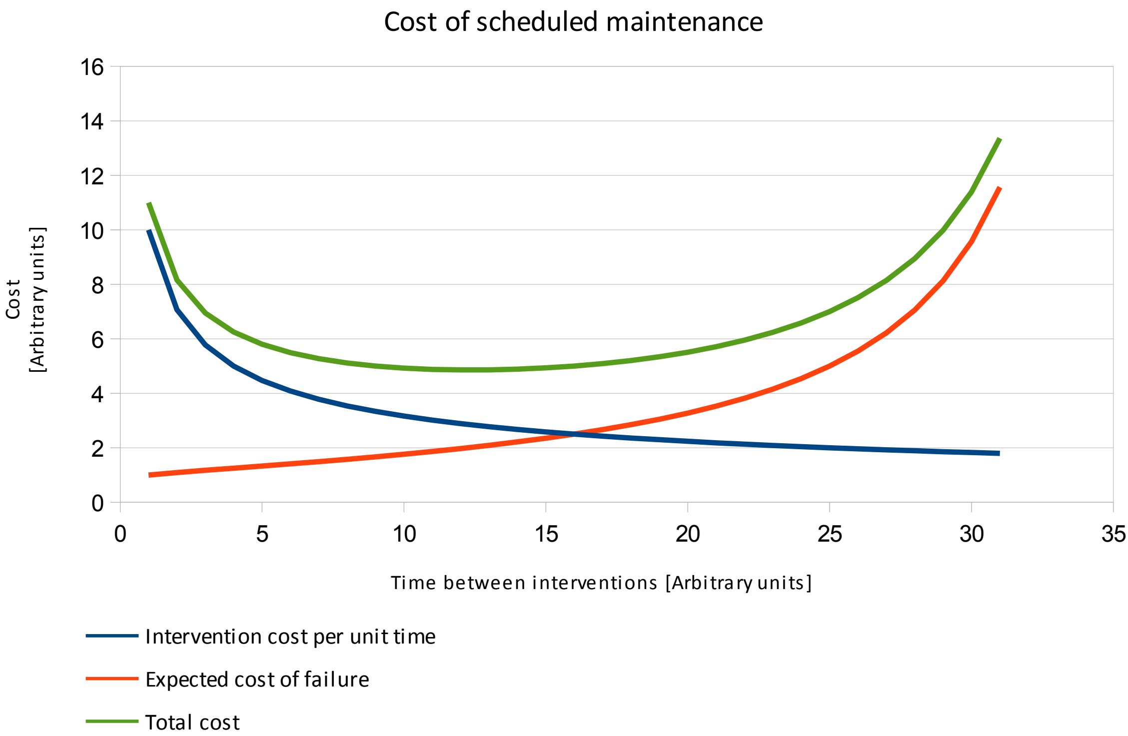
ML4 – Optimisation

Stefano Rovetta

A.y. 2023-2024

Motivation

- Maintenance interventions on a machine operating in a production line
- Scheduled maintenance interventions
- If done early, they reduce the probability of failure (but cost increases)
- If done late, they reduce the cost of interventions (but failures become more frequent)


Cost of scheduled maintenance
16
14
12
10
8
6
4
2
0
Cost
[Arbitrary units]
0
5
10
15
20
25
30
35
Time between interventions [Arbitrary units]
Intervention cost per unit time
Expected cost of failure
Total cost

- I have a rectangular sheet of wood from which I have to cut parts of non-rectangular shape. What is the layout that maximises the number of parts, or minimises the wasted material?
- Select the mix of goods (solid items) that maximises the value/cost ratio of a container to be shipped.
- Design a mechanical linkage that provides conversion of linear motion into a given 2-D trajectory, while minimising the number of links and joints.
- Decide the optimal number of years to keep your car before changing it.
- Given a set of different financial assets, each characterised by its expected return and its degree of uncertainty (variance), find the proportions of assets (portfolio mix) that maximises the total return.

Optimisation concepts and terms

The task of optimization:

Finding extrema of an **objective function** $J(\mathbf{w})$, where $J : S \subset \mathbb{R}^m \to \mathbb{R}$.

$S =$ feasible region, solution space, search space, feasible set

An **extremum** is a point $\mathbf{w}^* \in S$ that may be a **maximum** or **minimum**.

A point $\mathbf{w}^*$ is a minimum if there is a neighbourhood $R \subseteq S$, where the following holds:

$$J(\mathbf{w}) \geq J(\mathbf{w}^*) \quad \forall \mathbf{w} \in R$$

i.e.: A minimum is a point where J has a value smaller than in any other point in a given neighbourhood.

A point $\mathbf{w}^*$ is a maximum if it is a minimum of $-J$, or if $\leq$ is used.

Note: We will consider MINIMIZATION

- A minimum is **relative** if $R \subset S$ strictly
  i.e., there is some other point in S (outside R) where J has a smaller value than $J(\mathbf{w}^*)$.

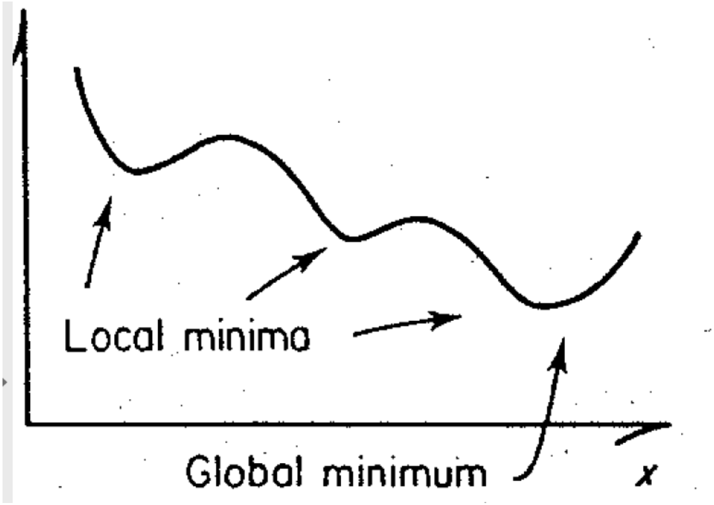
  A **relative minimum** is a minimum only in a neighbourhood (i.e. locally)

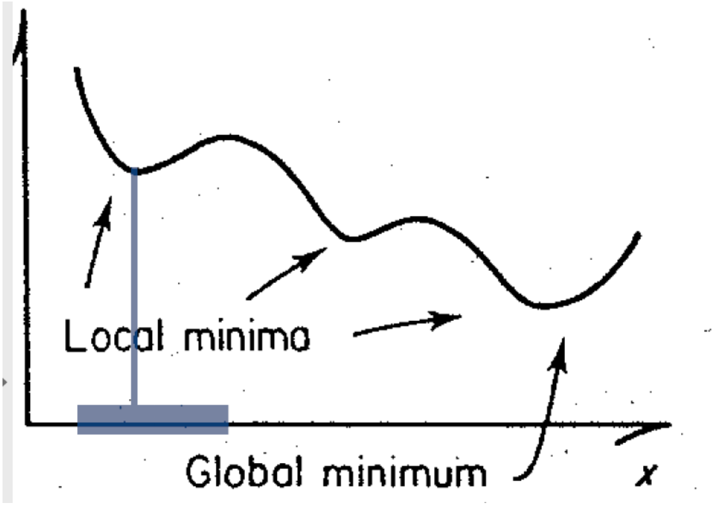
  So we also use **local minimum**

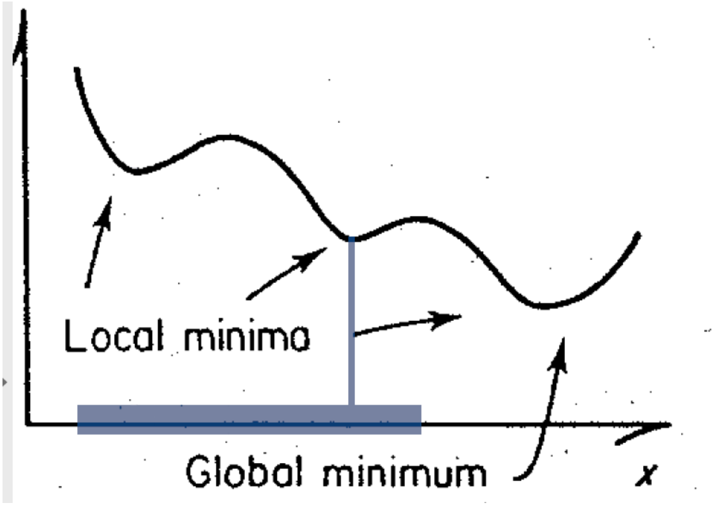
- A minimum is **absolute** if $R = S$.

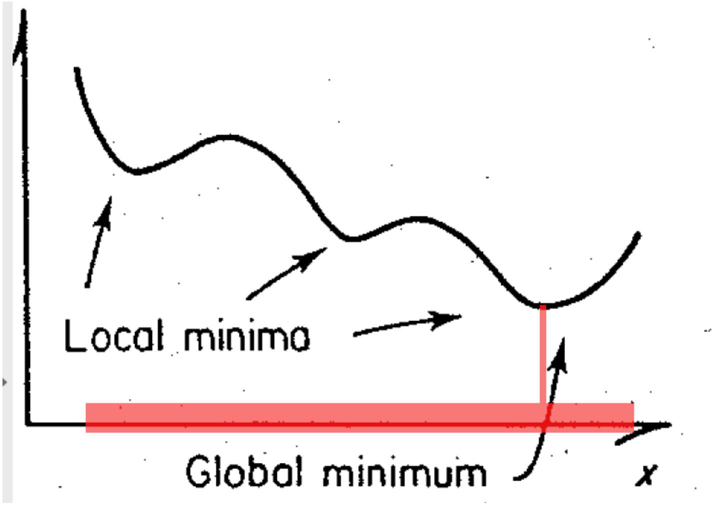
  Global minimum

- A minimum is **isolated** if we can draw a sphere around $\mathbf{w}^*$ with nonzero radius contained in R (i.e. if there are no other minima adjacent to it)


Local minima
Global minimum
x


Local minima
Global minimum
x


Local minima
Global minimum
x


Local minima
Global minimum
x

Objective functions in machine learning

- Often: **Cost function**, inversely related to how much we like a solution (the higher the cost, the less the quality of the solution)

- **Risk** (expected loss)

$$J(\mathbf{w}) = \int_{\mathscr{X}} \lambda\left(\mathbf{t}, \mathbf{y}(\mathbf{w})\right) p(\mathbf{x}) d\mathbf{x}$$

- **Regularised risk**
  Expected loss + terms penalising “irregular” or “extreme” or “weird” solutions
  Example: penalising high values of the parameters (**weight decay**)

$$J(\mathbf{w}) = \int_{\mathscr{X}} \lambda\left(\mathbf{t}, \mathbf{y}(\mathbf{w})\right) p(\mathbf{x}) d\mathbf{x} \ + \ \|\mathbf{w}\|$$

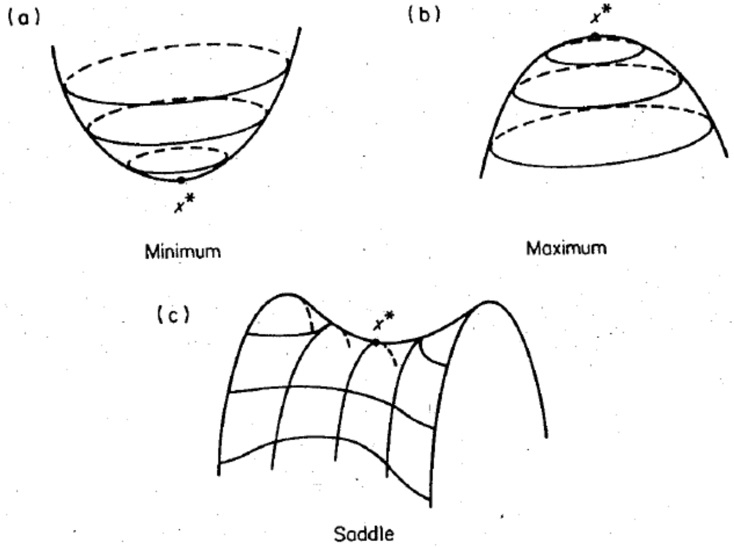
- **Likelihood** is a common objective that we have to **maximise** instead

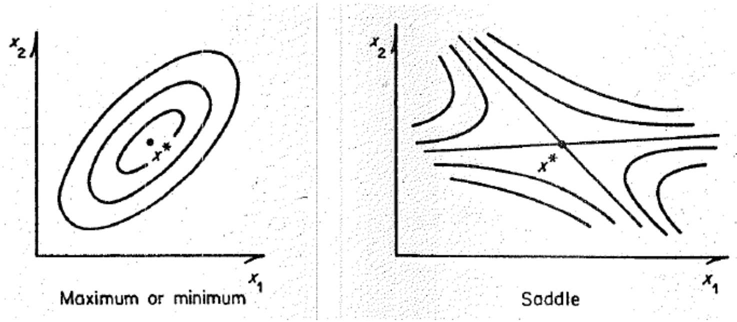
- An **optimal solution** is an extremum $\mathbf{w}^*$ of J

- An **optimal value** is $J(\mathbf{w}^*)$

- An **approximate solution with precision** ϵ is a point $\tilde{\mathbf{w}}$ whose value of the objective is close to an optimal solution: $|J(\mathbf{w}^*) - J(\tilde{\mathbf{w}})| < \epsilon$
  absolute value not necessary once the type of problem (min or max) is given

- A **feasible solution** is any point $\mathbf{w}$ which satisfies all hypotheses of the optimization problem


(a)
x*
Minimum
(b)
x*
Maximum
(c)
x*
Saddle


x_2
x^*
x_1
Maximum or minimum
x_2
x^*
x_1
Saddle

Convex sets

A set $S \subset \mathbb{R}^m$ is convex if and only if, for any $\theta \in [0,1]$,

$$\forall \mathbf{v}, \mathbf{w} \in S \Rightarrow \theta \mathbf{v} + (1-\theta)\mathbf{w} \in S$$

more generally if for any $\theta_1 > 0, \dots, \theta_n > 0$ such that $\sum_k \theta_k = 1$

$$\forall \mathbf{v}_1, \dots, \mathbf{v}_n \in S \Rightarrow \sum_k \theta_k \mathbf{v}_k \in S \qquad \textbf{Convex combination}$$

Properties:

- $\mathbf{v} \in \mathbb{R}^m$ (a single point) is convex
- $\emptyset = \{\ \}$ (the empty set) is convex
- $\mathbb{R}^m$ is convex

and if S_1 and S_2 are convex, then

- $S_1 \cap S_2$ is convex
- $S_1 \cup S_2$ IS NOT NECESSARILY convex ($o + o = \infty$)

Convex functions

A function $J : S \subseteq \mathbb{R}^m \rightarrow \mathbb{R}$ is convex if S is a convex set
and if $\forall \mathbf{v}, \mathbf{w} \in S$, and with $0 \leq \theta \leq 1$:

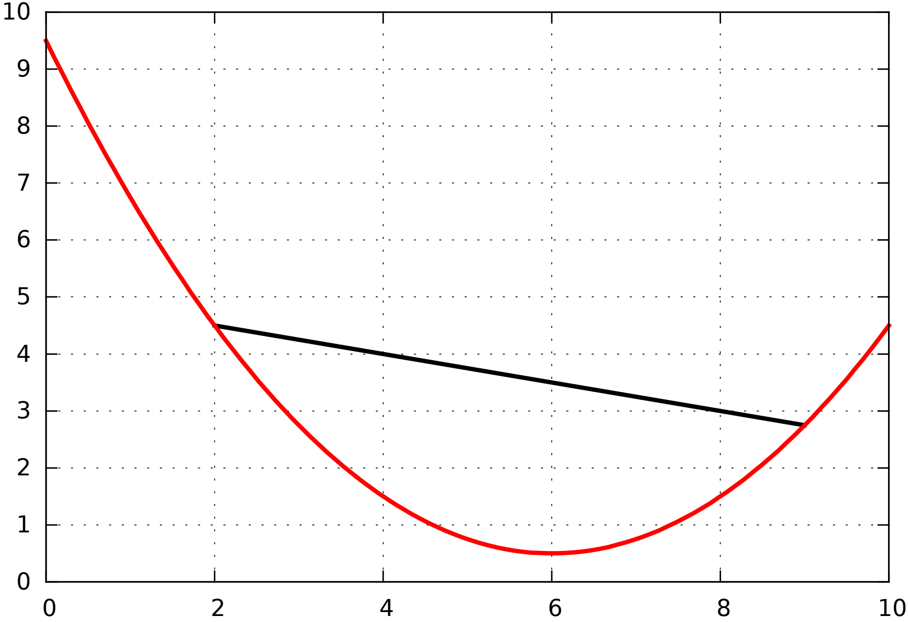
$$J(\theta \mathbf{v} + (1-\theta)\mathbf{w}) \leq \theta J(\mathbf{v}) + (1-\theta) J(\mathbf{w})$$

i.e., if its epigraph is a convex set

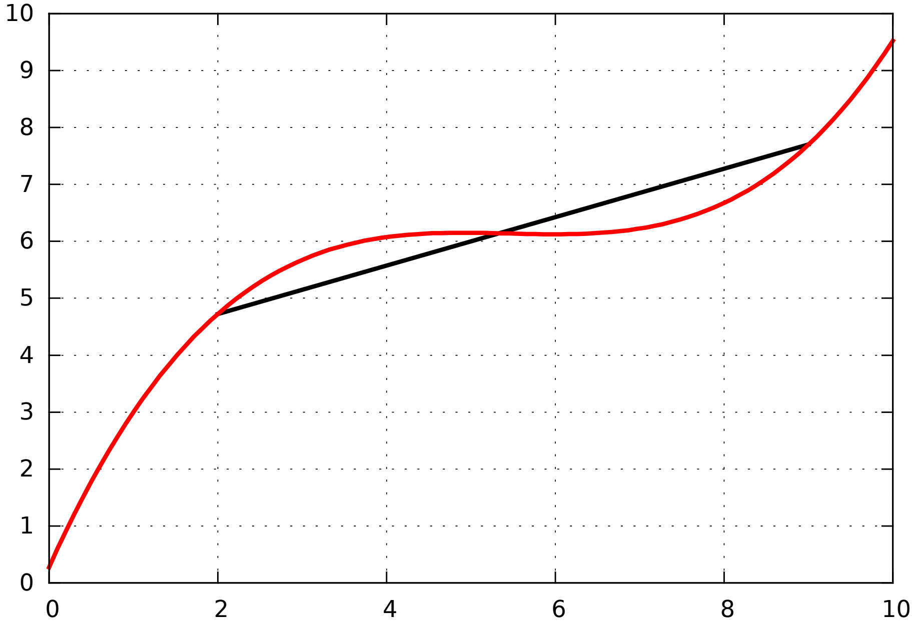
more generally if for any $\theta_1 > 0, \ldots, \theta_n > 0$ such that $\sum_k \theta_k = 1$

$$\forall \mathbf{v}_1, \ldots, \mathbf{v}_n \in S \Rightarrow J\left(\sum_k \theta_k \mathbf{v}_k\right) \leq \sum_k \theta_k J(\mathbf{v}_k)$$

J is concave if $-J$ is convex



10
9
8
7
6
5
4
3
2
1
0
0
2
4
6
8
10

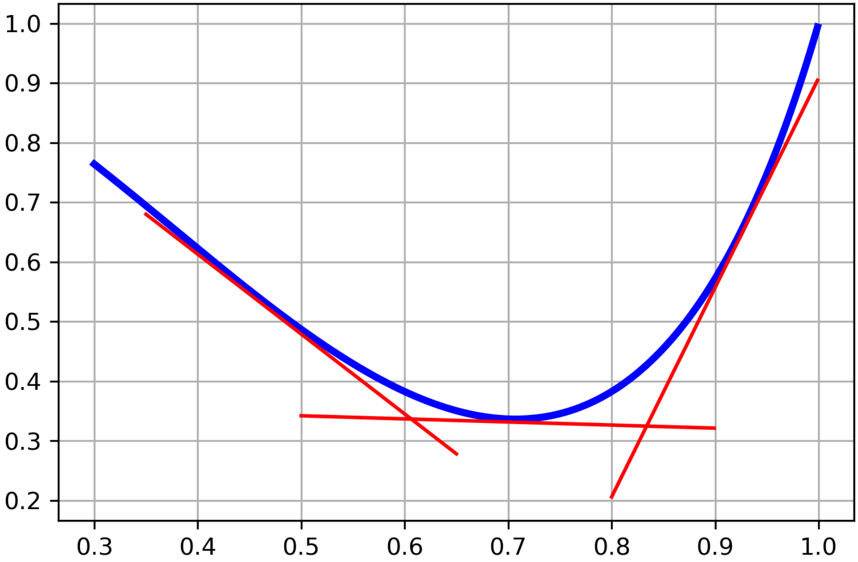


10
9
8
7
6
5
4
3
2
1
0
0
2
4
6
8
10

Convex functions, alternate (equivalent) definition

A function $J : S \subseteq \mathbb{R}^m \to \mathbb{R}$ is convex if S is a convex set and if $\forall \mathbf{v}, \mathbf{w} \in S$,

$$J(\mathbf{v}) \geq J(\mathbf{w}) + \nabla J(\mathbf{w}) \cdot (\mathbf{v} - \mathbf{w})$$



Why should we be interested in convexity?

Convexity is a property of a problem, not just of a loss function:

- Objective
- Learning machine on which the objective is computed (because we want to take derivatives)
- **The parameter space** – remember that a function is not convex if its domain is not convex!

Convexity is a good thing:

- Uniqueness of extrema
- Convergence of iterative algorithms

Not all problems are convex:

- Some learners guarantee a convex problem (e.g., SVM)
- For non-convex problems we usually **cannot be sure whether a minimum is absolute (global) or relative (local)**

Types of optimisation problems

Types of optimisation problems – By type of optimisation variables **w**

- **Discrete**: $S \in \mathbb{N}^m$, or more generally **countable variables**
  Example: Maximise the number of boxes placed in a storage area

- **Continuous**: $S \in \mathbb{R}^m$
  Example: Minimise the surface area of a box or container to be built in expensive materials (e.g. rare wood)

Types of optimisation problems – By feasible region S

- **Unconstrained**: $S \equiv \mathbb{R}^m$ (or $S \equiv \mathbb{N}^m$)
  Example: Find the coefficients of a linear-threshold classifier that minimise the average misclassification error ($\approx$ error probability).
  Feasible region $S = \mathbb{R}^{d+1}$

- **Constrained**: $S \in \mathbb{R}^m$ (or $S \in \mathbb{N}^m$)
  Example: Given N financial assets, each with unit cost C_i, find the portfolio (quantity w_i per each asset) that maximises the expected return, *subject* to:
  - $w_i \geq 0 \; \forall i : 1, \ldots, N$ (non-negative quantities)
  - $\sum_i C_i w_i \leq C_{\text{tot}}$ (the sum of invested money, sum of unit costs C_i times quantities w_i, must not exceed my total capital available!!)

  Feasible region S a subset of the all-positive region $\in \mathbb{R}^N$ (plus the origin)

Constraints are expressed by **constraint functions**:

- **Equality constraints:** $f_1(\mathbf{w}) = 0,\ f_2(\mathbf{w}) = 0,\ \ldots f_k(\mathbf{w}) = 0$
- **Inequality constraints:** $f_1(\mathbf{w}) \geq 0,\ f_2(\mathbf{w}) \geq 0,\ \ldots f_k(\mathbf{w}) \geq 0$

Both may be present in the same problem

Types of optimisation problems – By smoothness of the objective

- **Smooth**: The objective is at least differentiable, maybe twice
  Example: Find the coefficient of a linear regression model in the least squares sense ($\min J_{\text{mse}}$)

- **Nonsmooth**: The objective is not differentiable
  Example: Find the coefficients of a linear-threshold classifier that minimise the average misclassification error ($\approx$ error probability).

Obviously discrete $\mathbf{w} \Rightarrow$ nonsmooth objective (it is not even continuous!)

Types of optimisation problems – By shape of the objective

- **Linear**
  Example: Buy the best car
- **Convex**
  Example: Find the coefficient of a linear regression model in the least squares sense
- **Nonconvex**
  Example: Find the princing for goods in a shop, such that the expected income is maximised
- special case: **Quadratic**
  Example: Find the coefficient of a linear regression model in the least squares sense
  Easier to analyse but not guaranteed to be convex

Obviously linear $\Rightarrow$ constrained, otherwise the objective can grow or decrease indefinitely!
(Example: buy the best car with price ≥ 0 and price $\leq$ my bank account balance)

WARNING

The type of the loss alone **is not sufficient to describe the type of problem!**

$$\lambda(t, y) = \lambda(t, y(\mathbf{w}))$$

The loss as a function of the parameters is a **composite function**

loss $\longrightarrow$ a function of t and y
$y \longrightarrow$ a function of $\mathbf{w}$

Examples:

Loss	**Model**		**Problem**
$\|t-y\|^2$	Linear in $\mathbf{w}$: $y(\mathbf{w}) = \mathbf{w} \cdot \mathbf{x}$	$\longrightarrow$	Quadratic
$\|t-y\|^2$	Convex in $\mathbf{w}$: $y(\mathbf{w}) = \log(1 + e^{\mathbf{w} \cdot \mathbf{x}})$	$\longrightarrow$	Convex, but not quadratic
$\|t-y\|^2$	Non-convex in $\mathbf{w}$: $y_i(\mathbf{w}) = e^{g_i(\mathbf{x},\mathbf{w})} / \sum_j e^{g_j(\mathbf{x},\mathbf{w})}$, $i = 1 \dots c$	$\longrightarrow$	Non-convex

The main types of problems in machine learning

1. Unconstrained, smooth, quadratic, convex problems
   Example: least squares for linear models

2. Unconstrained, smooth, general (non-convex) problems
   Example: all non-linear models, e.g. neural networks

3. Constrained, smooth, quadratic problems
   Example: support vector machines, in general "kernel" methods

Conceptual tools

The gradient

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \dfrac{\partial J(\mathbf{w})}{\partial w_1} \\[2ex] \dfrac{\partial J(\mathbf{w})}{\partial w_2} \\ \vdots \\ \dfrac{\partial J(\mathbf{w})}{\partial w_m} \end{bmatrix}$$

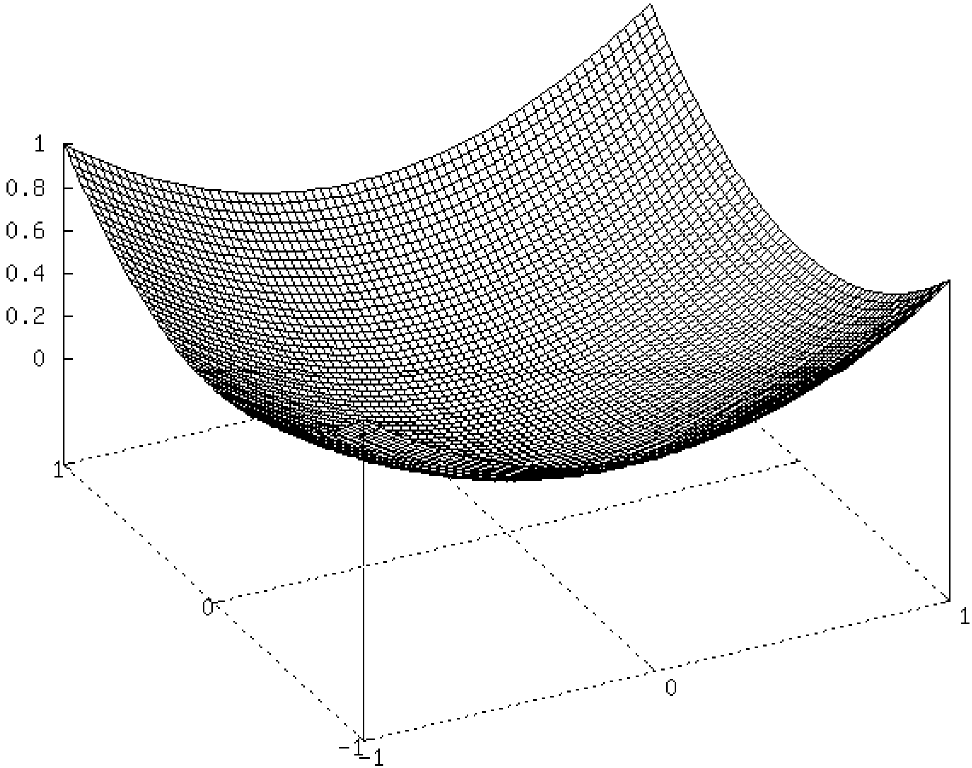
Other notation:

$\nabla_{\mathbf{w}} J(\mathbf{w})$ to highlight differentiation w.r.t. $\mathbf{w}$

The gradient is a vector field (vector function of vector argument)

- Derivative → rate of growth of a function of a scalar variable
- Negative sign → decreasing

- Gradient *length (norm)* → rate of maximum growth
- *Direction* → *direction of maximum growth*


1
0.8
0.6
0.4
0.2
0
1
0
-1
-1
0
1

The gradient indicates the direction of maximum increase, and moving in the opposite direction $-\nabla J(\mathbf{w})$ we achieve the *maximum rate of decrease*.

This observation is very useful in optimization techniques.

Hessian matrix

or simply Hessian

$$H_J(\mathbf{w}) : \mathbb{R}^m \to \mathbb{R}^m \times \mathbb{R}^m \qquad \text{s.t.} \qquad H_J(\mathbf{w})_{ij} = \frac{\partial^2 J(\mathbf{w})}{\partial w_i \partial w_j}$$

Other notations:

$$H$$

$$\nabla_{\mathbf{w}} \nabla_{\mathbf{w}} J(\mathbf{w}) \qquad \text{or} \qquad \nabla^2_{\mathbf{w}} J(\mathbf{w})$$

$$\nabla \nabla J(\mathbf{w}) \qquad \text{or} \qquad \nabla^2 J(\mathbf{w})$$

The Hessian matrix can be thought of as a list of m vectors

$$\mathbf{h}_i = \nabla \left(\frac{\partial J(\mathbf{w})}{\partial w_i} \right)$$

Derivative is a linear operator and the order of differentiation does not matter:

$$\frac{\partial^2 J(\mathbf{w})}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i} \left(\frac{\partial J(\mathbf{w})}{\partial w_j} \right) = \frac{\partial}{\partial w_j} \left(\frac{\partial J(\mathbf{w})}{\partial w_i} \right)$$

$\Rightarrow H$ is a symmetric matrix.

$$H = \begin{pmatrix} \frac{\partial^2 J}{\partial w_1^2} & \frac{\partial^2 J}{\partial w_1 \partial w_2} & \frac{\partial^2 J}{\partial w_1 \partial w_3} & \cdots & \frac{\partial^2 J}{\partial w_1 \partial w_m} \\ \frac{\partial^2 J}{\partial w_2 \partial w_1} & \frac{\partial^2 J}{\partial w_2^2} & \frac{\partial^2 J}{\partial w_2 \partial w_3} & \cdots & \frac{\partial^2 J}{\partial w_2 \partial w_m} \\ \frac{\partial^2 J}{\partial w_3 \partial w_1} & \frac{\partial^2 J}{\partial w_3 \partial w_2} & \frac{\partial^2 J}{\partial w_3^2} & \cdots & \frac{\partial^2 J}{\partial w_2 \partial w_m} \\ & & \vdots & & \\ \frac{\partial^2 J}{\partial w_m \partial w_1} & \frac{\partial^2 J}{\partial w_m \partial w_2} & \frac{\partial^2 J}{\partial w_m \partial w_3} & \cdots & \frac{\partial^2 J}{\partial w_m^2} \end{pmatrix}$$

Taylor polynomials

The Taylor polynomial of degree 2 for a scalar function $J(w)$ centered around w_0:

$$J(w) \approx J(w_0) + J'(w)\,|_{w=w_0}\,(w - w_0) + \frac{1}{2}J''(w)\,|_{w=w_0}\,(w - w_0)^2$$

Brook Taylor, 1685-1731

Multi-dimensional Taylor polynomials

Equivalent formula when $\mathbf{w} \in \mathbb{R}^m$, centered around $\mathbf{w}_0$:

$$J(\mathbf{w}) \approx J(\mathbf{w}_0) + \nabla J(\mathbf{w})\,\big|_{\mathbf{w}=\mathbf{w}_0} (\mathbf{w} - \mathbf{w}_0) + \frac{1}{2} (\mathbf{w} - \mathbf{w}_0)^\mathsf{T} H\,\big|_{\mathbf{w}=\mathbf{w}_0} (\mathbf{w} - \mathbf{w}_0)$$

Characterizing minima

There are conditions for determining whether a feasible solution $\mathbf{w}$ is a minimum

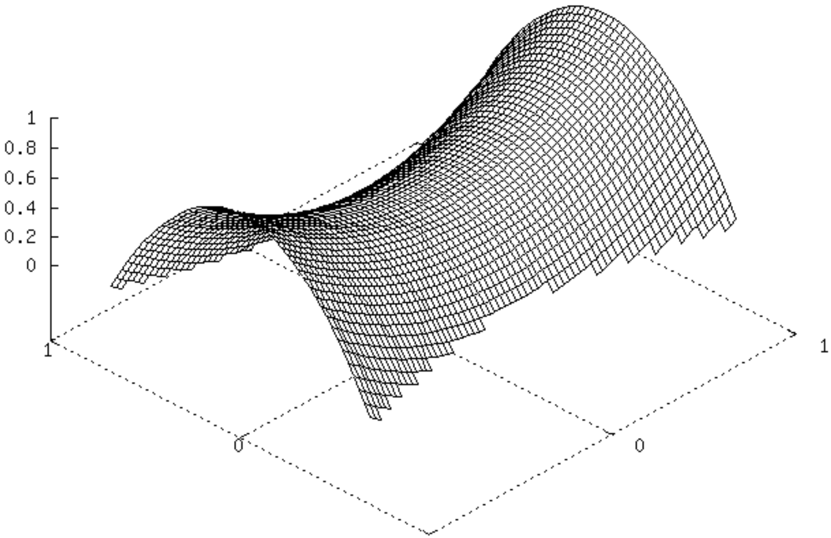
- One conditions is true for all extrema (**necessary** condition of minimum, but not sufficient)
- One condition is true only for minima (**necessary and sufficient** condition of minimum)
- In the case of convex objectives, the necessary condition becomes sufficient

Characterizing minima

Necessary first-order minimum condition:

$$\nabla J(\mathbf{w}^*) = 0$$

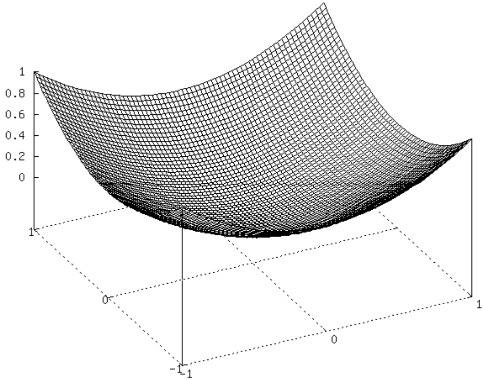
This condition characterizes all points which are local minima, but also local maxima or *saddle points* (points which are minima along one direction and maxima along another direction).



A saddle.

The case of convex objective

The first-order condition is also **sufficient**!



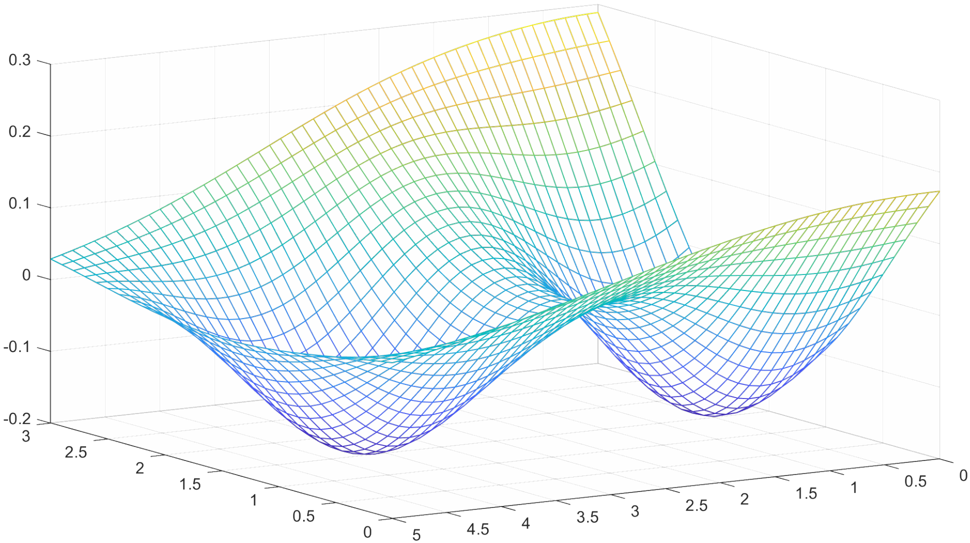
An elliptic paraboloid (quadratic function with rotational symmetry) is a convex function

Locally convex cost function

General, sufficiently smooth functions that are non-convex are nevertheless convex in a neighbourhood of $\mathbf{w}^*$
The first-order condition is then a **necessary and sufficient condition of <u>local</u> minimum**.
Other local minima belong to different neighborhoods ("basins").

Need methods for searching in different basins:

- Multiple re-starts from different (random) initial points
- Occasional random "jumps" in addition to regular gradient descent
- Regularisation that makes basins appear gradually



An objective function that has two “basins”. The bottom of each basin is a local minimum.

Definiteness and semidefiniteness

A matrix A is positive semidefinite (written $A \succeq 0$) if $\forall \mathbf{v} \in \mathbb{R}^m$

$$\mathbf{v}^\mathsf{T} A \mathbf{v} \geq 0$$

It is positive definite (written $A \succ 0$) if $\forall \mathbf{v} \in \mathbb{R}^m$

$$\mathbf{v}^\mathsf{T} A \mathbf{v} > 0$$

(similarly for "negative (semi)definite")

Hessian generalizes the second derivative
"Positive semidefinite" generalises "non-negative", "Positive definite" generalises "positive"

Conditions of convexity

$J(\mathbf{w})$ is convex if $H = \nabla^2 J(\mathbf{w})$ is positive semidefinite everywhere (for all $\mathbf{w}$)

$J(\mathbf{w})$ is locally convex around a point $\mathbf{w}_0$ if H is positive semidefinite at $\mathbf{w}_0$

To check whether a given point $\mathbf{w}$ is a minimum...

1. Necessary conditions of extremum: gradient $= 0$ in the point of interest
2. Necessary and sufficient condition of local convexity: Hessian $\succeq 0$ at the point of interest
3. Sufficient conditions of local minimum: gradient $= 0$ and Hessian $\succeq 0$ at the point of interest

4. Necessary and sufficient condition of convexity: Hessian $\succeq 0$ everywhere
5. Sufficient conditions of global minimum: gradient $= 0$ and Hessian $\succeq 0$ everywhere

Basic optimisation methods

The “oracle”

Important assumption:

In general, **we don’t know everything about the objective function**
(realistic problems are too complex!)

Example: The objective value is the result of a physical experiment

Example: The objective value is the performance of executing a program that is not open source

Example: The objective function depends on the data (training set) in a complex way

The “oracle”

Given a point $\mathbf{w}$ in the optimisation feasible space S,
we can query an **oracle**, a *black-box* system
that can only provide information about some of these quantities:

- The value of the objective at $\mathbf{w}$, $\quad J(\mathbf{w})$
- The gradient of the objective at $\mathbf{w}$, $\quad \nabla J(\mathbf{w})$
- The Hessian of the objective at $\mathbf{w}$, $\quad H = \nabla^2 J(\mathbf{w})$

Types of optimization algorithms

- **Oracle of order 0:** we only know
  - $J(\mathbf{w})$ computed values of the objective
- **Oracle of order 1:** we know
  - $J(\mathbf{w})$
  - $\nabla J(\mathbf{w})$ computed values of the gradient
- **Oracle of order 2:** we know
  - $J(\mathbf{w})$
  - $\nabla J(\mathbf{w})$
  - and also H computed values of the Hessian

A basic concept: Relaxation methods

Relaxation sequence:

$$\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \ldots$$

such that

$$J(\mathbf{w}_0) \geq J(\mathbf{w}_1) \geq J(\mathbf{w}_2) \ldots$$

Approximation:
To generate a relaxation sequence, we employ **local approximations** to the objective, that are easier to deal with than the objective itself.

We will use **Taylor polynomials** of first and second order as our approximating functions.

A note about jargon:

- Where the mathematician see a sequence…
- …the programmer sees a **loop** (iteration)

So “relaxation sequence” ≡ “iterative algorithm”

The general relaxation algorithm

- Iteratively descend toward the minimum

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \Delta\mathbf{w}_i$$

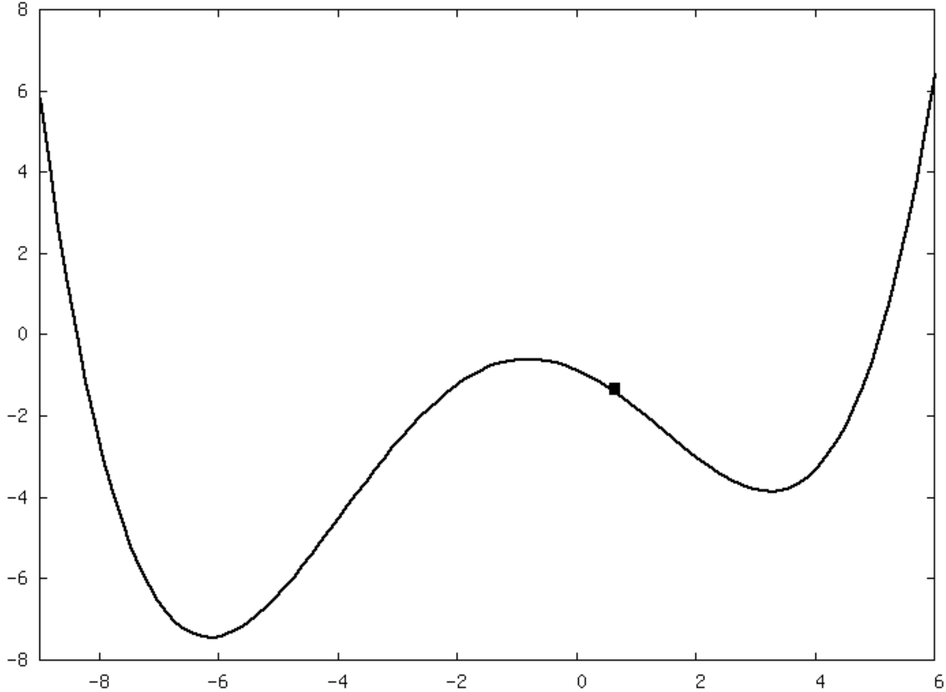
by taking *additive* steps $\Delta\mathbf{w}_i$ chosen so as to decrease the objective J

- Stop iterating when a pre-defined **stopping criterion** is satisfied
Examples of stopping criteria:

$\nabla J(\mathbf{w}_i) < \epsilon$ (gradient ≈ 0)

$J(\mathbf{w}_i) - J(\mathbf{w}_{i+1}) < \epsilon$ (improvement too small)

$i == N_{\max}$ (max. number of iterations reached)


8
6
4
2
0
−2
−4
−6
−8
−8
−6
−4
−2
0
2
4
6

Oracle of order 0: Direct search methods

We only know

- $J(\mathbf{w})$

- **NO TAYLOR EXPANSION AVAILABLE!**
- Only **direct search** techniques are possible:
  find promising points by using (meta)heuristics
  - Simulated annealing
  - genetic algorithms
  - particle swarm optimization
  - ant colony optimization
  - ...

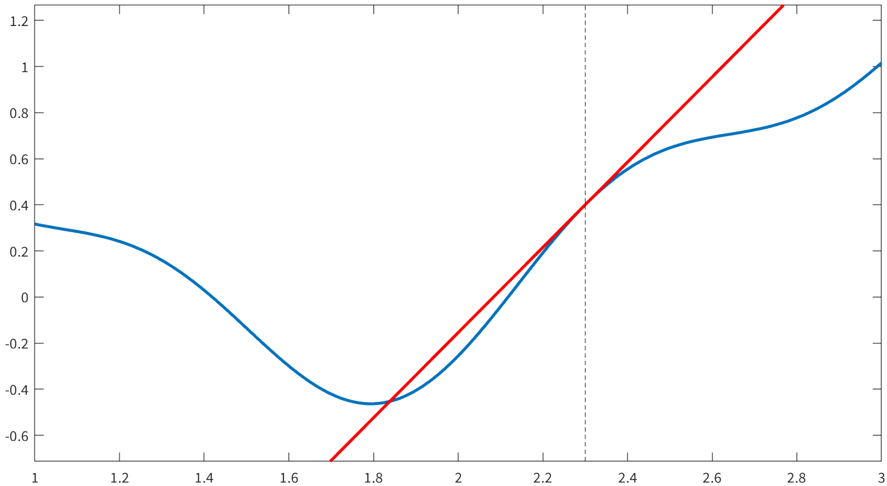
  or by branch-and-bound methods, or by random search...

PRO: simple, and guaranteed to find the GLOBAL extrema
CON: ...only if infinite time is available!

Oracle of order 1: Gradient methods

The objective is locally approximated with its **first-order Taylor polynomial** around the current point

$$J(\mathbf{w}) \approx J(\mathbf{w}_i) + \nabla J(\mathbf{w}_i) \cdot (\mathbf{w} - \mathbf{w}_i)$$



Gradient descent algorithm

1. Initialize: set $i = 0$; select some $\mathbf{w}_0 = \mathbf{w}_{\text{start}}$
2. Compute $\nabla J(\mathbf{w}_i)$
3. Select the appropriate step size η_i
4. Compute the step $\Delta\mathbf{w}_i = -\eta_i \nabla J(\mathbf{w}_i)$
5. Perform step $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i + \Delta\mathbf{w}_i$
6. Compute convergence test. If necessary, iterate from step 2.

```
epsilon = 1e-3;
maxiter = 100;
w = w0;
etavals = logspace(log10(.75),log10(.075),maxiter);
i = 1;
G = grad_J(w);
while norm(G)>=epsilon && l < maxiter
   eta = etavals(l);
   w = w - eta*G;
   G = grad_J(w);
end
```

Step size strategies

- Constant η, $\eta_i = \eta_0 \quad \forall i$
- Predefined sequence of η_i
- Full relaxation, or "line search":

  $\eta_i = \arg\min_\eta J(\mathbf{w}_i) + \eta \nabla J(\mathbf{w}_i)$ $\longleftarrow$ N.B. this is a one-dimensional function of η
- Adaptive η

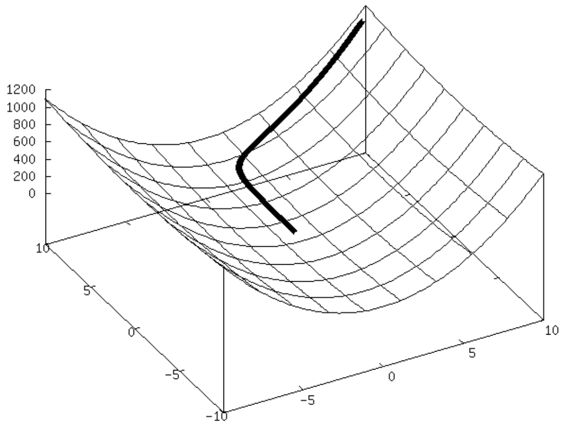
Features of gradient descent

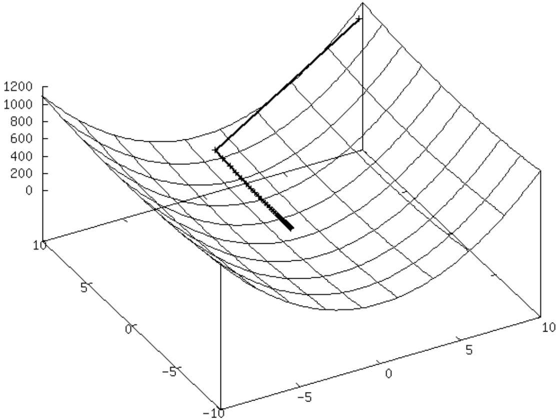
Pros

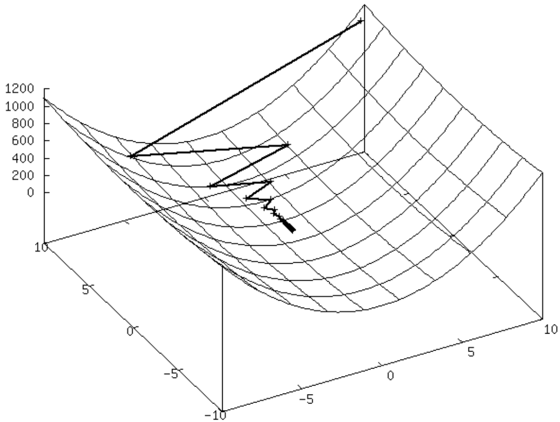
- Simple!

Cons

- Unnecessarily slow convergence (always directed exactly as the negative gradient)


1200
1000
800
600
400
200
0
10
5
0
-5
-10
-5
0
5
10

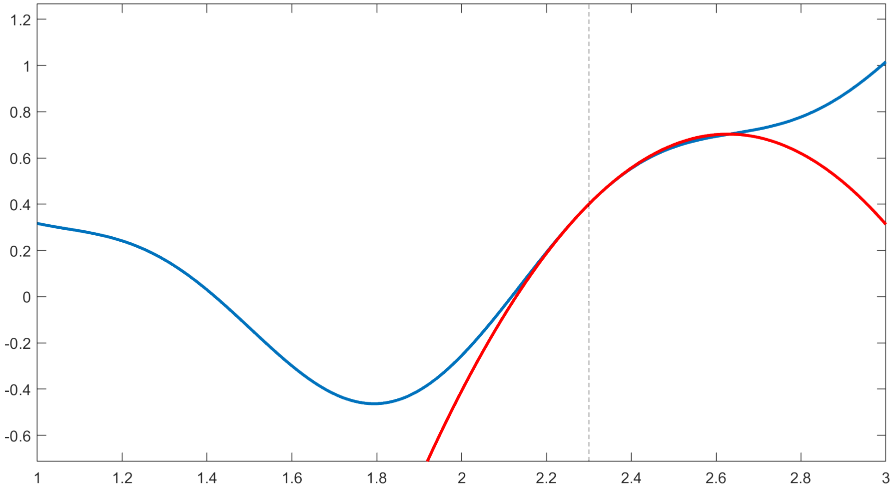

1200
1000
800
600
400
200
0
10
5
0
-5
-10
-5
0
5
10


1200
1000
800
600
400
200
0
10
5
0
-5
-10
-5
0
5
10

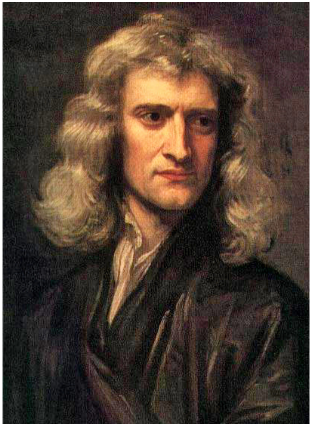
Oracle of order 2: Second-order methods

The objective is locally approximated with its **second-order Taylor polynomial** around the current point

$$J(\mathbf{w}) \approx J(\mathbf{w}_i) + \nabla J(w_i) \cdot (\mathbf{w} - \mathbf{w}_i) + (\mathbf{w} - \mathbf{w}_i)^\intercal H(w_i)(\mathbf{w} - \mathbf{w}_i)$$



Newton-Raphson method



Isaac Newton (1642-1726)

?

Joseph Raphson (1648?-1715?)

Iterative relaxation method for finding **zeros** of functions.

Newton-Raphson method

To find a zero of a function $g(x)$, iterate as follows:

- Start at x_0
- At each step τ compute the update as

$$x_{\tau+1} = x_\tau - \frac{g(x_\tau)}{g'(x_\tau)}$$

The update finds exactly the zero of the **1st order Taylor approximation of g in x_τ**
...but the Taylor approximation is not g, so we repeat

Newton-Raphson method for optimization

Necessary 1st order condition of minimum:

Minimum of J = zero of J'

To find a minimum of a function $J(x)$, set $g(x) = J'(x) = \dfrac{\mathrm{d}J(x)}{\mathrm{d}x}$
and then apply the Newton-Raphson method:

- Start at x_0
- At each step τ compute the update as

$$x_{\tau+1} = x_\tau - \frac{J'(x_\tau)}{J''(x_\tau)}$$

Newton-Raphson method for multidimensional optimization

To find a minimum of a function $J(\mathbf{w}) : \mathbb{R}^m \to \mathbb{R}$,

$$\mathbf{w}_{\tau+1} = \mathbf{w}_\tau - [H_J(\mathbf{w}_\tau)]^{-1} \nabla J(\mathbf{w}_\tau)$$

PRO: simple, much faster than gradient descent
CON: Need to compute the Hessian ($O(m^2)$ space complexity)
and to invert it ($O(n^q)$ time complexity, with $2 < q \leq 3$ depending on the algorithm)

Quasi-Newton methods

The most popular methods.

They use a **first-order oracle** (gradient only) to approximate **second-order information**.

Rationale:
Second derivative $\approx$ (first derivative at $\mathbf{w} + \mathbf{v}$ − first derivative at $\mathbf{w}$) / $\|\mathbf{v}\|$

How do methods work in different problem types?

- Most problems in machine learning are **smooth** $\quad \Rightarrow \quad$ **first- and second-order oracles**
- Many are unconstrained
- Many are non-convex

Quadratic (convex) problems

- Encountered in linear regression
- First-order (gradient) and second-order (Newton, quasi-Newton) are efficient

General (non-convex) problems

- Encountered in neural networks and in most non-trivial models
- First-order (gradient) and quasi-second-order can only **find local minima**
- Newton-type are usually impractical because H is too large
- To search for **better** minima, many optimisation cycles are attempted **starting from different initial conditions**
- This is a hybrid between random search (zero-order) and higher-order methods
- Several heuristics to speed up convergence exist:
    - adaptive step size (Vogl, SuperSAB, . . . many others)
    - momentum
    - adaptive non-isotropic step size (AdaGrad, RMSprop, AdaDelta)
    - adaptive momentum (Adam, NAG-Nesterov's Accelerated Gradient, AdaMax, N-Adam)

Example of acceleration heuristic

Adaptive step size

The learning step size, or learning rate, η is changed according to some heuristics

Example:

```
if J increases:
   reduce eta
   if J increases more than a quantity T:
       cancel step and go back to previous iteration
else:
   increase eta
```

Example of acceleration heuristic

Momentum

The current step also includes a fraction $\alpha < 1$ of the previous one

Example:

$$\Delta \mathbf{w}_i = -\eta \nabla J(\mathbf{w}_i) + \alpha \Delta \mathbf{w}_{i-1}$$

Helps traversing uninteresting regions where the gradient is small (“plateaus”, saddle points) by keeping a “memory” (inertia) of the previous motion.

Constrained quadratic problems

- Encountered in support vector machines
- Methods for **constrained optimisation** are used to construct a dual problem, an unconstrained quadratic convex one
- Then methods for quadratic optimisation are used (first- and quasi-second order)