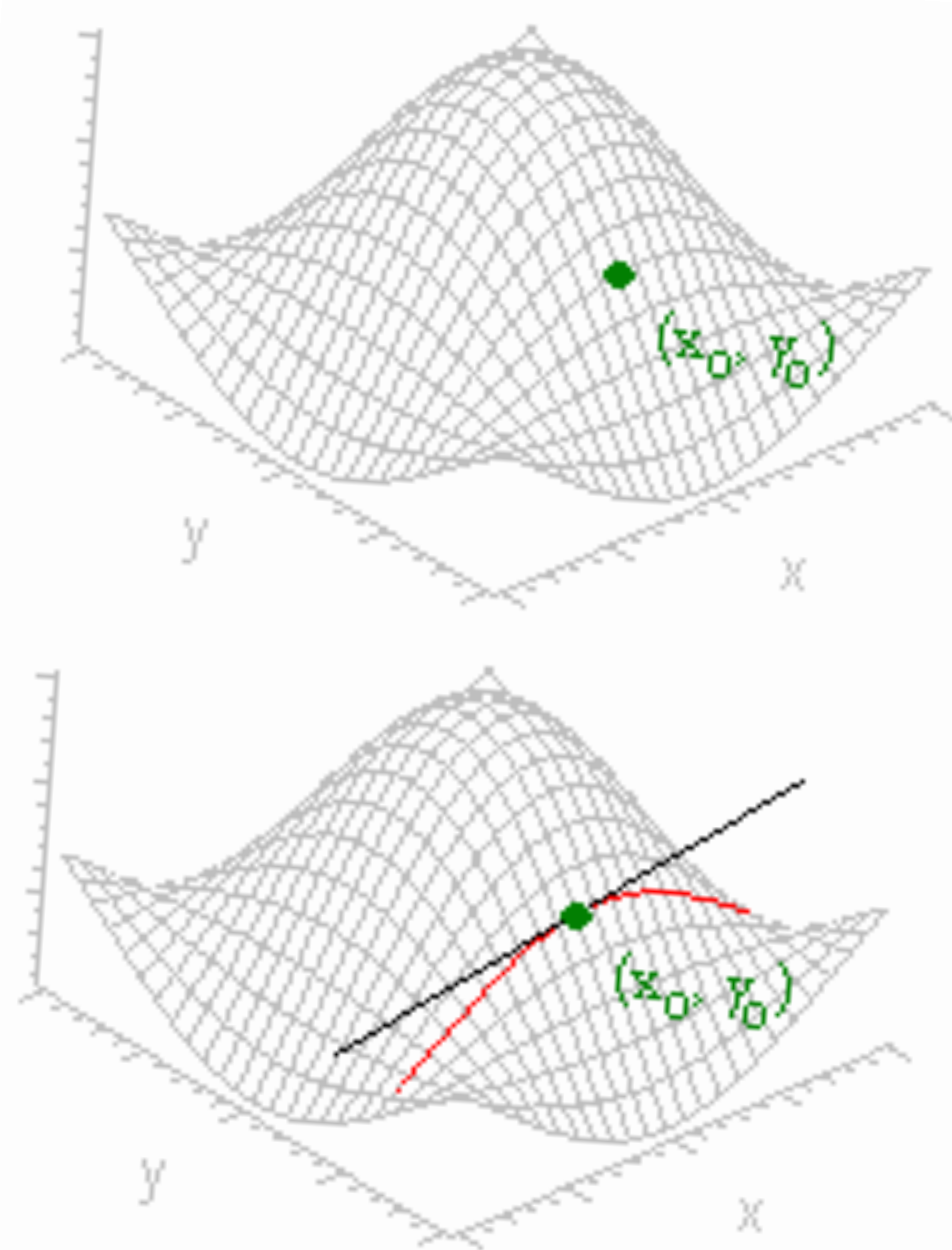
- Measures the **sensitivity** of a function  $f$  to a change of its input (at a point  $x$ ).
- Let  $f : \mathbb{R} \rightarrow \mathbb{R}$ . At a point  $x$ , if we add a small  $\epsilon$  to  $x$ , how much will  $f(x)$  move (in multiples of  $\epsilon$ )?

$$\begin{aligned} f'(x) &= \frac{\partial f}{\partial x}(x) \\ &= \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon) - f(x)}{\epsilon} \end{aligned}$$

- Geometrically: this is the **slope** of the curve at each point.



# Partial derivative



- For a function taking vectors (i.e. several parameters) as input.

$$f : \mathbb{R}^m \rightarrow \mathbb{R}$$

- We can evaluate it at  $x$ :

$$f(x) = f(x_1, \dots, x_m)$$

- And measure how **sensitive** it is to each of its input separately (assuming the others fixed)

$$\begin{aligned} f'_{x_k}(x) &= \frac{\partial f}{\partial x_k}(x) && \text{partial derivative of } f \text{ with respect to } x_k, \text{ evaluated at } x \\ &= \lim_{\epsilon \rightarrow 0} \frac{f(x_1, \dots, x_k + \epsilon, \dots, x_m) - f(x_1, \dots, x_k, \dots, x_m)}{\epsilon} \end{aligned}$$

# Gradient

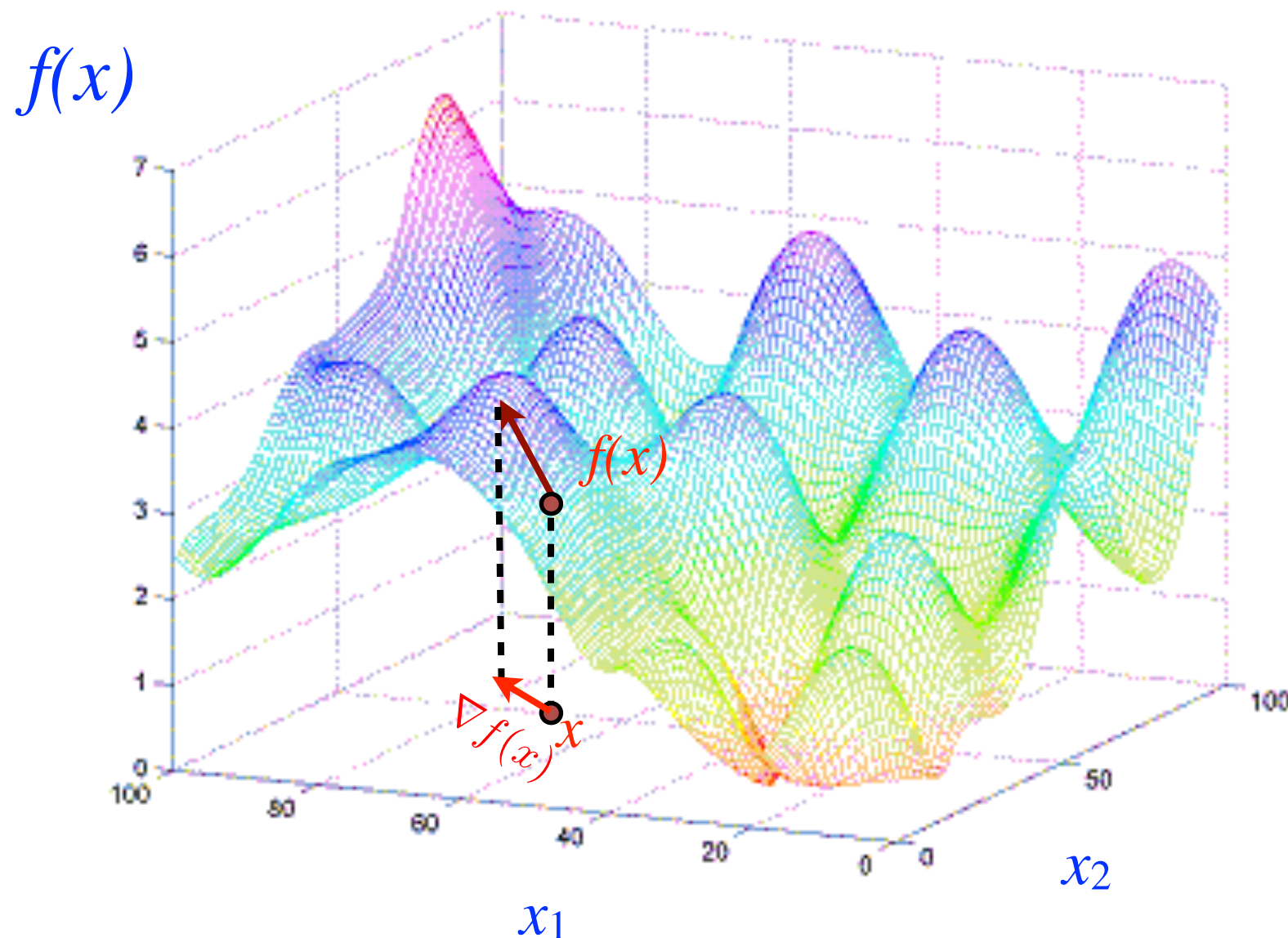
- The gradient is the vector of partial derivatives:

$$\nabla f(x) = \frac{\partial f}{\partial x}(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_m} \end{pmatrix} (x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_m}(x) \end{pmatrix}$$

if  $f : \mathbb{R}^m \rightarrow \mathbb{R}$  then  $\nabla f : \mathbb{R}^m \rightarrow \mathbb{R}^m$

# Gradient: geometry

- The gradient points in the direction of the steepest slope (uphill/ascending)





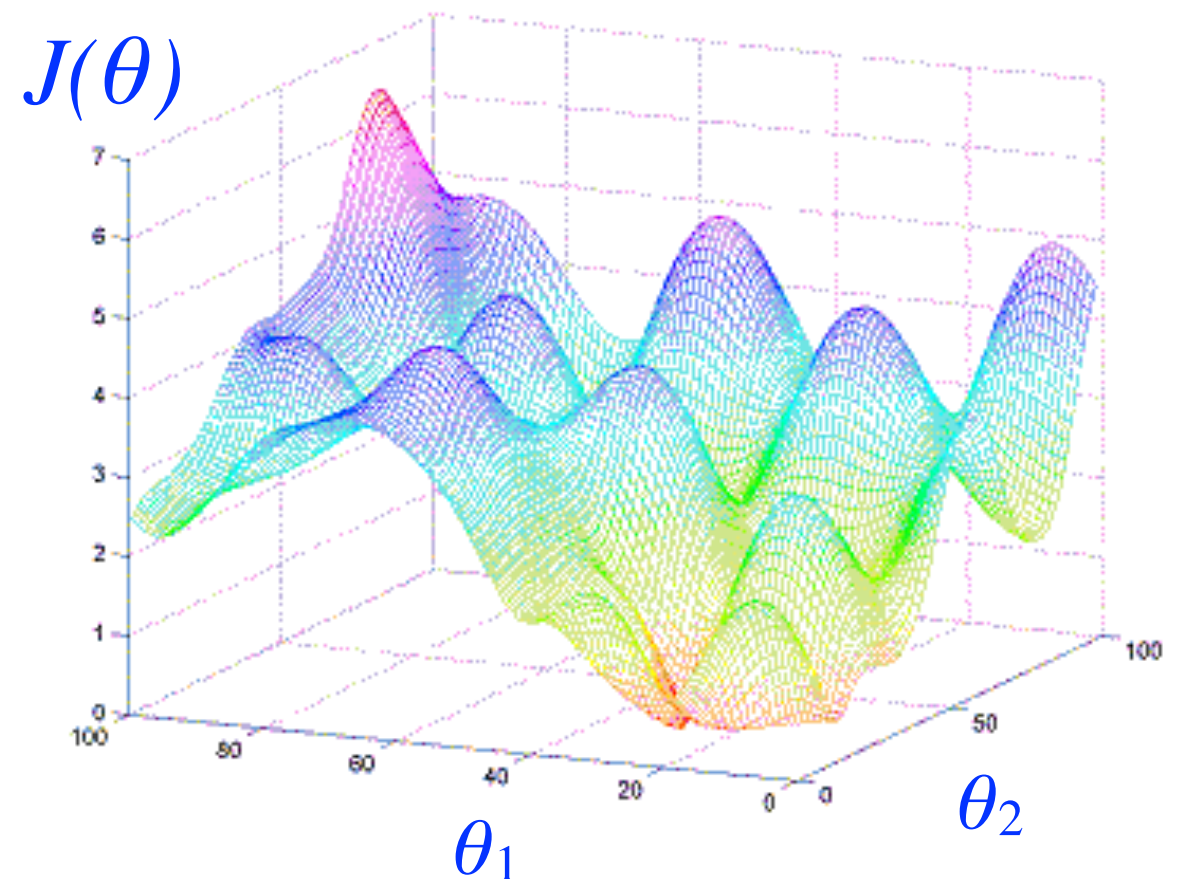
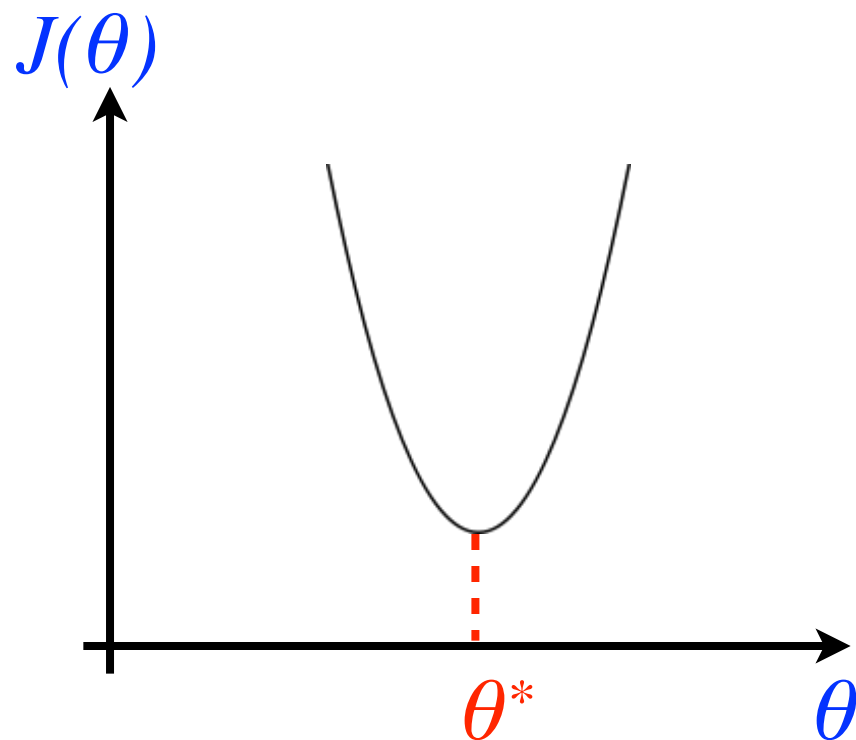
# The optimization problem

$$\theta^* = \arg \min_{\theta} J(\theta)$$

Optimal value of  
the parameters

How to  
find it ???

Landscape of the loss function:



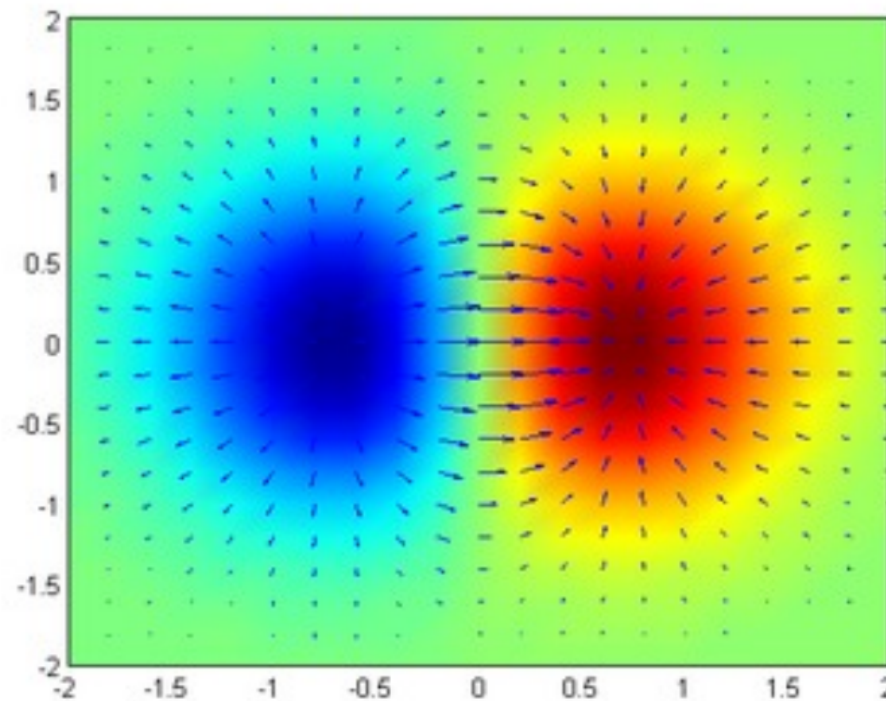


# Gradient

- Let  $\theta=(\theta_1, ..., \theta_m)$  be a vector of parameters.
- Let  $J(\theta)$  be a scalar function: the objective function to minimize.
- The gradient is the vector of the partial derivatives:

$$\nabla J(\theta) = \frac{\partial J}{\partial \theta}(\theta) = \begin{pmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_m} \end{pmatrix}(\theta) = \begin{pmatrix} \frac{\partial J}{\partial \theta_1}(\theta) \\ \frac{\partial J}{\partial \theta_2}(\theta) \\ \vdots \\ \frac{\partial J}{\partial \theta_m}(\theta) \end{pmatrix}$$

# Properties of the gradient



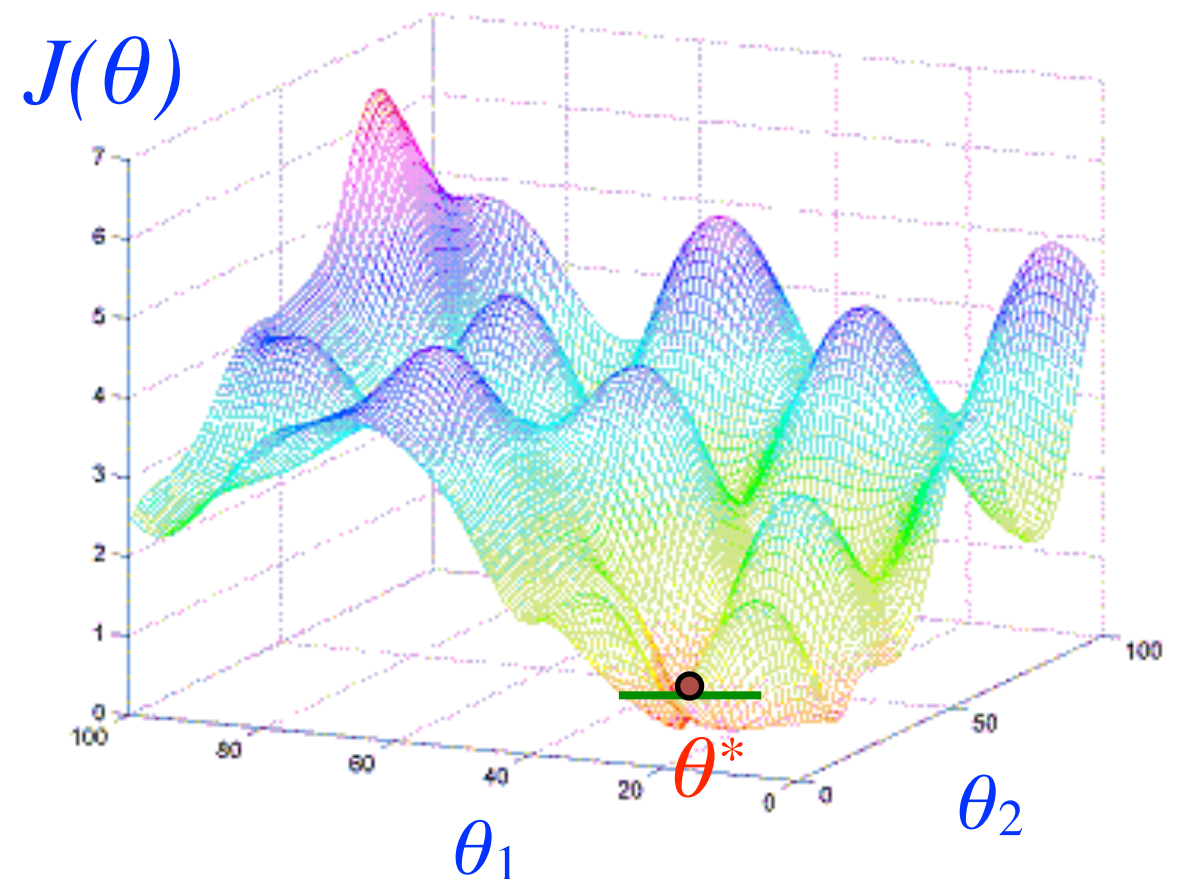
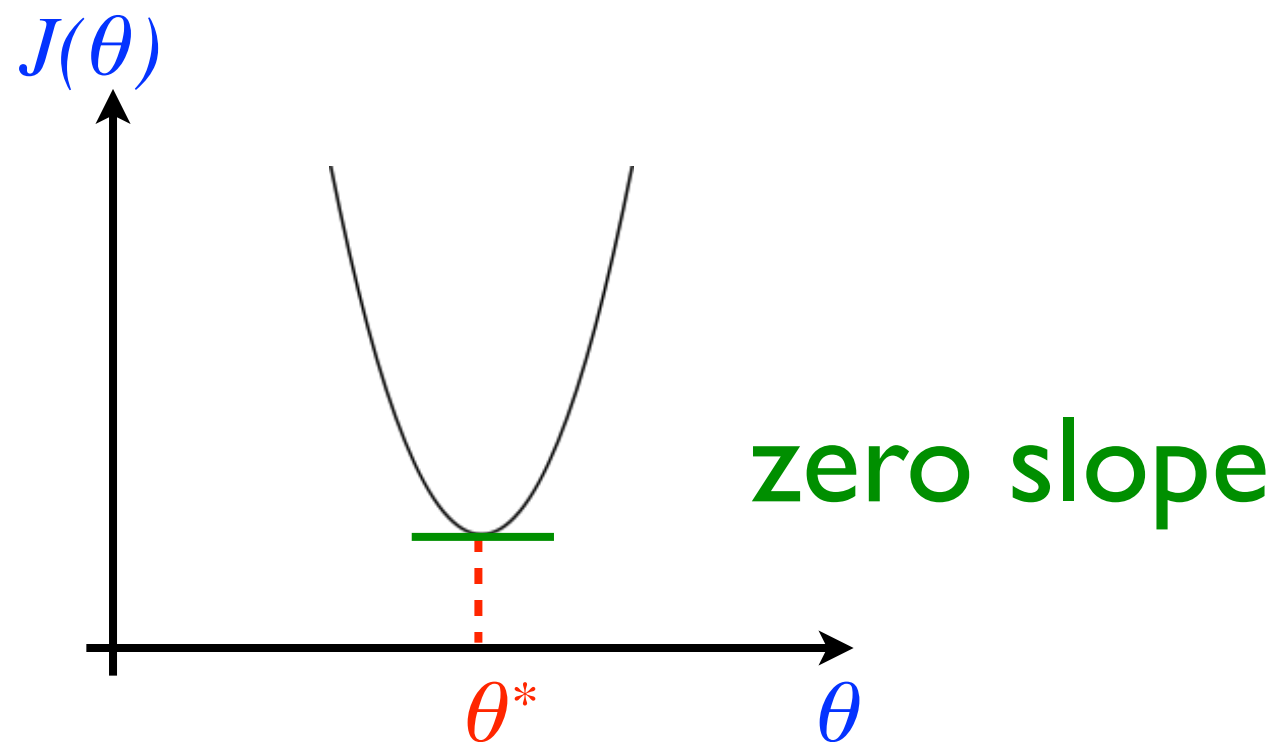
- The inner product  $\langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$  between the gradient of  $f$  at  $\mathbf{x}$  and any unit vector  $\mathbf{v} \in \mathbb{R}^n$  is the *directional derivative* of  $f$  in the direction of  $\mathbf{v}$  (i.e. the rate at which  $f$  changes at  $\mathbf{x}$  in the direction  $\mathbf{v}$ ).
- $\nabla f(\mathbf{x})$  points towards the direction of greatest increase of  $f$  at  $\mathbf{x}$ .
- Points such that  $\nabla f(\mathbf{x}) = \mathbf{0}$  are called **stationary points**.
- The gradient  $\nabla f(\mathbf{x})$  is orthogonal to the contour line passing through  $\mathbf{x}$ .

# The main property

At the optimum, the gradient is zero:  
the «slope»

$$\frac{\partial J}{\partial \theta}(\theta^*) = 0$$

Landscape of the loss function:



# Analytical solution

- **Sometimes** we can solve the equation (system) analytically to find the optimal  $\theta$ :

$$\frac{\partial J}{\partial \theta} = 0 \quad \left\{ \begin{array}{l} \frac{\partial J}{\partial \theta_1} = 0 \\ \frac{\partial J}{\partial \theta_2} = 0 \\ \vdots \\ \frac{\partial J}{\partial \theta_m} = 0 \end{array} \right.$$

...

$$\theta = \dots$$

- Examples of problem with analytical solutions:
  - Maximum likelihood to estimate the parameters of a Gaussian
  - Linear regression / Ridge regression

# Search algorithm

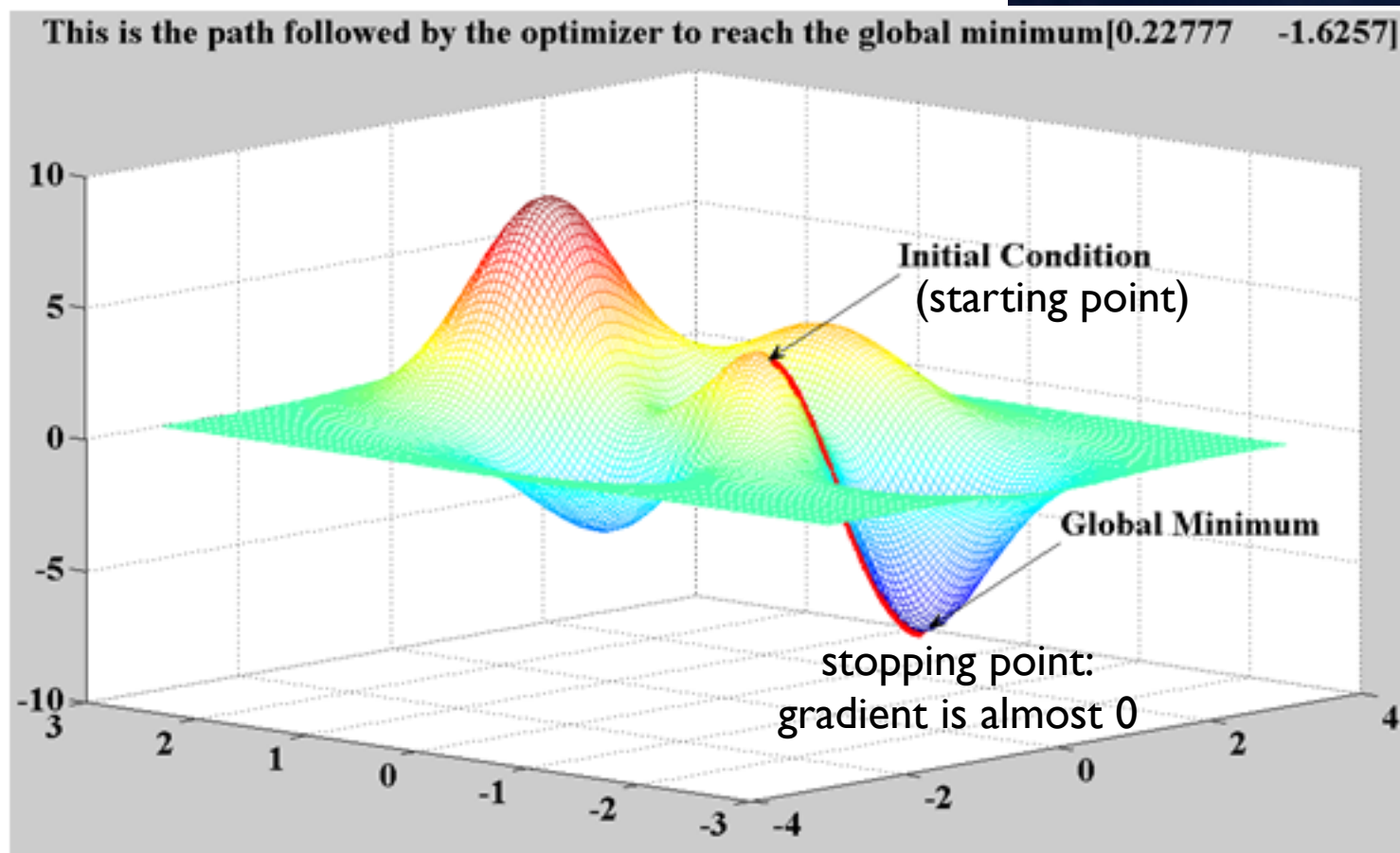
- **In most cases**, there is no analytical solution to the equation
$$\frac{\partial J}{\partial \theta} = 0$$
- In this case we can use an **iterative algorithm** to search for an optimal solution.
- Method 1: **exhaustive search!**  
try all possible values of  $\theta$  (or fine grid search) and keep the best one...  
**=> NOT POSSIBLE FOR HIGH-DIMENSIONAL  $\theta$  ( $m > 4$ ).**
- Method 2: **gradient descent!**



# Gradient descent

- Initialize  $\theta$  randomly (using an appropriate heuristic)
- Until convergence, repeat  $\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}(\theta)$

We update  $\theta$  by taking a step in the direction opposite to the gradient (direction of the steepest descent).





# Gradient descent

## learning rate

$$\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}(\theta)$$

- $\eta$  is a positive real number called the «*learning rate*» or «*step size*».  
It controls the size of the steps in parameter space (how much we update  $\theta$ ).
- This is a *crucial* hyper-parameter for the optimization procedure (needs to be chosen carefully).
- Often, we slightly decrease the learning rate at each iteration.  
Ex.: at iteration  $t$ , we set the learning rate to  $\eta(t) = \frac{\lambda}{t_0 + t}$   
Here  $t_0$  and  $\lambda$  are hyper-parameters (to be carefully chosen...).

# Gradient descent

## stopping criteria

$$\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}(\theta)$$

- We stop the gradient descent iterations when the updates leave  $\theta$  almost unchanged.
- That is, when the norm of the gradient gets smaller than some small threshold:  $\left\| \frac{\partial J}{\partial \theta}(\theta) \right\| < \epsilon$
- We may even want to stop earlier, before reaching the optimal point (for example to prevent overfitting and generalize better) => **early stopping** (use an other criterion to decide when to stop, for example using a validation set; more on that later...)

# Gradient descent

what solution do we get in the end?

Gradient descent converges to a point  
where the **gradient** is (almost) zero:

$$\frac{\partial J}{\partial \theta}(\theta) \approx 0$$

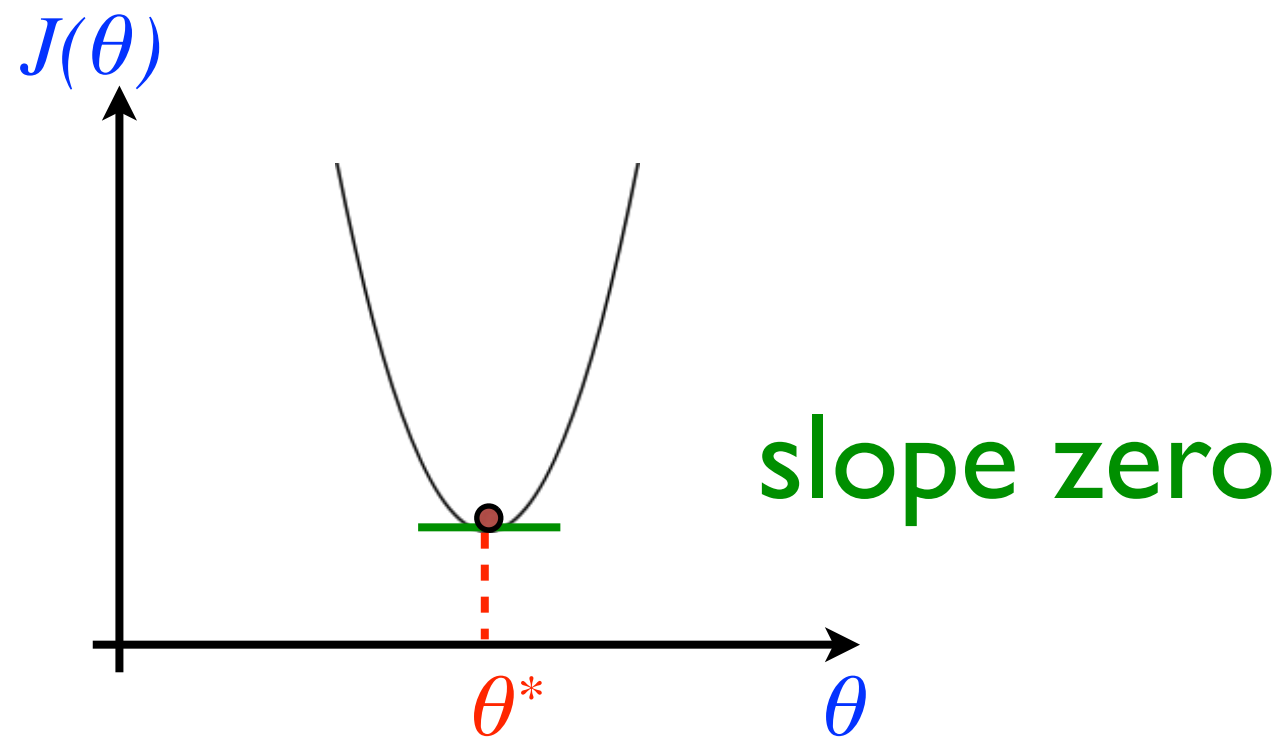
# Gradient descent

## Convex objective

Gradient descent converges to a point where the **gradient** is (almost) zero:

$$\frac{\partial J}{\partial \theta}(\theta) \approx 0$$

Landscape of the loss function:



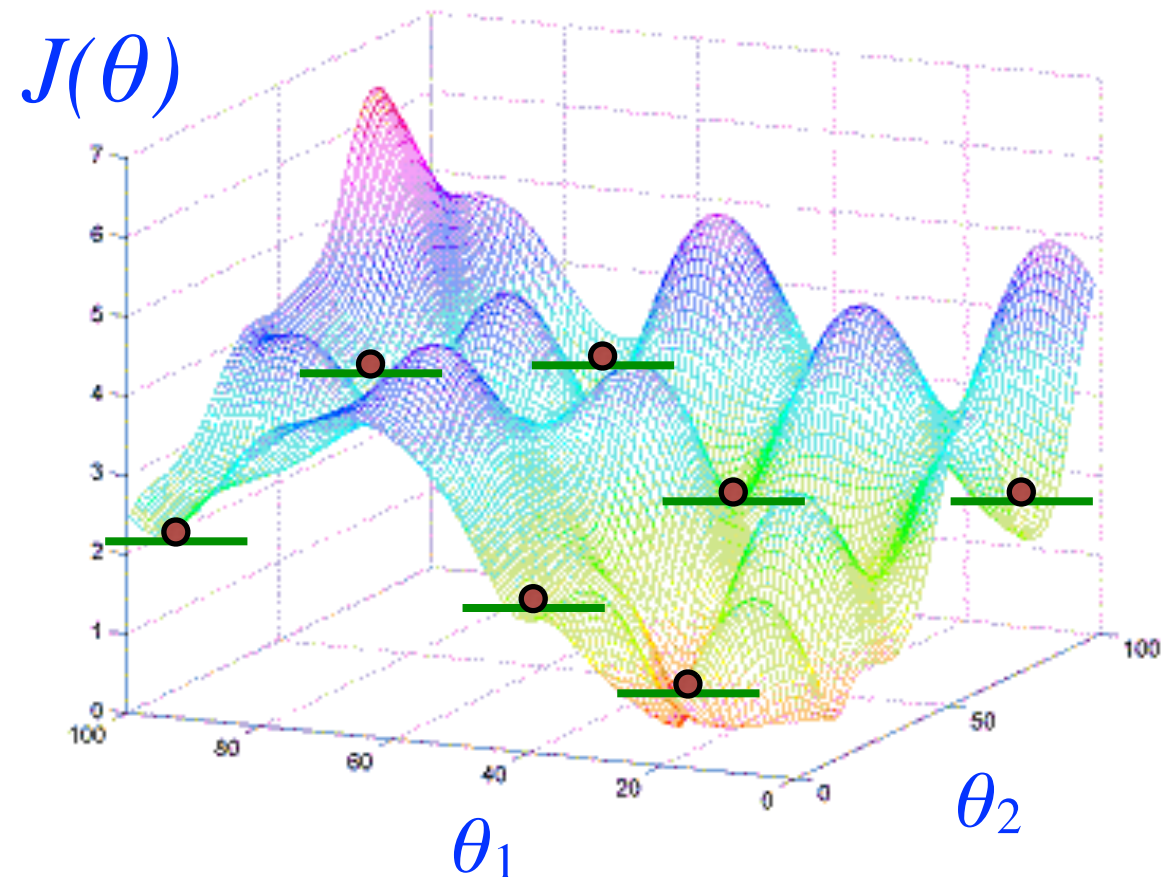
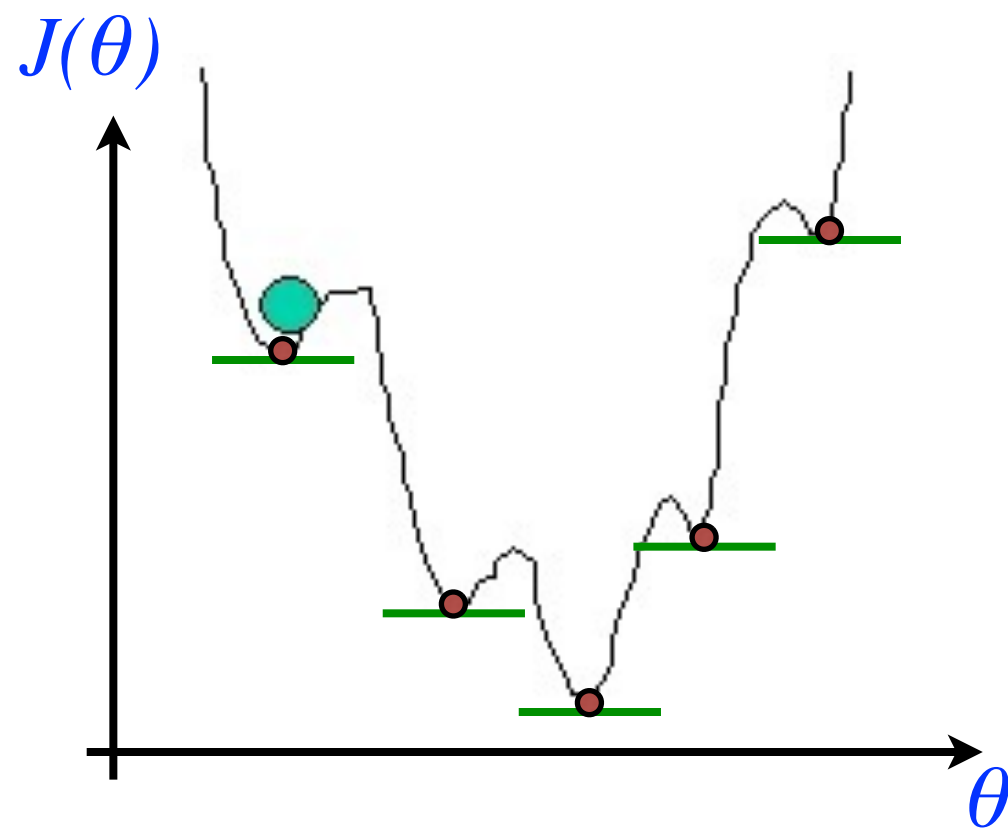
For a convex objective, we reach the **global minimum**

# Gradient descent

objective is **non-convex**

Gradient descent converges to a point where the gradient is (almost) zero:  $\frac{\partial J}{\partial \theta}(\theta) \approx 0$

For a **non convex objective function** there can be *many* such points: all *local* extrema (minima, maxima saddle points).



➡ In practice, we converge to a **local minimum** (rather than a global one) **which depends on the starting point**.

# Typical minimization objective for learning tasks

- In machine learning, the objective function to minimize is often a sum or a mean over the  $n$  examples of the training set  $D_n = \{z^{(1)}, \dots, z^{(n)}\}$  of a loss/cost function  $L$  (empirical risk):

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n L(z^{(i)}; \theta)$$

- **Remark:** minimizing the sum or the mean are equivalent (they have the same minima).



# Typical gradient for learning tasks

- Gradient of the mean = mean of the gradients:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n L(z^{(i)}; \theta)$$

$$\begin{aligned} \frac{\partial J}{\partial \theta}(\theta) &= \frac{\partial}{\partial \theta} \left( \frac{1}{n} \sum_{i=1}^n L(z^{(i)}; \theta) \right) \\ &= \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial \theta} L(z^{(i)}; \theta) \end{aligned}$$

# Gradient descent

## batch, mini-batch

- In «batch» gradient descent, we compute the mean of the gradients over all  $n$  examples in  $D_n$

$$\nabla(\theta) = \frac{\partial J}{\partial \theta}(\theta) = \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial \theta} L(z^{(i)}; \theta) = \operatorname{mean}_{z \in D_n} \left[ \frac{\partial}{\partial \theta} L(z; \theta) \right]$$

before each update of the parameters:  $\theta \leftarrow \theta - \eta \nabla(\theta)$

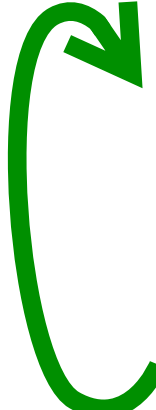
- In «mini-batch» gradient descent with batch size  $n'$ , approximate the mean of the gradients by computing the mean over only  $n' < n$  examples in  $D_n$

$$\nabla(\theta) = \operatorname{mean}_{z \in \text{minibatch}} \left[ \frac{\partial}{\partial \theta} L(z; \theta) \right] \approx \frac{\partial J}{\partial \theta}(\theta)$$

(these  $n'$  examples, different each time, are called a *mini batch*)

# Gradient descent

batch, mini-batch, stochastic/online

**Algo:** 

$$\nabla(\theta) = \text{mean}_{z \in \text{minibatch}} \left[ \frac{\partial}{\partial \theta} L(z^{(i)}; \theta) \right] \approx \frac{\partial J}{\partial \theta}(\theta) \text{ Gradient computation}$$
$$\theta \leftarrow \theta - \eta \nabla(\theta) \text{ Parameter update}$$

- Before each computation of  $\nabla$  the  $n'$  examples of the mini-batch should ideally be drawn randomly from  $D_n$ .
- But for efficiency, we often take examples *sequentially* in  $D_n$  to construct the mini-batches (starting with the first  $n'$  examples, then the following  $n'$ , etc., and we repeat).
- **Remark:** with  $n'=n$  we fall back onto the *batch* gradient descent.
- The case  $n'=1$  (we use only one examples for each gradient computation) is called **online/stochastic gradient descent**.

# Gradient descent variants

There exist a lot of optimization algorithms based on gradient descent:

- *Momentum* technique
- Conjugate gradient
- Second-order methods. (use information about the «curvature» of the objective by taking second-order derivatives into account: the Hessian)  
Ex: Newton's method.
- ...


All these are for **continuous parameters**  
and assume we can efficiently compute a «gradient».

# Newton's method


**Gradient:** vector of first-order derivatives

$$\nabla_J = \frac{\partial J}{\partial \theta} = \begin{pmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_m} \end{pmatrix}$$

Simple (batch) gradient descent:


$$\theta \leftarrow \theta - \eta \nabla_J(\theta)$$

**Newton's** method (batch):


$$\theta \leftarrow \theta - \eta H(\theta)^{-1} \nabla_J(\theta)$$

**Hessian:** matrix of second-order derivatives

$$H = \frac{\partial^2 J}{\partial \theta^2} = \begin{pmatrix} \frac{\partial^2 J}{\partial \theta_1^2} & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_m} \\ \frac{\partial^2 J}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_2^2} & \cdots & \frac{\partial^2 J}{\partial \theta_2 \partial \theta_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \theta_m \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_m \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_m^2} \end{pmatrix}$$

Not straight-forward for mini-batch

$H^{-1}$ : difficult and costly to compute

# Constrained optimization

- In what we saw, there were no constraints on the parameters
- Sometimes, we also want the parameters to satisfy one or more constraints (ex: positive, sum to 1, ...)
- Constrained optimization  $\Rightarrow$  more complex algorithms.  
Ex: linear/quadratic 'programming', ...