ARTIFICIAL INTELLIGENCE

LEI/3, LMA/3, MBE/1

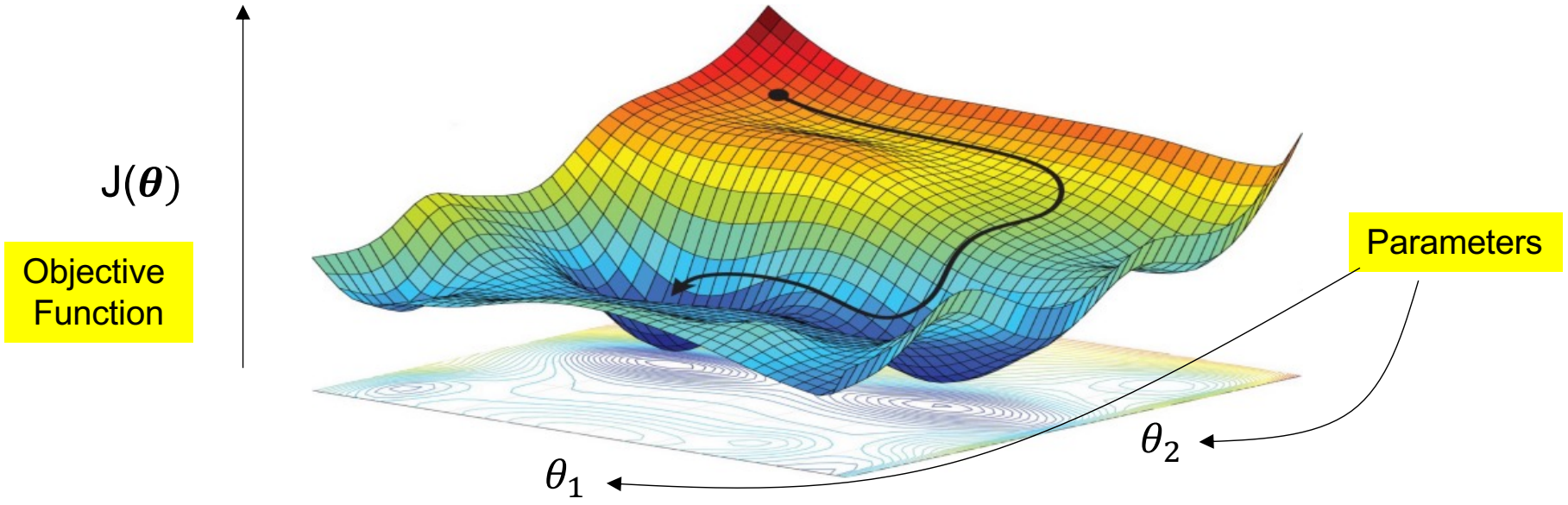
University of Beira Interior, Department of Informatics

Hugo Pedro Proença

hugomcp@di.ubi.pt, **2022/23**

Local Search/Optimization

- Up to now, we've seen that many different problems can be formulated as Artificial Intelligence search problems
- We have typically… 1) a huge number of states (our state space); 2) a starting state; and 3) a goal state.
- However, for many problems (i.e., mostly of the *real-world* problems), searching all possible solutions is not feasible, either because there is a potentially infinite number of states, or because that number is simply too high.
  - In local search/optimization problems, we are interested in finding a parameterization for our model ($\boldsymbol{\theta} = \theta_1,\ldots\theta_n$) that provides a good solution



Local Search/Optimization

- At the bottom line, Local Search involves to search across a sub-space of the parameterization space (at a given granularity), and find the configurations of $\boldsymbol{\theta}$ that maximize/minimize the **objective function**.
  - Additionally, there are also some constraints that should be satisfied.
- Examples:
  - Circuits Design
    - Given: a board, a set of components and connections
    - Goal: place each component in the board, so as to maximize energy efficiency, minimize production costs,…
  - Logistics
    - Given: a set of places to be visited/supplied
    - Goal: Generate the shortest route, to maximize efficiency in terms of fuel consumption
- Moreover, optimization is used in a myriad of other areas including medicine, manufacturing, transportation, supply chain, finance, government, physics, economics,
- The goals range from minimizing the cost in a production system, or - in a hospital - to minimize the wait time for patients in an emergency room before they are seen by a doctor. Also, in Marketing, the goals can be to maximize the profit obtained by targeting the right customers under budget and operational conditions.
- Often (or always), it is very hard (NP-complete) to find the optimal solution.

Local Search/Optimization

- Broadly, there are two families for local search methods:
  - Constructive Methods. They start from scratch (∅), and iteratively build a solution.
  - Repair/ Methods. These methods start from a randomly chosen (or expert-based) solution and iteratively improve it.
- Importantly, Local Search algorithms operate using a single current state (rather than multiple paths as the previously family of algorithms studied, e.g., A∗) and move only to neighbors of that state.
- At each step we have a complete but imperfect solution to a search problem.
- There are good properties that yield from this
  - The is a constant dimension in terms of the state space, i.e., it uses very little memory.
  - Can find reasonable solutions in very large state spaces, where exhaustive search would fail miserably.
  - All states have an objective function
  - The goal is to find state with max (or min) objective value

Hill-Climbing

- This is the simplest form of Local Search
- The idea is to design "*a loop that continuously moves towards increasing value*"
- It terminates when a peak is reached, i.e., when none of the successors of a state provides a better value for the objective function.
- This paradigm is also known as Greedy Local Search
- It does not look ahead of the immediate neighbors
- If more than one successors have equal objective value (better than the current state), it randomly choose among the set of best successors
- It is regarded as "*climbing Mount Everest in a thick fog with amnesia*"

It is prune to local maxima, i.e., if there is one iteration where none of the successors provide a better objective value than the current state, it stops

Tabu Search

- As we've seen previously, Hill climbing is too sensitive to local maxima. One solution is to take steps back from that optimum point and go down to reach the bottom.
- Once the bottom is reached, the search is resumed, hoping that a better solution will be reached.
- This is regarded as a **sideway move.**
- However, we should limit the number of possible sideways moves, in order to prevent infinite looping. This is exactly the idea of **Tabu Search.**
- We keep a fixed length queue, a.k.a. the Tabu list.
- We add the current state to the queue, and drop one element (i.e., the oldest).
- We never allow movements to a currently tabu'ed state.
- If the size of the tabu'ed set increases, tabu search asymptotically becomes non-redundant. That is, it would not visit the same state twice.
- In practice, tabu list queue size of 100 or such improves the performance of tabu search over hill climbing in many problems. If the tabu list size is extremely large or ∞ then tabu search essentially becomes a systematic search.

Simulated Annealing

- In practice, Hill-climbing algorithm **is incomplete** (*)
  - In the sense that it does not guarantee the convergence to a solution
  - Then, random walk variants (e.g., Tabu Search) are complete, but extremely inefficient.
- The idea to combine both families of algorithms yielded the Simulated Annealing algorithm, that considers a tradeoff between **exploration** of the search space and **exploitation** of an imperfect solution.
- Physical analogy:
  - *"Annealing of metals is the process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, allowing the material to coalesce into a low-energy crystalline state."*

Simulated Annealing

- At every iteration, a random move is chosen.
  - If it improves the situation then the move is accepted, otherwise it is accepted with some probability less than 1.
- The probability decreases exponentially with the badness of the move. It also decreases with respect to a temperature parameter T.
- Simulated annealing starts with a high value of T and then T is gradually reduced. At high values of T , simulated annealing is like pure random search. Towards the end of the algorithm when the values of T are quite small, simulated annealing resembles ordinary hill-climbing.
- Simulated annealing **finds a global optimum with probability approaching 1** if we lower T slowly enough.
- The exact bound for parameter t and schedule for T is usually problem dependent. Thus we need to experiment heavily with every new problem at hand to see whether simulated annealing makes a difference.
- Simulated annealing is a very popular algorithm and has been used to solve various classes of optimization problems

Simulated Annealing

- The rationale is to allow some apparently "bad transitions", in the hope of escaping from local maxima.
- However, we should assure that the frequency of such bad moves decreases over time, i.e., when we should be approaching the global optimum.
- Essentially, when the energy of a successor is higher than the current node, we simply move to that state
- Otherwise, we move to the new state with probability modelled by the Boltzmann distribution:

$$\exp\left(\frac{E_{new} - E}{T}\right)$$

- T > 0 is the temperature, that starts high and goes (over time) toward 0.
- When T is high, the exponent is close to 0, and thus the probability of accepting any move is close to 1
- When T approaches 0, the probability of moving to a worse solution is almost 0.
- We decrease T by multiplying it with a constant $\alpha < 1$
- When T is high, we are moving in the **exploratory phase**
- When T is low, we approach the **exploitation phase**
  - The temperature annealing schedule is crucial (so it needs to be tweaked)
    - Cool too fast and we do not reach optimality
    - Slow cooling leads to very slow improvements

Optimization

- Among the three major concepts in Artificial Intelligence, we have:
  - **Linear Algebra**
  - **Statistics**
  - **Optimization**
- At the bottom line, optimization refers to "*maximizing or minimizing a real function f(**x**) by systematically choosing input values from an allowed set and computing the value of f(**x**).*"
- In this setting, **x** refers typically to the parameters of our model (a.k.a. $\boldsymbol{\theta}$)
- Also, it is common to have a set of constraints that can be either hard constraints, which set conditions for the variables that are required to be satisfied, or soft constraints, which have some variable values that are penalized in the objective function if, and based on the extent that, the conditions on the variables are not satisfied.
- A general optimization problem can be formulated as:

$$\min f(\boldsymbol{x})$$

subject to

$$g_i(\boldsymbol{x}) = c_i \forall\, i \in 1,..,n$$
$$h_i(\boldsymbol{x}) \geq c_i \forall\, i \in 1,..,n$$

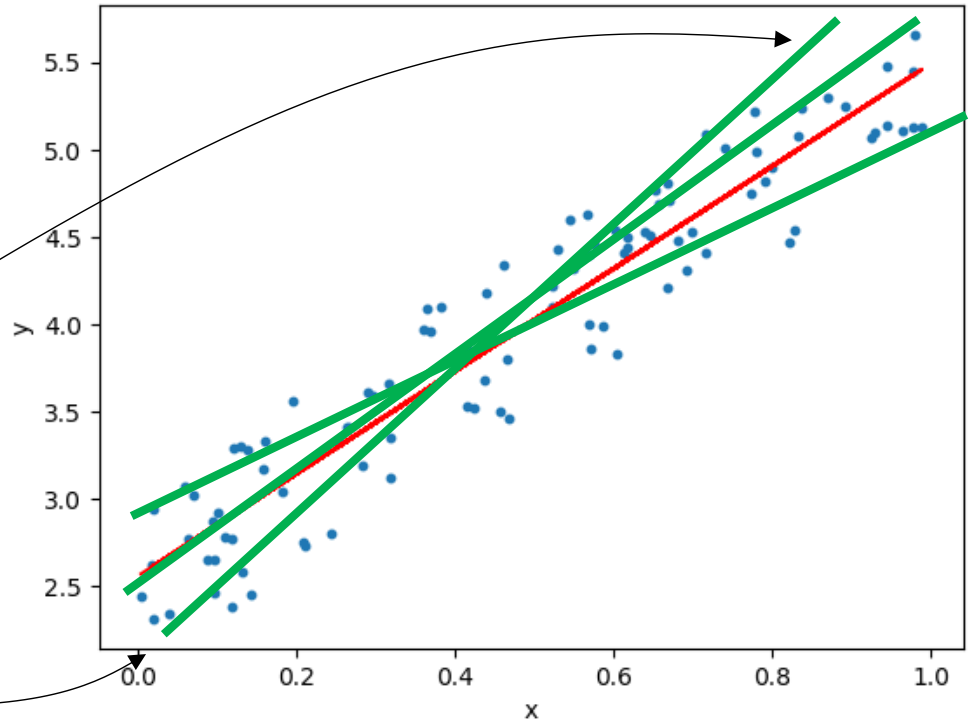
Optimization - Example

- Suppose we have a set of observations of an input variable (x_i) and the corresponding outputs (y_i), for a particular problem.
- As it seems obvious that x, y vary directly in a roughly linear way, we are interested in obtaining the model that optimally expresses the relationship between x and y.
- Clearly, there are many (infinite) potential solutions to the problem

Finding the best model (i.e., a straight line $y = mx + b$) is an optimization problem, in this case without constraints

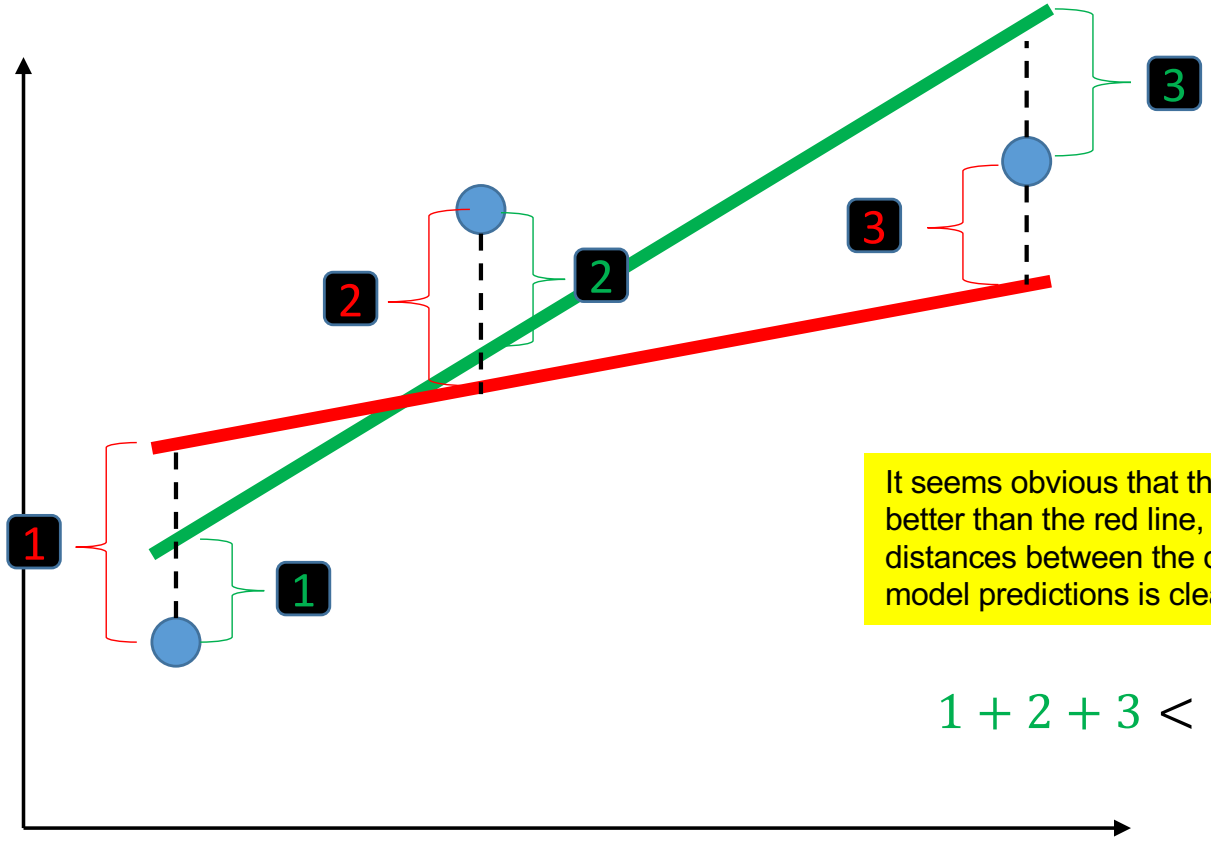
Parameters: (m, b) = $\boldsymbol{\theta} = (\theta_1, \theta_2)$

Each straight line has a different configuration $\boldsymbol{\theta}$, but which one is the best of all?



Optimization

- A key part of any optimization problem is to perceive how can we distinguish between two potential solutions, i.e., hoe can we say that "*Solution A is better than solution B*"?



It seems obvious that the green line is better than the red line, as the sum of distances between the data points and the model predictions is clearly smaller.

$$1 + 2 + 3 < 1 + 2 + 3$$

Optimization

- Hence, we start by defining **an objective function** $J(\boldsymbol{\theta})$ that accumulates the distances between the actual output (y_i) and the prediction $f_\theta()$ given by the model for a particular input (x_i).

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(yi - f_{\boldsymbol{\theta}}(xi))^2$$

Constant term, that can be disregarded from the process

Euclidean distance

- The optimal model will be given by:

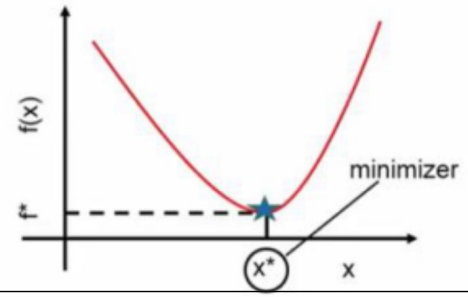
$$\widehat{\boldsymbol{\theta}} = arg\ min_{\boldsymbol{\theta}}\, J(\boldsymbol{\theta})$$

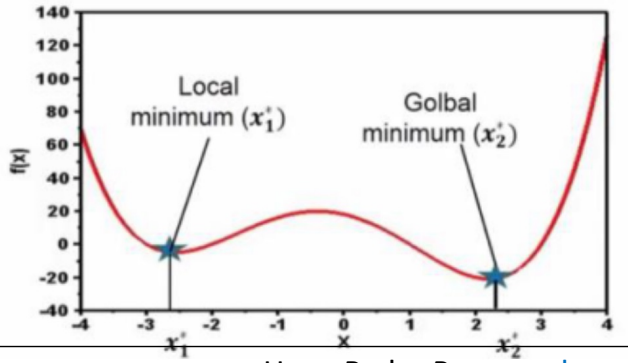
The optimal parameterization will be the argument that minimizes J($\boldsymbol{\theta}$), when searching over all $\boldsymbol{\theta}$ configurations

Optimization

- Depending on the type of objective function, constraints and decision variables, the optimization process can be solved in different ways:
  - Linear programming: If the functional form is linear and all constraints are also linear.
  - Non linear programming: If the decision variables are continuous and either the objective function or constraints are non linear,
- The variables can be integer or real, and in the former case, the term "integer" is commonly added to refer that problem.
  - For example, If the objective function and constraints are linear and the decision variable is an integer, it is called a Linear integer programming problem.
- An important property of this class of problems is their convexity, i.e., problems where the local and global minima might not coincide.
  - Such functions are known as Non-convex functions. They might have multiple local minima

Convex





Non-Convex

Optimization: Closed-Form

- Minimizing $J(\boldsymbol{\theta})$ can be done in one of two ways: 1) using the closed-form solution; and 2) iteratively, according to gradient descent.
  - An optimization problem is closed-form solvable if it is differentiable with respect to the weights $\boldsymbol{\theta}$ and the derivative can be solved.
  - The closed-form solution is obtained at-once (i.e., non-iteratively) and is exact.
  - However, using the closed-form might be harder, if the model has a complicated expression (i.e., far from linear, as in a multi-layer neural network) or the amount of data is too large (a matrix should be inverted in the process).
- Typically, the problem can be formulated as a set of inputs $\boldsymbol{x_i}$ that should be mapped to the corresponding y_i elements ($\boldsymbol{x_i} \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$).
- Having a matrix $\boldsymbol{Y} = [y_1, \ldots, y_n]$, representing the observed outputs.
- We create a matrix: $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\mathsf{T}$ with $\boldsymbol{x}_i$ representing each observation concatenated to a "1" in the final position, i.e., $\boldsymbol{x}_i = [\boldsymbol{x}_i, 1]$.
- We create a matrix of weights $\boldsymbol{\theta} = [\theta_1, \ldots \theta_d, \theta_b]$
- The mapping can be formulated as

bias

$$\boldsymbol{Y} = \boldsymbol{X}\,\boldsymbol{\theta}, \text{ with } \boldsymbol{Y} \in \mathbb{R}^n, \boldsymbol{X} \in \mathbb{R}^{n\times(d+1)}, \boldsymbol{\theta} \in \mathbb{R}^{(d+1)}$$

Optimization: Closed-Form

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2} (\boldsymbol{X\theta} - \boldsymbol{Y})^T (\boldsymbol{X\theta} - \boldsymbol{Y})$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} = \frac{1}{2} [(\boldsymbol{\theta}^T \boldsymbol{X}^T \boldsymbol{X\theta}) - \boldsymbol{\theta}^T \boldsymbol{X}^T \boldsymbol{Y} - \boldsymbol{Y}^T \boldsymbol{X\theta} + \boldsymbol{Y}^T \boldsymbol{Y}]$$

$$= (\boldsymbol{X}^T \boldsymbol{X\theta}) - \boldsymbol{X}^T \boldsymbol{Y}$$

- To find the minimum, we obtains the zeroes of the derivative, by solving for $\boldsymbol{\theta}$:

$$\frac{\partial}{\partial \boldsymbol{\theta}} = (\boldsymbol{X}^T \boldsymbol{X\theta}) - \boldsymbol{X}^T \boldsymbol{Y}) = 0$$

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{Y}$$

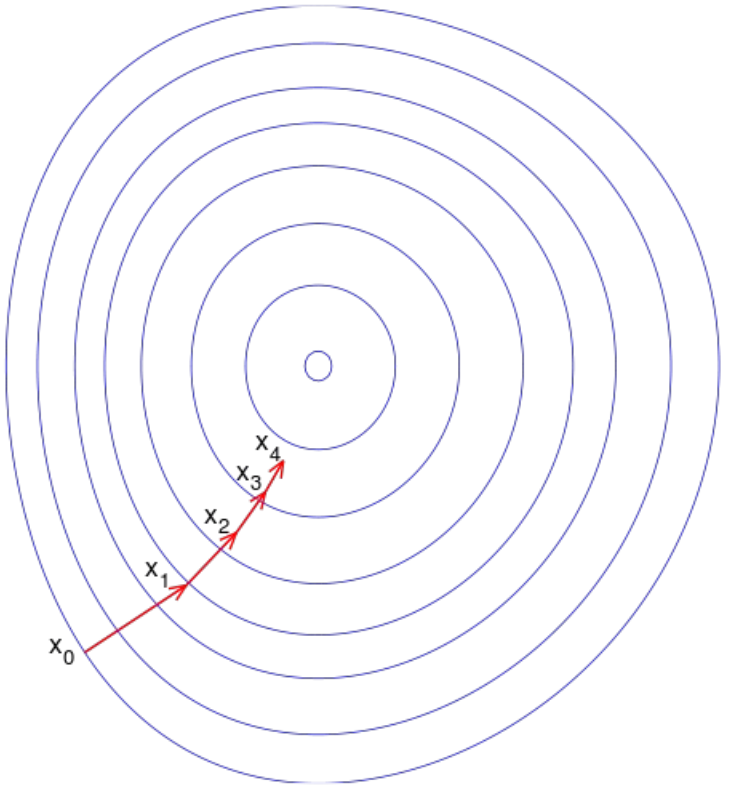
This is the **tricky operation**. Not only the matrix might not have an inverse, but it might be too expensive to obtain it (e.g., if there are too many features)

Optimization: Gradient Descent

- Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function.

- To find a minimum of a function using gradient descent, **one takes multiple steps** proportional to the negative of the gradient.
  - This way, it is an iterative algorithm, which **converges to the local minimum if the learning rate is low enough**.

- It is based on the observation that if a multi-variable function $f(\boldsymbol{x})$ is **differentiable in a neighborhood of a point $\mathbf{x}_i$**, then $f()$ decreases fastest if one goes from $\mathbf{x}_i$ in the direction of the negative gradient of f() at $\boldsymbol{x}$: $-\nabla f(\boldsymbol{x}_i)$



Optimization: Gradient Descent

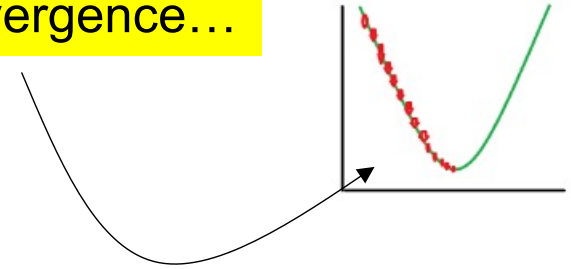
- In order to obtain the point **x that minimizes f(x), we update it according to the following rule**:

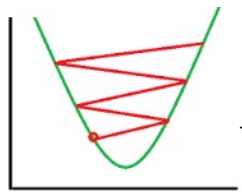
$$\mathbf{x}_{t+1} = \mathbf{x}_t - \gamma \nabla f(\mathbf{x}_t)$$

Learning rate

- In practice, one starts with an initial guess $\mathbf{x}_0$ typically random and update iteratively $\mathbf{x}_{t+1}$ such that the sequence $\{\mathbf{x}_i\}$ converges to a minimum.
- The learning rate γ plays a major role in the results of the optimization algorithm.
  - Too small values would take too long time to achieve a minimum;
  - Too large values might be even worse: might lead to diverging sequences.

Convergence…





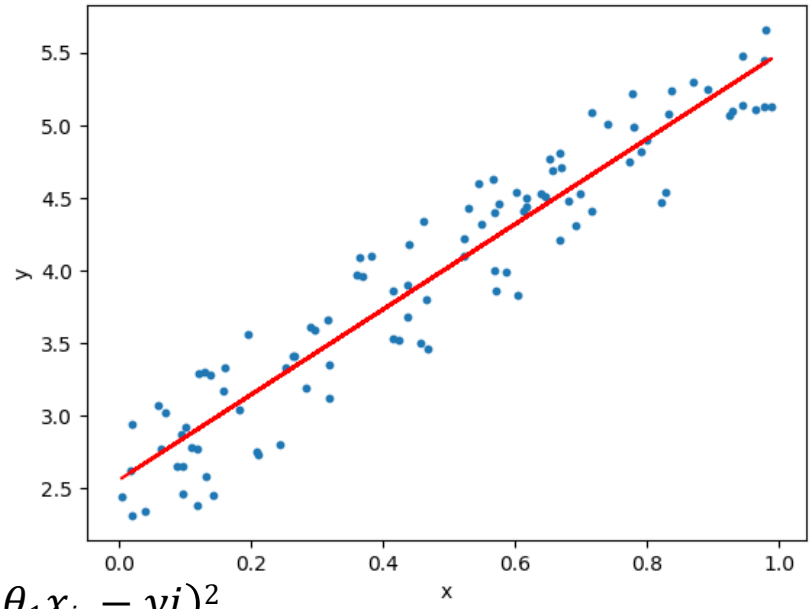
Divergence!!

Optimization: Gradient Descent Example

- Revisiting our previous example, suppose that we want to find the model (straight line) that better expresses the relationship between the inputs x_i and outputs y_i.

$$y_i = \theta_0 + \theta_1 xi$$

y = mx + b



$$J(\theta_0, \theta_1) = \frac{1}{2}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - yi)^2$$

- Hence, the problem can be regarded as finding the $\boldsymbol{\theta}$ parameters (unknowns) that minimize *J()*

Optimization: Gradient Descent

- **Step 1**. Find the partial derivatives of J() with respect to each θ_i :

$$\frac{\delta J}{\delta \theta_0} = \sum_{i=1}^{n} \theta_0 + \theta_1 x_i - yi$$

$$\frac{\delta J}{\delta \theta_1} = \sum_{i=1}^{n} (\theta_0 + \theta_1 x_i - yi) x_i$$

- **Step 2.** Draw the initial values of θ_i (0) : (e.g., $\boldsymbol{\theta}$=[1, 0])
- **Step 3.** Define the learning rate (e.g., $\gamma = 1$)
- **Step 4.** Repeat "m" times
  - **Step 4.1.** Find the new values of θ_i
    - Important: Update all θ_i simultaneously, i.e., do not use θ_0 (t+1) to find θ_1 (t+1)
  - **Step 4.2.** Update all θ_i values
    - $\theta_i(t+1) = \theta_i(t) - \gamma \frac{\delta J}{\delta \theta_i}$

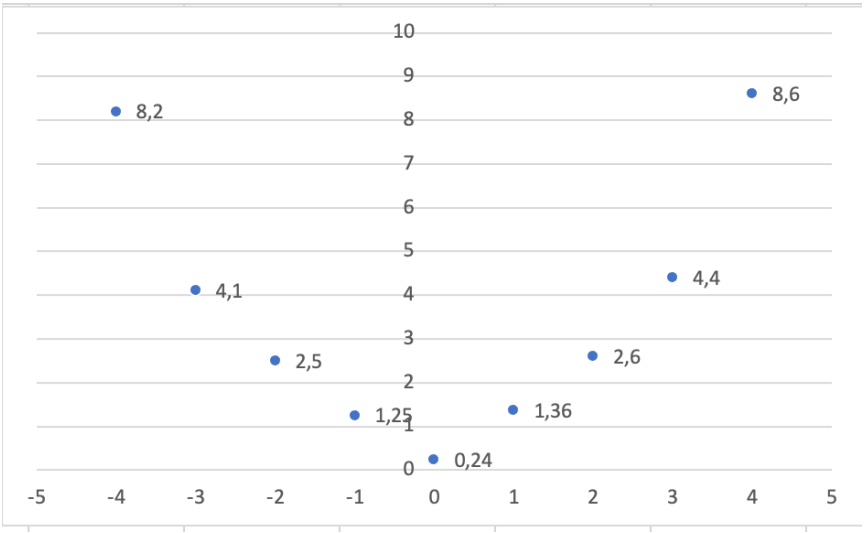
Optimization: Gradient Descent Exercise

- Consider the following data set. Consider $\gamma = 1$ and $\boldsymbol{\theta}(0)=[0, 0, 0]$.
- Use the gradient descent algorithm to find the quadratic model that optimally fits the dataset:

$$J(\theta_0, \theta_1, \theta_2) = \frac{1}{2}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i + \theta_2 x_i^2 - yi)^2$$

- Implement a Python script that obtains the first "n" iterations of $\boldsymbol{\theta}$ values.

X	Y
-4	8,2
-3	4,1
-2	2,5
-1	1,25
0	0,24
1	1,36
2	2,6
3	4,4
4	8,6

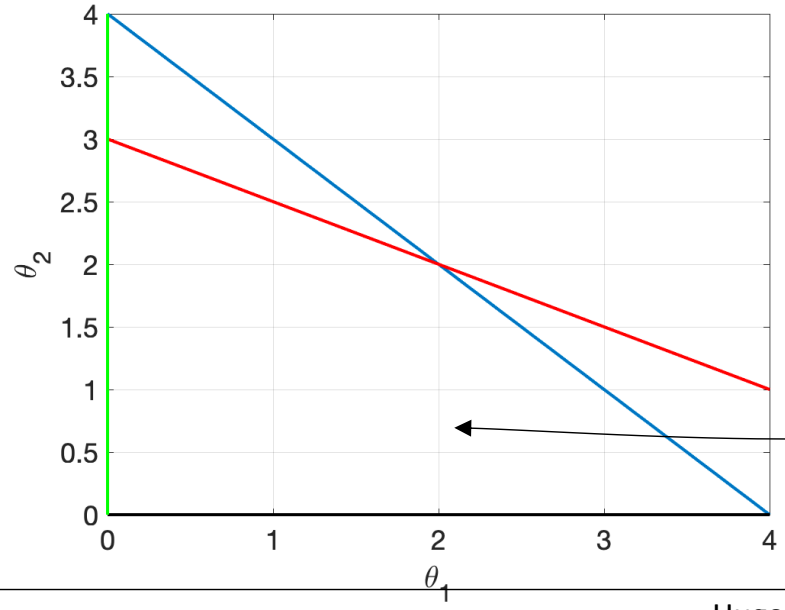


Optimization: Linear Programming

- **Linear Programming** (a.k.a. Linear Optimization) is used to solve mathematical problems in which the relationships are linear in nature.
- The idea in Linear Programming is to maximize or minimize an objective function, subject to some constraints.
- The objective function is a linear function which is obtained from the mathematical model of the problem. The constraints are the conditions which are imposed on the model and are also linear.
- There are mainly two ways of solving linear programming problems:
  - **Graphical Method**, or **Simplex Method**.
- Graphical Method
  - Having an objective function $J(\boldsymbol{\theta})$, and a set of constraints, we start by drawing the constraints on a graph, **to find the feasible region**.
  - The feasible region is the intersection of all the constraints.
  - Next, we find the vertices of the feasible region and find the value of $J(\boldsymbol{\theta})$ at these vertices.
  - The vertex that maximizes/minimizes $J(\boldsymbol{\theta})$ is the final answer.

Linear Programming: Graphical Method Example

- Suppose that we want to maximize: $J(\boldsymbol{\theta}) = \boldsymbol{4\theta}_1 - 3\boldsymbol{\theta}_2$
- Also, there are four constraints:
  - $\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2 \leq 4$
  - $\boldsymbol{\theta}_2 + \frac{\boldsymbol{\theta}_1}{\boldsymbol{2}} \leq 3$
  - $\boldsymbol{\theta}_1 \geq 0$
  - $\boldsymbol{\theta}_2 \geq 0$
- We start by obtaining the feasible region.

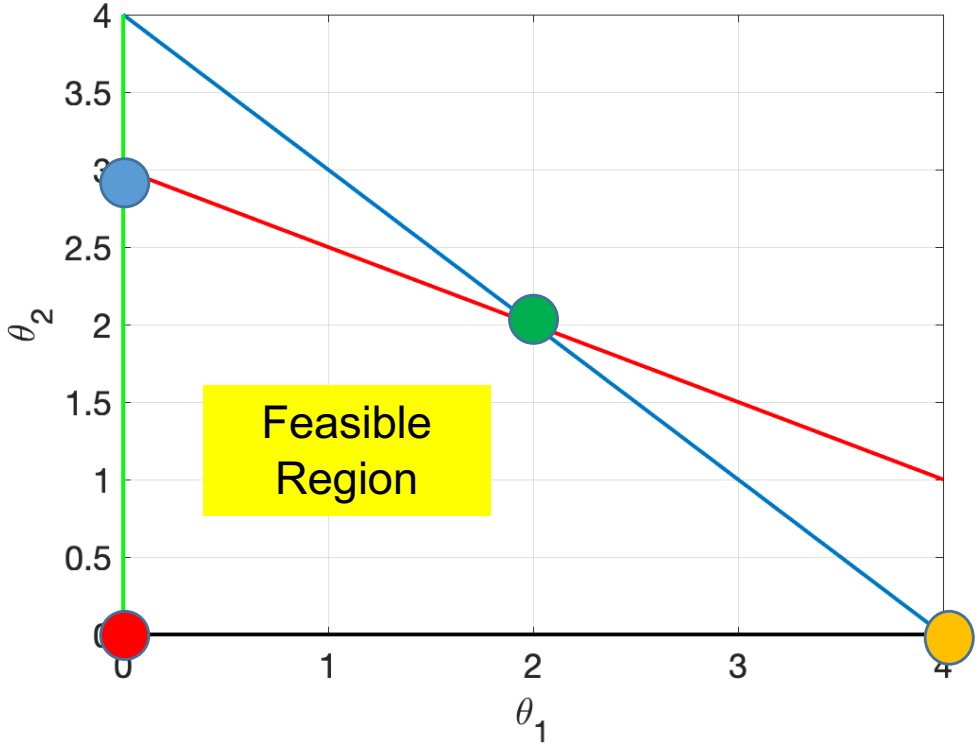


Intersection of all constraints

Linear Programming: Graphical Method Example

- Next, we find the vertices of the polygon that delimitates the feasible region:

  - A= (0,0)
  - B = (0,3)
  - C = (2,2)
  - D = (0,4)



- Finally, we obtain the value of $J(\boldsymbol{\theta})$ for all vertices. The vertex that maximizes $J(\boldsymbol{\theta})$ is the solution.
  - *J(A) = 0-0=0; J(B)=0-9= -9; J(C)=8-6=2;* ***J(D)=0+12=12***
  - $J(\boldsymbol{\theta}^*) = (\theta_1, \theta_2) = (0,4)$

Linear Programming: Graphical Method Exercise

☐ Consider the following optimization problem:

$$J(x_1, x_2) = 40x_1 + 30x_2$$

subject to:

$$x_1 + x_2 \leq 12$$
$$2x_1 + x_2 \leq 16$$
$$x_1 \geq 0, x_2 \geq 0$$

☐ Obtain the optimal configuration for (x_1, x_2) using the graphical method.

Linear Programming: Simplex Method

- The **Simplex** method is mainly used in problems that involve many (>2) decision variables, due to the difficulties of representing such high dimensional spaces in a graphical way.
- The process starts by transforming all inequalities into equalities (using slack variables) and then – using linear algebra – iteratively find pivots and reduce them (***pivoting***) until a solution is reached.
- Example. Consider the optimization problem solved previously:

$$J(x_1, x_2) = 40x_1 + 30x_2$$

subject to:

$$\begin{aligned} x_1 + x_2 &\leq 12 \\ 2x_1 + x_2 &\leq 16 \\ x_1 \geq 0, x_2 &\geq 0 \end{aligned}$$

Linear Programming: Simplex Method

- We start by adding two slack variables (y_1 and y_2) that convert the inbequatity constraints into equalities.

$$40x_1 + 30x_2 + J() = 0$$
$$x_1 + x_2 + y_1 = 12$$
$$2x_1 + x_2 + y_2 = 16$$

- Next, we construct the initial simplex matrix:

x_1	x_2	y_1	y_2	J	
1	1	1	0	0	12
2	1	0	1	0	16
−40	−30	0	0	1	0

- Identify the column with the largest (in magnitude) negative value
  - The first column will be the initial pivot.

Linear Programming: Simplex Method

- We then, divide elements in the final column by the corresponding values in the pivot column.
  - $12/1 = 12$ and $16/2 = 8$

x_1	x_2	y_1	y_2	J	
1	1	1	0	0	12
2	1	0	1	0	16
-40	-30	0	0	1	0

- The row with the smallest coefficient is the pivot row.
- Hence, pivot row and pivot column define the pivot (2).
  - Using Linear Algebra and perform *pivoting (i.e., set all elements in the pivot column equal to 0)*
    - *Divide elements in the pivot row by the pivot value*

x_1	x_2	y_1	y_2	J	
1	1	1	0	0	12
1	$\frac{1}{2}$	0	$\frac{1}{2}$	0	8
-40	-30	0	0	1	0

Linear Programming: Simplex Method

- *Set all values in the pivot column equal to 0 (Row 1 = Row 1 – Row 2 and Row 3 = Row 3 + 40 Row 2)*

x_1	x_2	y_1	y_2	J	
0	$\frac{1}{2}$	1	$-\frac{1}{2}$	0	4
1	$\frac{1}{2}$	0	$\frac{1}{2}$	0	8
0	-10	0	20	1	320

- *Check if the last row has negative values.*
  - *If yes (in this case -10) we repeat the process and find the column with the largest (in magnitude) negative value.*
  - *If not, the process stops.*

Linear Programming: Simplex Method

- *Repeating the previous steps, we obtain the following matrix:*

x_1	x_2	y_1	y_2	J	
0	1	2	−1	0	8
1	0	−1	1	0	4
0	0	20	10	1	400

- *The final row is written in equation form, and we get:*

$$J = 400 - 20y_1 - 10y_2$$

- Hence, J = 400 is the maximum value we can get
  - when $y_1 = y_2 = 0$
- Also, $x_1 = 4$ *and* $x_2 = 8$ *is the solution to our problem.*

Linear Programming: Simplex Exercise

- Solve the following Linear Programming problem through the Simplex Method:

$$\text{maximize } J(x_1,x_2,x_3) = 3x_1 + x_2 + 3x_3$$

subject to:

$$2x_1 + x_2 + x_3 \leq 2$$

$$x_1 + 2x_2 + 3x_3 \leq 5$$

$$2x_1 + 2x_2 + x_3 \leq 6$$

$$x_1 , x_2 , x_3 \geq 0$$

Genetic Algorithms

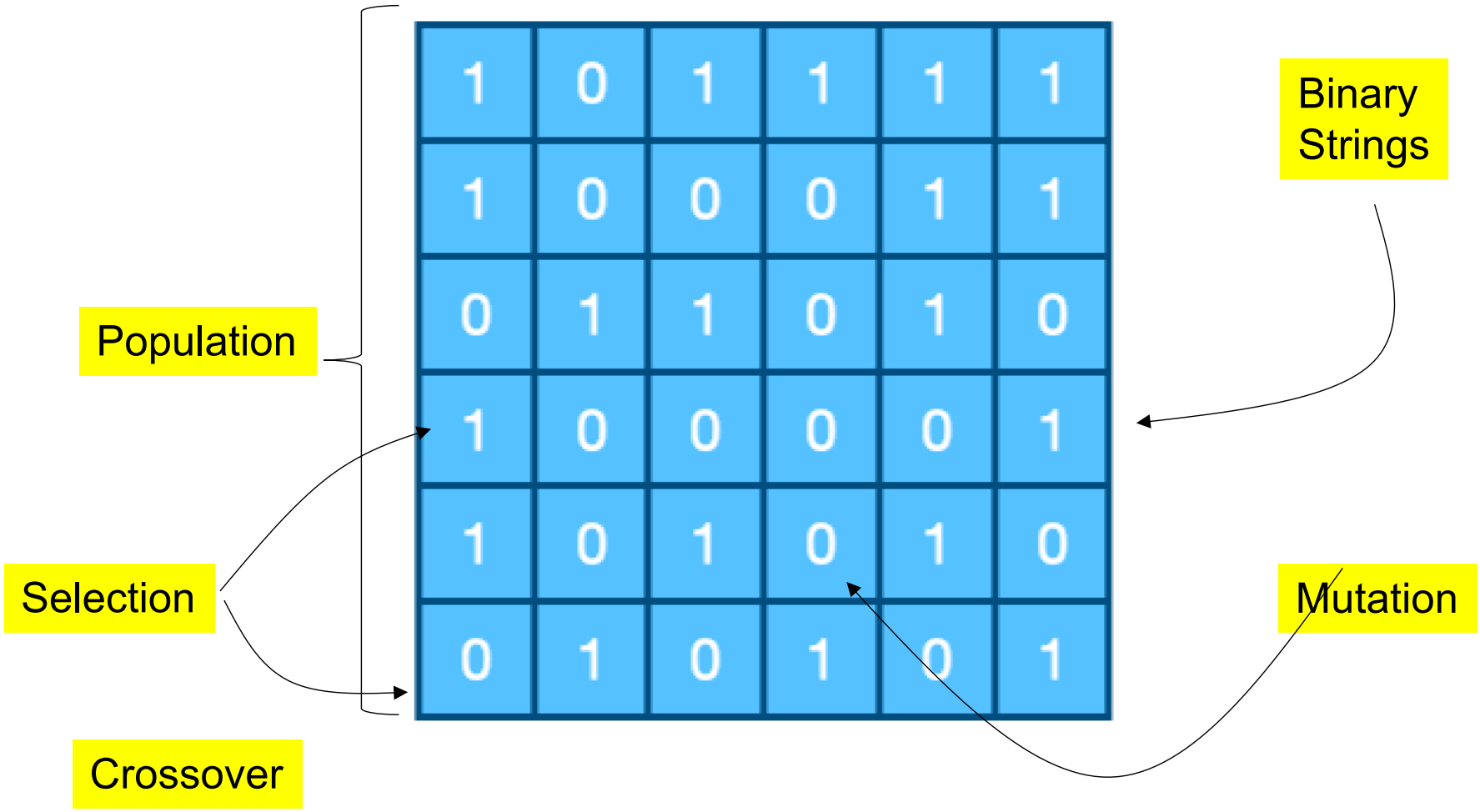
- **Genetic Algorithms** (Gas) are a very popular choice, among search-based optimization methods.
  - The term optimization refers to find the parameters of a model to get the best output values.
  - In practice, it refers to maximizing/minimizing one objective function, by varying the possible values for the parameters.
- **GAs** were proposed by John Holland and David E. Goldberg (1970s) and have been used in various optimization problems with high success.
  - Traveling Salesman Problems
    - Used to find an optimal way to be covered by the salesman, in a given map with the routes and distance between two points.
  - Vehicle Routing Problems
    - Used to find an optimal weight of goods to be delivered or an optimal set of delivery routes when other things like distance,
  - Finantial Markets
    - Used to find optimal set or combination of parameters that can affect the market rules and trades.
  - Medical Science
    - Used in predictive analysis like RNA structure prediction, operon prediction, and protein prediction

Genetic Algorithms

- Genetic Algorithms are typically used in problems that are nonlinear and where there are multiple correlations between parameters.
  - This implies that it is not possible to treat each parameter as an independent variable which can be solved in an independent way from the other variables
- The first assumption typically made is that the **variables** representing the parameters can be represented by **bit strings**
  - This means that the variables are discretized in an a priori fashion and that the range of the discretization corresponds to some power of 2.
  - For example, using 10 bits per parameter, it is possible to represent $2^{10} = 1024$ values.
- Then, there is also an **evaluation function**, usually given as part of the problem description

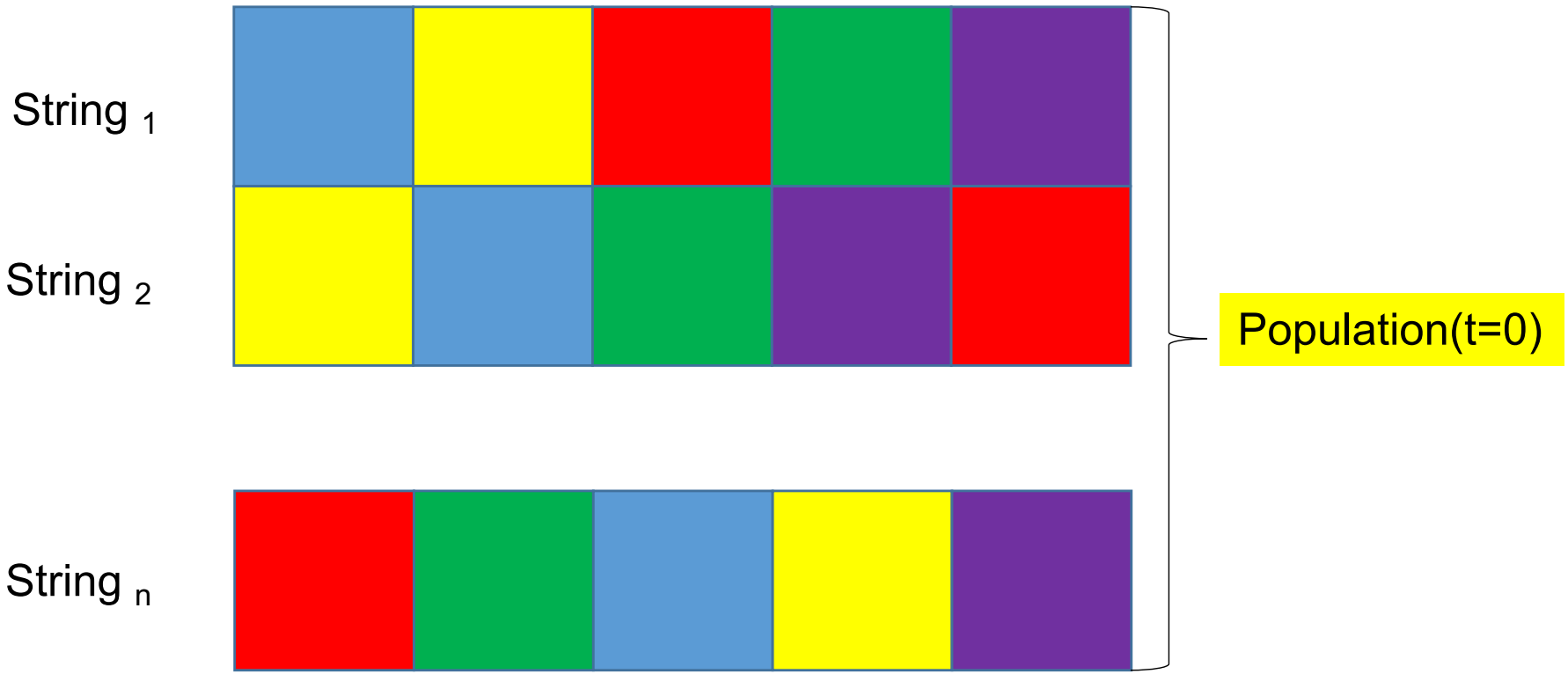
Genetic Algorithms

☐ There are some terms used in the context of **GAs:**



Genetic Algorithms

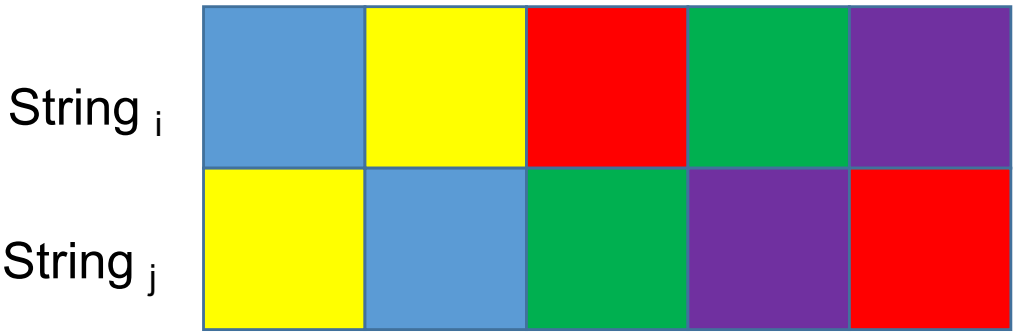
- **Step 1**: We start with a (random?) population, composed of "n" elements (strings):

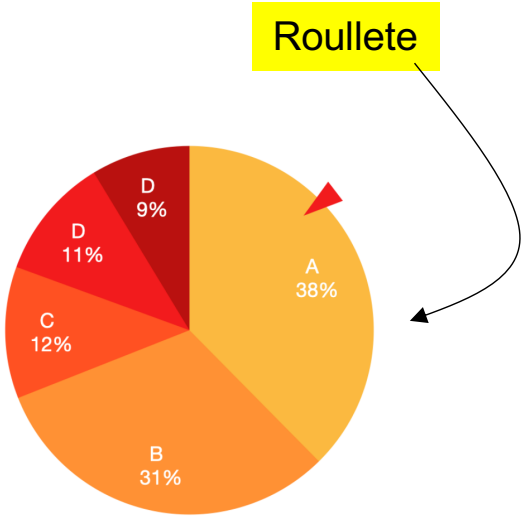


Genetic Algorithms

- **Step 2 – Selection.** After obtaining the values of the objective function f(String $_i$), $\forall\, i \in \{1, \ldots, n\}$, we select k (out of n) elements to be used as parents for the next generation.
- There are three main ways to perform selection:
  - **Random Selection**, by randomly choosing pairs of elements, without the effect of fitness values.
  - **Tournament Selection**, by randomly sampling from the population and then, select with probabilities p, p(1-p), $p(1-p)^2$,… the best, second-best, third,… elements.
  - **Roulette Wheel**, based on the fitness of each element. The size of the proportion of elements in the roulette wheel varies depending on the fitness value. The selection is made by raising a random value from the range of all fitness values.

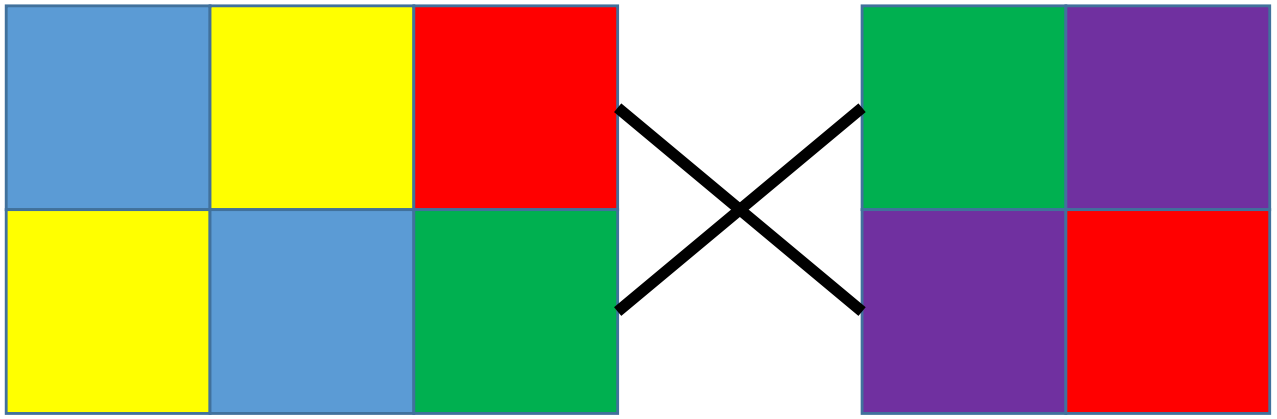
There are different variants in this step, as "**Elitism**", that guarantees that the best individual(s(are always selected for the next generation



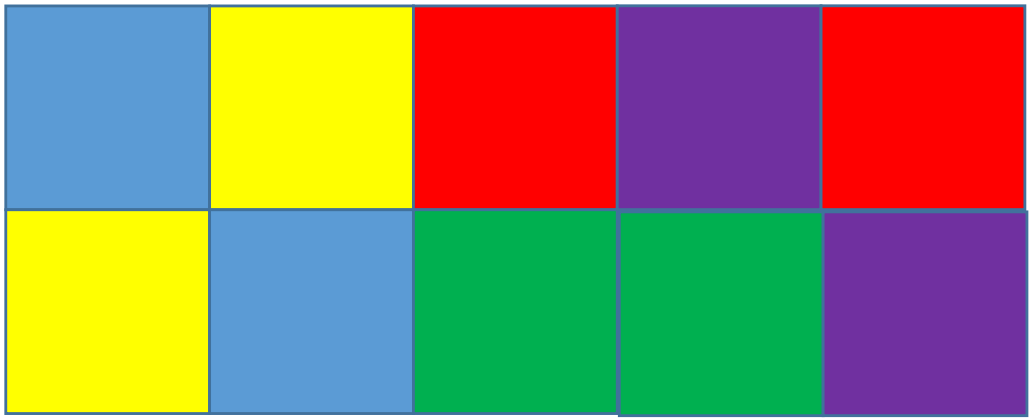


Genetic Algorithms

- **Step 3 – Crossover.** After selecting the parents, crossover is used for producing new elements. We randomly (or not) select the position where both elements will be swapped



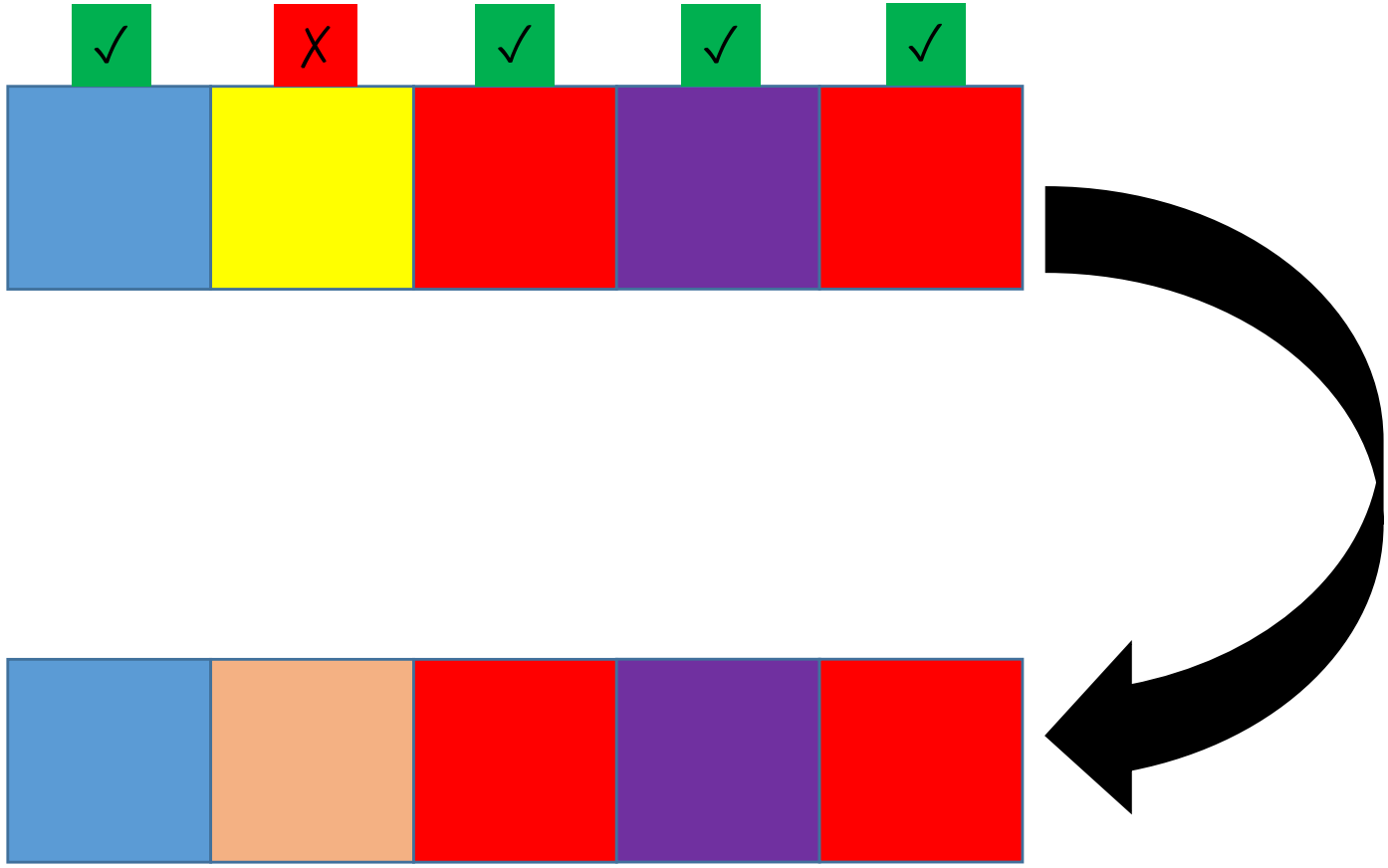
Parents



New elements

Genetic Algorithms

- **Step 4 – Mutation.** Each position of the newly generated elements will be mutated (randomly changed) with some probability



Genetic Algorithms: Exercise

- Consider the following population (at time t=0). Using the evaluation function f() below, write the population at time t=1, when…
  - Pairs (0, 3), (1, 5), (0, 1), (4, 3), (0, 2), (2, 4) are used for selection, respectively at positions [2,1,3,2,4,2]
  - For each new element, respectively the (1), (0), (1,2), (4,5), (3) and (2) bits will be mutated.

$$f(b_0b_1b_2b_3b_4b_5) = \sum_{i=0}^{5} b_i^i$$

- Which population (t=0 or t=1) is "*better*"? (i.e., has a higher average f() value)

	[b_0]	[b_1]	[b_2]	[b_3]	[b_4]	[b_5]
[0]	1	0	1	1	1	1
[1]	1	0	0	0	1	1
[2]	0	1	1	0	1	0
[3]	1	0	0	0	0	1
[4]	1	0	1	0	1	0
[5]	0	1	0	1	0	1