ALGORITHMS IN AI & DATA SCIENCE 1 (AKIDS 1)

Numerical Optimization

Prof. Dr. Goran Glavaš

25.1.2024

Content

- Calculus Basics
- Gradient-Based Optimization
  - Newton Method
  - Gradient Descent
- Search-Based Optimization
  - Genetic Algorithm

Recap: Discrete Constrained Optimization

Discrete Constrained Optimization Problems

In **discrete constrained optimization**, we search for an **optimal state** in large space of possible states. Each state **X** can be seen as consisting of n variables **X** = $x_1, x_2, \ldots, x_n$, each with a corresponding domain $D_1, D_2, \ldots, D_n \subseteq \mathbb{Z}$ (whole numbers). The optimal state is the one that maximizes/minimizes the **objective function** $f: D_1 \times \cdots \times D_n \rightarrow \mathbb{R}$. Finally, the constraints $C_1, \ldots, C_m$, with $C_i \subseteq D_1, D_2, \ldots, D_n$ define the subsets of the state space that encompass **valid solutions** to the problem

- Optimal state (or the state with the best f that was found) is the **solution**
- No path between start and goal state – often there isn't a clear start state
- We're not making moves like in classic **SSS** problems, just **searching for the best possible solution** over a very large space of candidate solutions

Numerical Optimization

- In **numerical optimization**, instead of a space of discrete states, we're optimizing (minimizing or maximizing) some **real-valued function**

Numerical Optimization

Numerical optimization refers to optimizing **real-valued functions** $f(\mathbf{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}$, $\mathbf{x} = x_1, x_2, ..., x_n \in \mathbb{R}$. This means finding values $x_1, x_2, ..., x_n$ for which f obtains the **minimal** or **maximal value**. The input variables $x_1, x_2, ..., x_n$ may be subject to constraints (e.g., linear inequality constraints such as $x_i \geq m$ or non-linear constraints such as $x_i^2 - x_j^2 < m$) in which case we are dealing with **constrained numerical optimization.**

Numerical Optimization

- **Some assumptions**

- We will talk about **unconstrained optimization**
  - **No constraints on the input variables** $x_1, \ldots, x_n$
  - For gradient-based methods
    - The function $f(\mathbf{x})$ is **differentiable** on the whole input domain $\mathbf{D} \subseteq \mathbb{R}^n$
    - **Q:** What does it mean for a function to be differentiable?
    - In some cases (e.g., for the Newton method) the function $f(\mathbf{x})$ will have to be **doubly differentiable** (two times differentiable)

Differentiable functions

Differentiable functions

A function $f(x)$ or (of one variable x) is **differentiable** if its **derivative** $f'(x)$ exists in every point of the domain $D \subseteq \mathbb{R}$ of the input variable (or parameter) x. A function of multiple parameters $f(\mathbf{x} = x_1, x_2, \ldots, x_n)$ is **differentiable** if its **gradient** $\nabla_{\mathbf{x}} f$ – a vector of **partial derivatives** $\nabla_{\mathbf{x}} f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}]$ – exists for every point on the input domain $\mathbf{D} \subseteq \mathbb{R}^n$. If function is differentiable, then it also **continuous**. Most continuous functions used in AI are differentiable.

- Recap: how to compute a derivative of a function ☺

$$f'(x) = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Optimum of a Differentiable Function

- For an infinitesimal change in x, *dx*, the corresponding infinitesimal change in *f*(x), *dy*, is such that the **slope** of the **tangential** in any point **x** corresponds to *dy*/*dx* = *f'*(x)
- In the **turning point** of the function, the function has a (possibly local) **optimum**, and the **tangential is horizontal** (slope is 0)
- So, solving f'(x) = 0 gives us the turning point(s) of *f* and its **optimum**
  - **Q:** How do we tell if its a minimum or maximum?

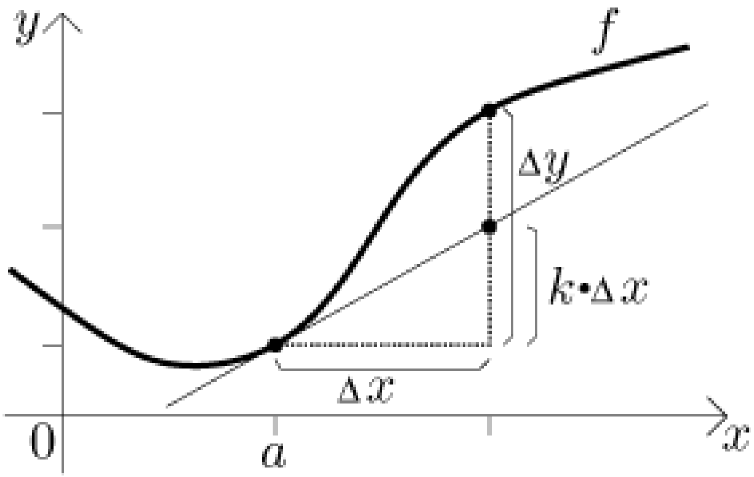


Image from:
https://math.fel.cvut.cz/mt/txtc/1/txe3ca1b.htm

Optimum of a Differentiable Function

- So, solving $f'(x) = 0$ gives us the turning point(s) of f and its **optimum**
- Algebraic conditions for min/max:
  - **MIN:** derivative sign changes from negative to positive
  - **MAX:** derivative sign changes from positive to negative
  - Change of derivative → second derivative $f''(x)$
- So, the function $f(x)$ has a **minimum** in a if $f''(a) > 0$ and a maximum if $f''(a) < 0$

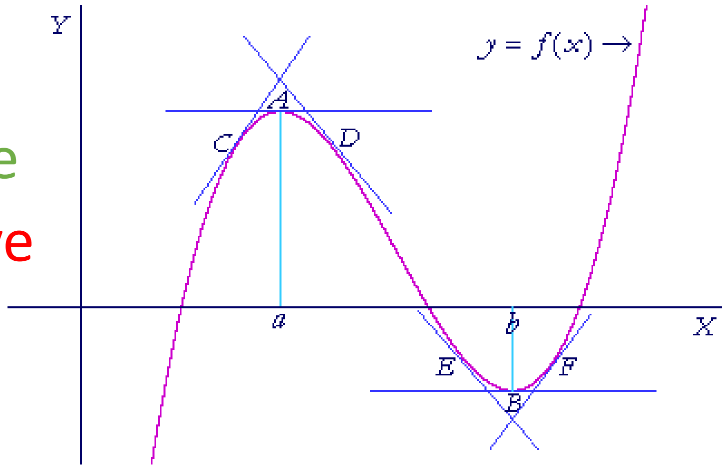


Image from:
https://www.themathpage.com/aCalc/max.htm

Differentiation & Optima: Example

- $f(x) = 2x^3 - 9x^2 + 12x - 3$
- $f'(x) = 6x^2 - 18x + 12$
- $f''(x) = 12x - 18$

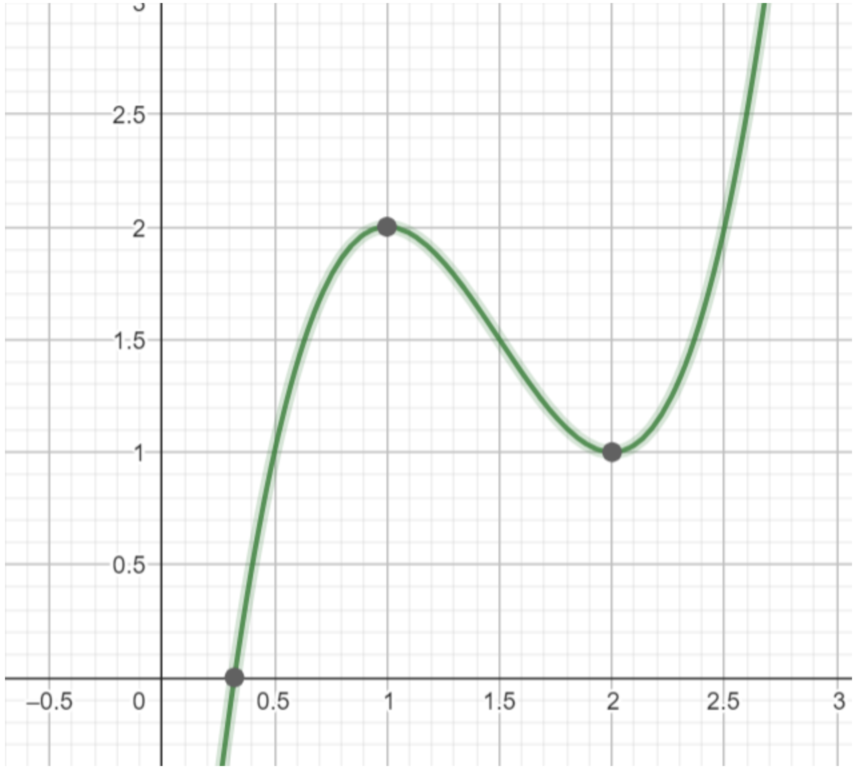
$f'(x) = 0, \quad x^2 - 3x + 2 = 0,$

$(x - 1) * (x - 2) = 0$

$x^{(1)} = 1, \; x^{(2)} = 2$

$f''(x^{(1)}) = 12 - 18 = -6 < 0$, so in $x^{(1)}$, **maximum**

$f''(x^{(2)}) = 24 - 18 = +6 > 0$, so in $x^{(2)}$, **minimum**



Plot generated via https://www.geogebra.org/graphing

Content

- Calculus Basics
- Gradient-Based Optimization
    - Newton Method
    - Gradient Descent
- Search-Based Optimization
    - Genetic Algorithm

Newton's Method

- **Newton's method** is an **iterative method** for finding the **root** of a function $f(x)$, that is, where $f(x) = 0$
  - **Note**: this is different then finding the optimum, where we solve $f'(x) = 0$
- We start from some **initial value** $x^{(0)}$ for which the function value, $f(x^{(0)})$, is „not too far" from 0
- Then we **iteratively update** x as follows:

$$x^{(k+1)} = x^{(k)} - f(x^{(k)}) \,/\, f'(x^{(k)})$$

- **Q:** why does this work? Why do we **converge** to x for which $f(x) = 0$?

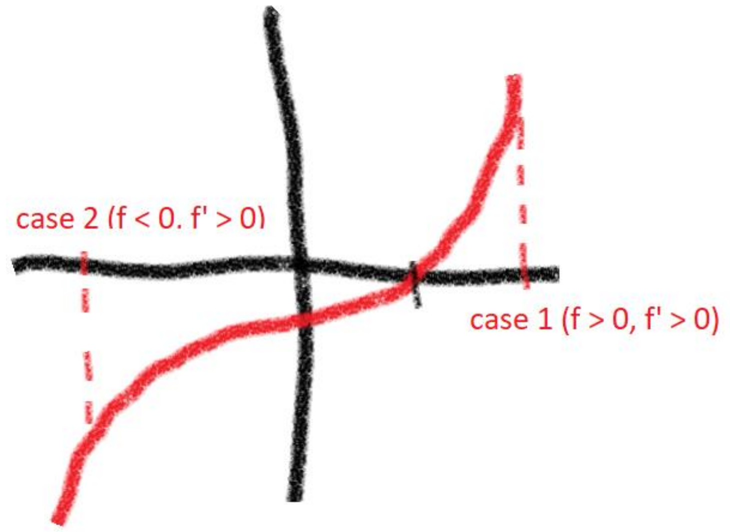
Newton's Method

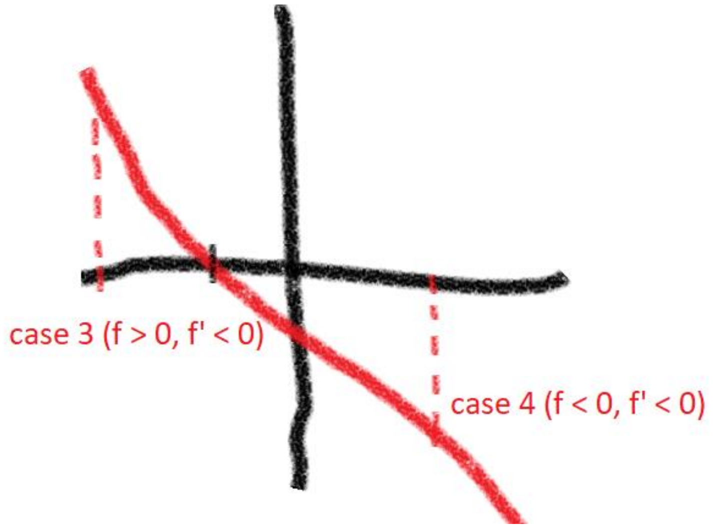
- We **iteratively update** x as follows:

$$x^{(k+1)} = x^{(k)} - f(x^{(k)}) \,/\, f'(x^{(k)})$$

- **Q:** why does this work? Why do we **converge** to x for which $f(x) = 0$?

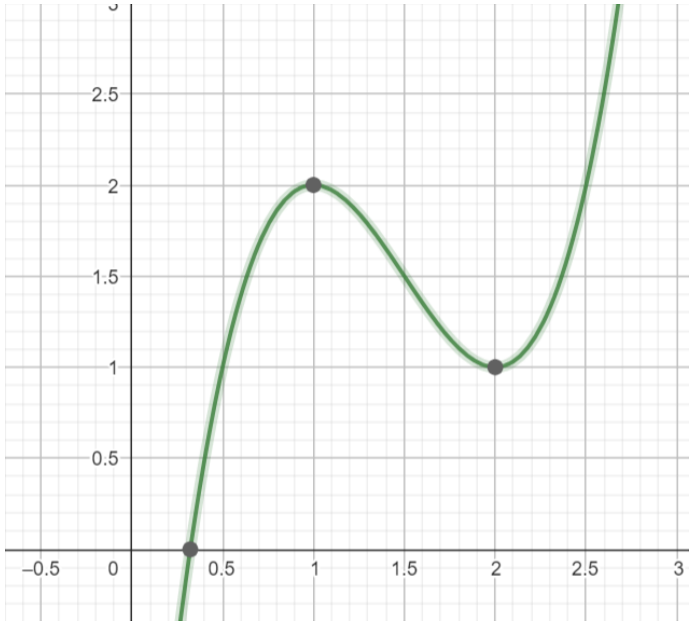
- We have four possibilities:
  1. $f(x) > 0$ and $f'(x) > 0$ → x gets smaller
  2. $f(x) < 0$ and $f'(x) > 0$ → x gets larger
  3. $f(x) > 0$ and $f'(x) < 0$ → x gets larger
  4. $f(x) < 0$ and $f'(x) < 0$ → x gets smaller





Newton's Method: Example

$$x^{(k+1)} = x^{(k)} - f(x^{(k)}) \,/\, f'(x^{(k)})$$



- $f(x) = 2x^3 - 9x^2 + 12x - 3$
- $f'(x) = 6x^2 - 18x + 12$
- For example, $x^{(0)} = -1$
  - $f(x^{(0)}) = -2 -9 -12 -3 = -24$
  - $f'(x^{(0)}) = 6 +18 +12 = +36$
  - $x^{(1)} = -1 - (-24 / +36) = -1+2/3 = -1/3$
  - …
- The **closer** we are to $f(x) = 0$, the **smaller the update** to x – because $f(x)$ is in the nominator of update rule and it's getting smaller (in absolute)
  - The update is 0 (**convergence**) when $f(x) = 0$ ☺

Newton's Method

- Newton's method finds x for which $f(x) = 0$
- But we're looking for an **optimum** of f, not its **root** – we're looking for x such that $f'(x) = 0$
- So we need to apply Newton's method to $f'(x)$ (not f) in order to find the optimum of f

$$x^{(k+1)} = x^{(k)} - f'(x^{(k)}) \, / \, f''(x^{(k)})$$

- But for this **(1)** f has to be doubly differentiable and **(2)** we must know it's first derivative f' (w.r.t. all parameters) in a closed form

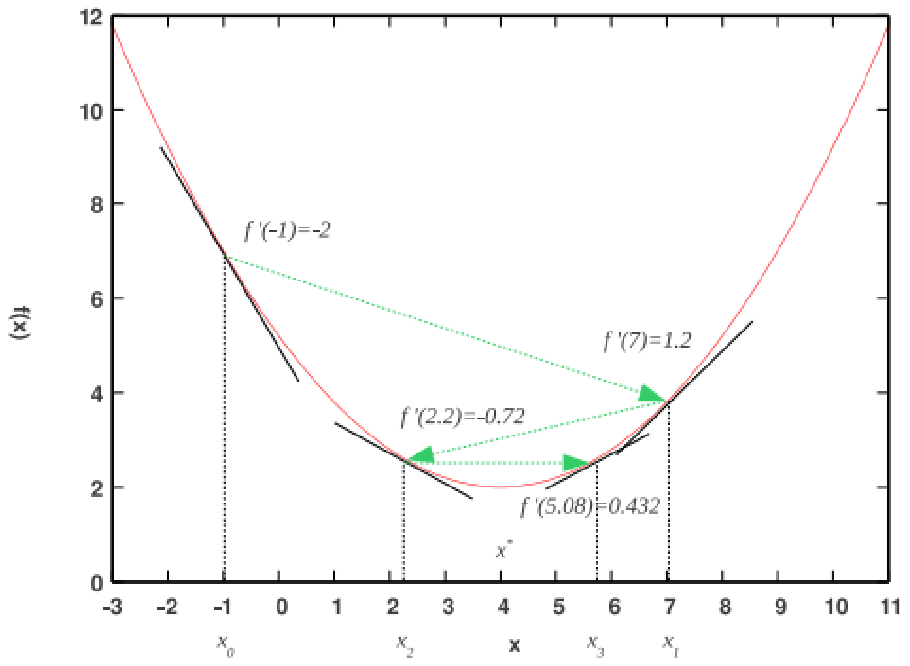
Content

- Calculus Basics
- Gradient-Based Optimization
  - Newton Method
  - Gradient Descent
- Search-Based Optimization
  - Genetic Algorithm

Gradient Descent/Ascent

- **Gradient descent** is a method that moves the parameter values in the direction **opposite** of the function's gradient in the current point
    - This is guaranteed to lead to a **minimum** only for **convex** functions*

- **Gradient ascent** moves the parameter values **in the direction** of the function's gradient in the current point
    - Used to find a **maximum** of a function
    - Guaranteed to find it only for **concave** functions



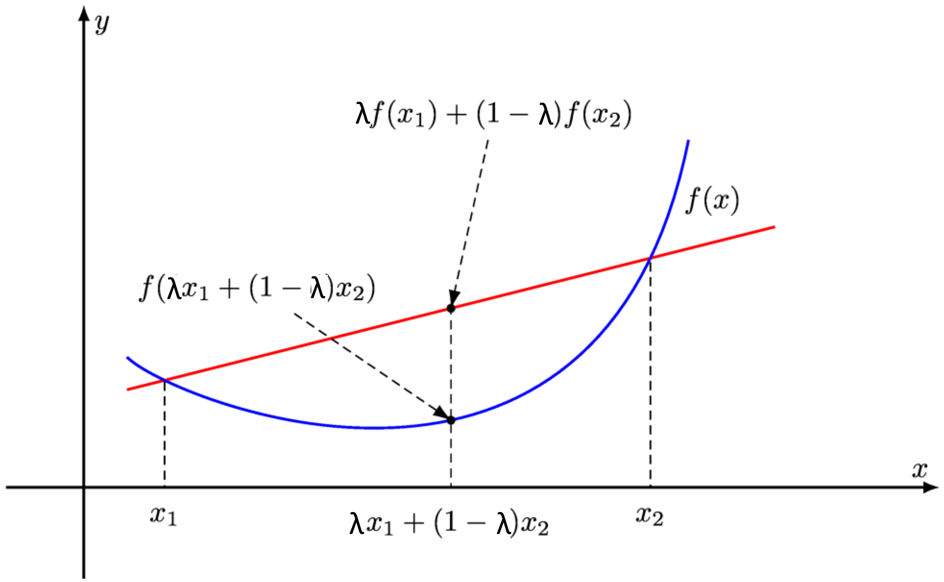
Convex Functions

Convex function

Convex function is a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ whose domain is a **convex set** and for all x_1, x_2 in its domain, and all $\lambda \in [0,1]$, the following inequality holds true:

$$f(\lambda * x_1 + (1-\lambda) * x_2) \leq \lambda * f(x_1) + (1-\lambda) * f(x_2)$$

- **Convex set**, simplified, means a „contiguous" function domain
- **A convex function** has a **unique minimum**



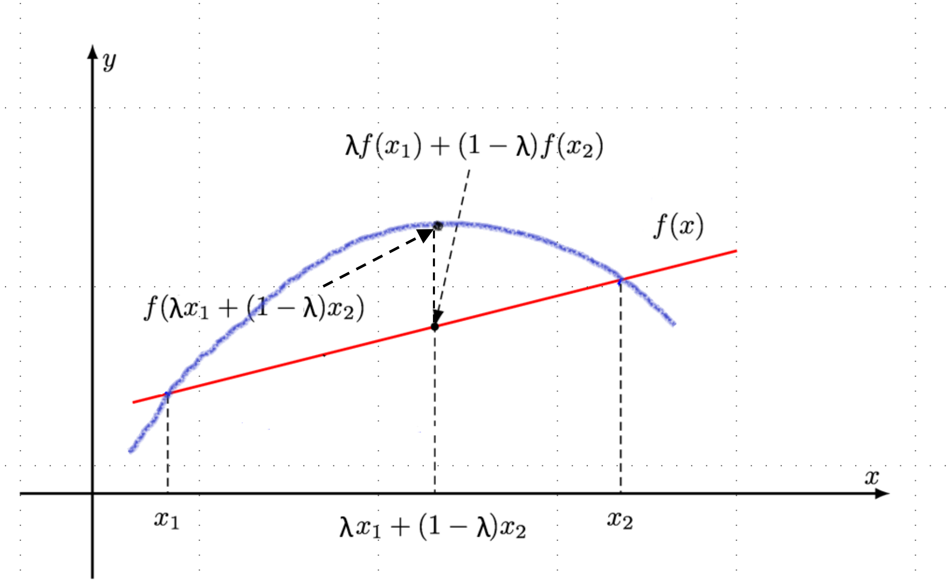
Concave Functions

Convex function

Concave function is a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ whose domain is a **convex set** and for all x_1, x_2 in its domain, and all $\lambda \in [0,1]$, the following inequality holds true:

$$f(\lambda * x_1 + (1-\lambda) * x_2) \geq \lambda * f(x_1) + (1 - \lambda) * f(x_2)$$

- **Convex set** basically means a „contiguous" function domain
- **A concave function** has a **unique maximum**



Gradient Descent

Gradient descent

Gradient descent (sometimes also called **steepest descent**) is an iterative algorithm for (continuous) optimization that finds a **minimum** of a **convex** (single) **differentiable function**.

- In each iteration GD moves the values of variables (vector $\mathbf{x} = [x_1, x_2, ..., x_n]$) **opposite** to the gradient in the current point

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta * \nabla_{\mathbf{x}} f(\mathbf{x}^{(k)})$$

- $\mathbf{x}^{(k)}$ – values of the input variables (arguments, parameters) in step k
- $\nabla_{\mathbf{x}} f(\mathbf{x})$ – value of the gradient (if more than one parameter, then also vector) of the function f in the point $\mathbf{x}$
- η – **step size** (in ML called **learning rate**), defines **how much** to move the parameters in the direction opposite of the gradient

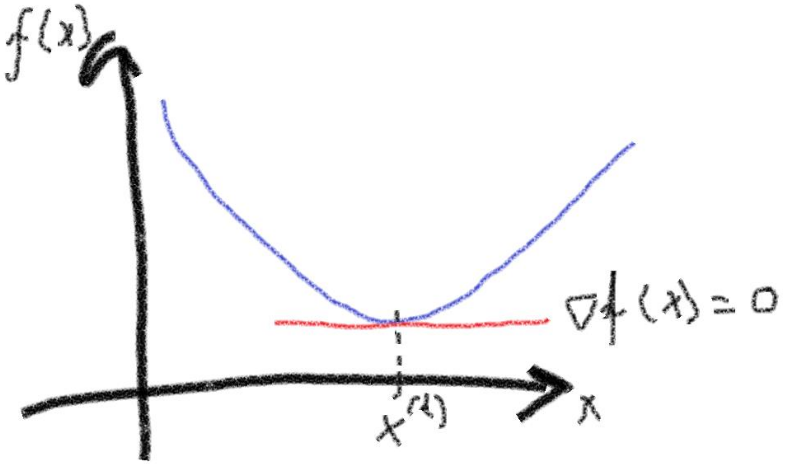
Gradient Descent – Properties

- **Gradient descent:** $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta * \nabla_{\mathbf{x}} f(\mathbf{x}^{(k)})$

- **Q_1**: where to start? Which point to set as initial $\mathbf{x}^{(0)}$?
- **Q_2**: when does this iterative computation stop (does it at stop at all)?
- **Q_3**: assuming it stops, will we have found the minimum of f?
  - What does it depend on?

Gradient Descent – Convergence

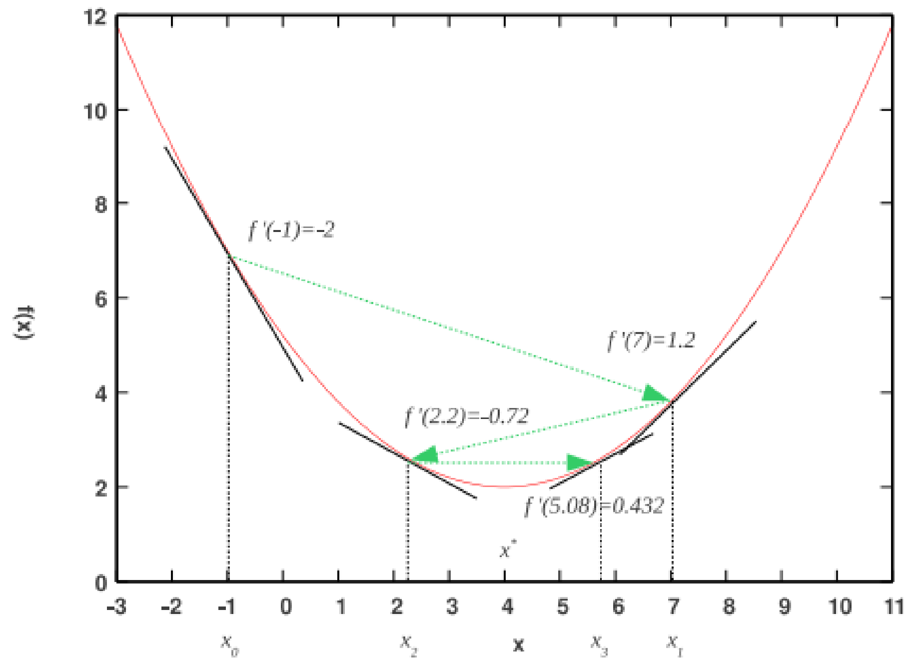
- In principle, unless we know something more about the function f, we would randomly choose an initial point $\mathbf{x}^{(0)}$
- **Convergence**
  - Natural ending of the GD, when the next point, $\mathbf{x}^{(k+1)}$, is the same as the previous, $\mathbf{x}^{(k)}$
  - Given the update formula, this is only possible if the **gradient is zero**: $\nabla f(\mathbf{x}^{(k)}) = 0$
  - This means we have found a minimum – if f is **convex**, gradient is 0 **only** in the minimum



Gradient Descent – Convergence

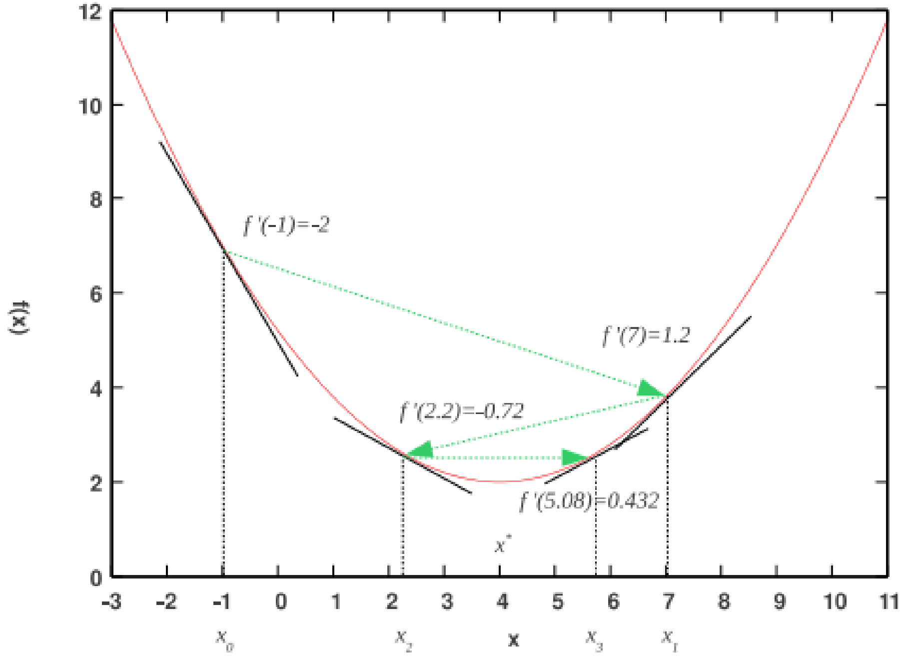
- **Gradient Descent:** $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta * \nabla_{\mathbf{x}} f(\mathbf{x}^{(k)})$

- **Convergence**
  - Whether GD converges depends also on the value of the **step size** η
  - **Q:** What values for η could lead to **divergence** (never converging)?



Gradient Descent – Convergence

- **Gradient Descent:** $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta * \nabla_{\mathbf{x}} f(\mathbf{x}^{(k)})$
- **Convergence**
  - Whether GD converges depends also on the value of the **step size** η
  - If η is **too large**, gradient descent will **diverge**
  - If η is **too small**, gradient descent may not converge in reasonable time (moving too slowly to the minimum)
  - A good step size is usually determined empirically



Gradient Descent – Example

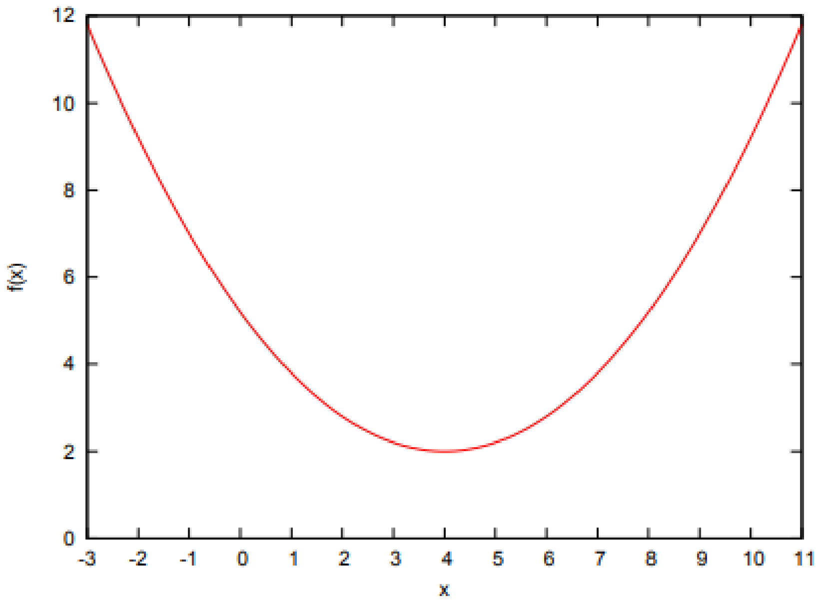
- Let's find the minimum of a single-parameter square function:

$$f(x) = 0.2(x-4)^2 + 2$$

- Of course, in this case, we can easily find the solution **analytically**

$$f'(x) = 0.4 * (x - 4) = 0 \rightarrow x = 4, f(4) = 2$$

- We'd find the same value if we applied GD iteratively, with a suitable step size



Gradient Descent – Example

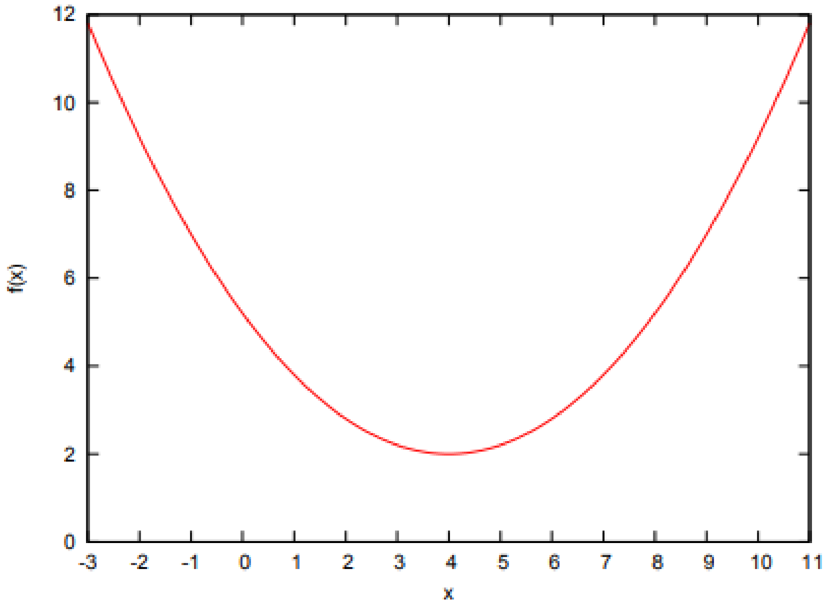
- Let's find the minimum of a single-parameter square function:

$$f(x) = 0.2(x-4)^2 + 2$$

$$f'(x) = 0.4 * (x - 4)$$

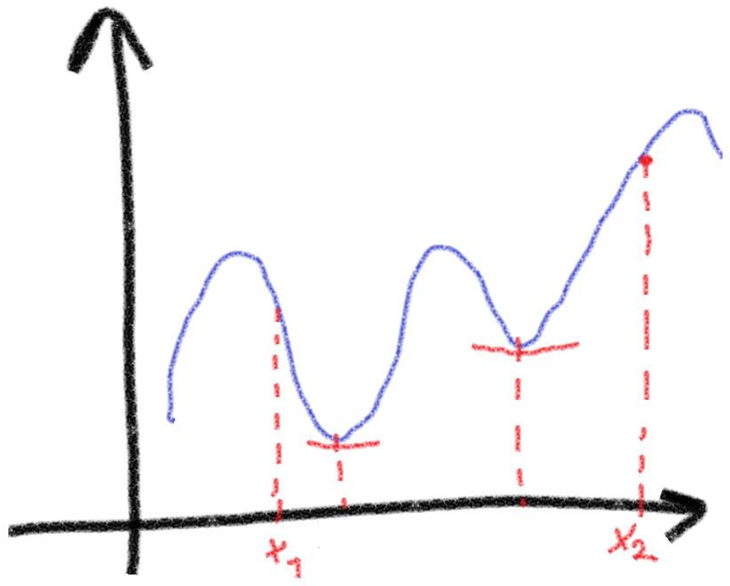
- **GD**: let's start with $x^{(0)} = -1$ and $\eta = 0.5$
  - $x^{(1)} = -1 - 0.5 * 0.4 * (-1 - 4) = 0$
  - $x^{(2)} = 0 - 0.5 * 0.4 * (0 - 4) = 0.8$
  - $x^{(3)} = 0.8 - 0.5 * 0.4 * (0.8 - 4) = 1.44$
  - ...

  - Try with $\eta = 3$. What happens?
  - Try with $\eta = 6$. What happens?
  - Try to start in another point, say $x^{(0)} = 9$



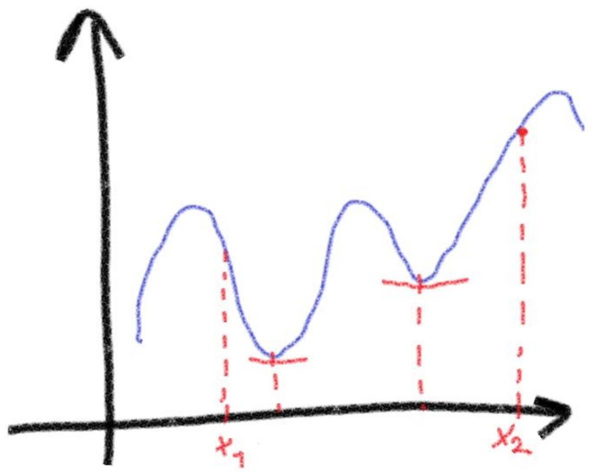
Gradient Descent – Non-Convex Optimization

- If the function is non-convex, gradient descent will not necessarily find a **global minimum**
- There are other, **local minimums** that it can end up in
- Gradient („steepest") descent is **guaranteed** to end up in the closest local minimum
  - Closest to the starting point
  - Assuming a small enough step size
- Where we end up depends on the **start**



Gradient Descent – Non-Convex Optimization

- Most complex functions that we optimize in practice are non-convex
- GD may not find the global minimum, but maybe the **local minimum** it finds is good enough
- **Improvement strategies**
  1. **Multiple GD runs** (from different initial points)
     - Take the smallest of the local optima
     - Computationally expensive (multiple optimizations)
  2. **Dynamic (adaptable) step size**
     - Not the same step size throughout the optimization
     - Not necessarily the same step size for all parameters
     - Several different adaptable GD variants
       - AdaGrad, RMSProp, **Adam**



Gradient Ascent

Gradient descent

Gradient ascent (sometimes also called **steepest ascent**) is an iterative algorithm for (continuous) optimization that finds a **maximum** of a (single) **differentiable concave function**.

- In each iteration GD moves the values of input variables (vector **x** = $[x_1, x_2, ..., x_n]$) **in the direction** of the gradient in the current point

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \eta * \nabla_{\mathbf{x}} f(\mathbf{x}^{(k)})$$

- In practice, **gradient ascent** is rarely used (especially in AI)
  - In **machine learning** we commonly compute **error/loss functions** (distance between predictions and correct labels) which we **minimize** (so GD, not GA)
  - Maximizing a function f is equivalent to minimizing $-f$

Content

- Calculus Basics
- Gradient-Based Optimization
  - Newton Method
  - Gradient Descent
- Search-Based Optimization
  - Genetic Algorithm for Numeric Optimization

Search-Based Optimization

- **Gradient-based optimization** applicable only for differentiable functions
  - **Q:** What to do for non-differentiable or non-smooth functions (noisy gradients)?
  - **Q:** What to do for numeric optimization with constraints?
  - Depending on the nature of the function and constraints, there may be **dedicated optimization algorithms**

- **Search-based methods** for numerical optimization
  - Useful if we don't have a **good initial guess** for **good parameter values**
  - **Good** if function *f* to be optimized is **not differentiable** or **not smooth** or if the function domain is discontinuous
  - **Easier to incorporate constraints** than in gradient-based methods
  - For **optimization of unconstrained differentiable functions** – slower and find worse solutions than gradient-based optimization

Metaheuristics for Numerical Optimization

- **Search-based methods** for numerical optimization, some examples:
  - Optimized Step Size Random Search (OSSRS)
  - Symmetric Perturbation Stochastic Approximation (SPSA)
  - Nelder-Mead Algorithm
  - Nature inspired metaheuristics: **Genetic Algorithm**

- **Genetic algorithms**, which we've seen in **discrete optimization** can also be leveraged for **numerical (i.e., continuous) optimization**
  - **Q:** How to represent the chromosome?
  - **Q:** what selection, crossover, and mutation strategies/operators to use?

Genetic Algorithm for Numerical Optimization

- Simplest case: one-parameter function, e.g., $f(x) = 7x^3 + 3x^2 - 15x + 21$
  - **Chromosome** must be some kind of encoding of the value of x
  - If we have **multiple parameters**, **chromosome =** concatenation of encodings

- **Binary encoding (binary chromosome)**
  - Vector of length N with binary values
  - E.g., N = 10, **[0, 1, 0, 1, 0, 0, 0, 1, 1, 0]**
  - **Q:** If we know that the domain of valid values for x is **[a, b]** what is the smallest increment (change in value) of x that we can encode?

  - If our vectors are of length N, then we can have **at most** 2^N **different vectors**
  - 2^N different values for the variable x, on its domain range **[a, b]**
    - So, the smallest „increment” in value change of x is $(b - a) / 2^N$

Genetic Algorithm for Numerical Optimization

- **Binary encoding (binary chromosome): example**
  - Single parameter (single value that we're encoding)
  - E.g., N = 10,
  - Range of values (domain) for x: [-10, 10] (a = -10, b = 10)
  - So, the smallest „increment" in value change of x is **(b – a) / 2^N**
  - **Increment** (precision): (10 – (-10)) / 2^{10} = 20 / 1024 = **0.0195**

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0] → -10
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1] → -10 + **0.0195 = -9.9805**
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0] → -9.9805 + **0.0195 = -9.961**
…
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0] → 10

Genetic Algorithm for Numerical Optimization

- The **genetic algorithm** itself is exactly the same as in discrete optimization
- **Fitness** of the chromosome is the actual value *f*(x) for the value x that the chromosome encodes
- **Selection**
  - tournament or **rhoulette wheel**
- **Mutation**
  - Bit flipping (0 to 1 and vice versa)

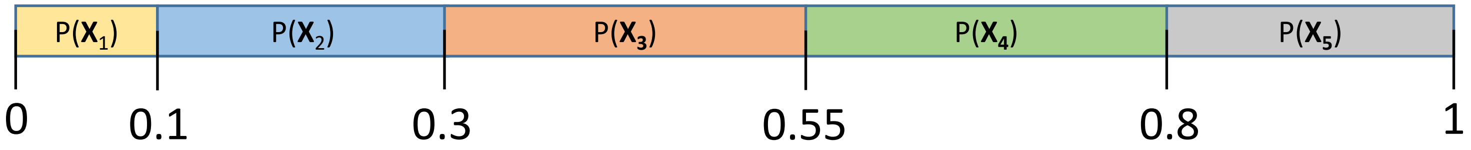
```
genetic_algorithm(S,end)
  p = create_init_population(S)
  iter = 0
  evaluate(p)
  while not end(p, iter)
    iter = iter + 1
    p' = recombine(p)
    mutate(p')
    evaluate(p')
    p = select(p ∪ p')
  return p
```

Genetic Algorithm: **Selection**

- **Roulette wheel** (or **proportional**) selection: probability of being selected for reproduction proportional to the fitness of the chromosome

$$P(\mathbf{X}_i) = f(\mathbf{X}_i) \,/\, \sum_j^S f(\mathbf{X}_j)$$

- Let us have a population of 5 chromosomes and let
  - $fit(\mathbf{X}_1) = 10$, $fit(\mathbf{X}_2) = 20$, $fit(\mathbf{X}_3) = 25$, $fit(\mathbf{X}_4) = 25$, $fit(\mathbf{X}_5) = 20$ → convert into probabilities



- But if we're doing numerical **minimization** then **smaller values of *f* are better**
  - The fitness of the chromosome can then be $fit(x) = f_{MAX} - f(x)$
  - f_{MAX} is the maximal value of the function we're minimizing (on the domain of x)
    - If we don't know the actual max, it can be the smallest large value, such that $f_{MAX} - f(x)$ is not negative for any x

Genetic Algorithm: **Recombination**

- **Common crossover operators**
  - **Single-point crossover**: select (typically randomly) the location at which to cut the chromosomes and „exchange them" → two „child" chromosomes
  - Unless we're doing constrained optimization, resulting chromosomes are **valid**

[0, 1, 0, 1, 0, 0, 0, 1, 1, 1] → [0, 1, 0, 1, 1, 0, 1, 0, 0, 1]
[1, 0, 1, 1, 1, 0, 1, 0, 0, 1] → [1, 0, 1, 1, 0, 0, 0, 1, 1, 1]

- **Mutation**: **flip the bit** (0 → 1 or 1 → 0) randomly (with some small mutation probability)

Questions?


Awọn ibeere?
ਸਵਾਲ?
Küsimusi?
Turite klausimų?
Pitanja?
Sorusu olan?
Dúvidas?
質問は？
有问题吗？
Fragen?
¿Preguntas?
Questions?
Frågor?
Pytania?
Domande?
Питання?
Vragen?
Ερωτήσεις;
Porandukuéra?
أسئلة؟