COMPX216-24A
Artificial Intelligence

Local Search in Continuous Spaces

Today: Local Search in Continuous Spaces

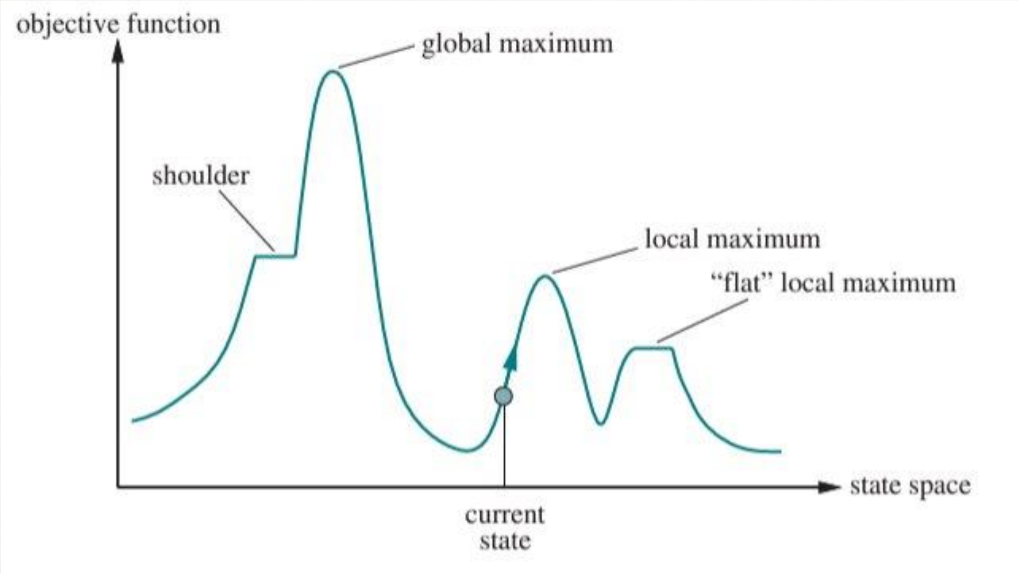
- Continuous state spaces are represented using vectors
- Gradient descent to find a local minimum
- Gradient descent when the state is a vector
- Gradient vectors represent directions
- Showing gradients as a vector field
- A function with local minima
- Going beyond basic gradient descent

Continuous state spaces

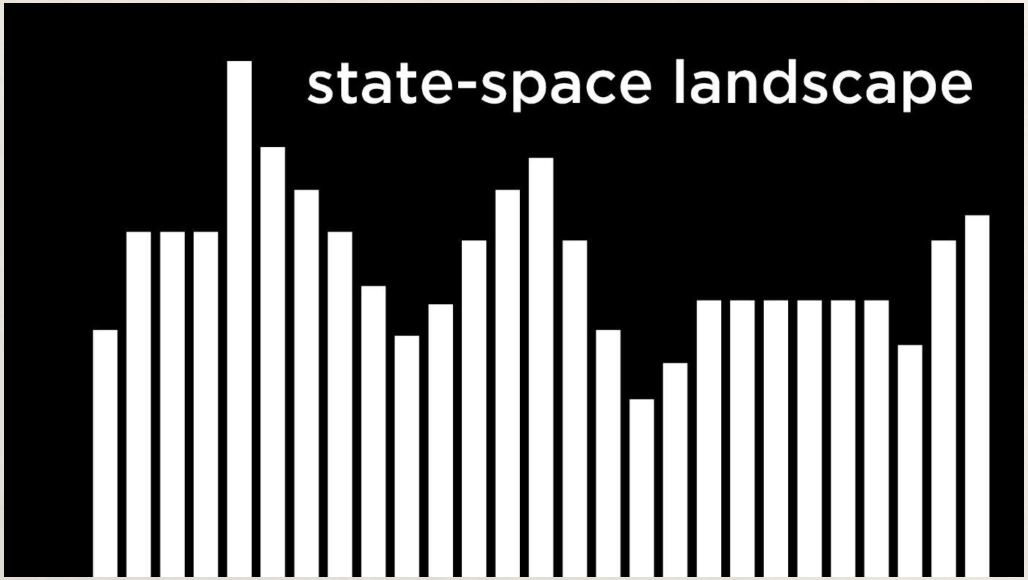
- **Continuous space:** the values of a variable (or coordinate) can vary smoothly and take any value within a range, typically without gaps
- Represented by real numbers, where the number of possible values between any two numbers is infinite.
- Example: Real numbers ($\mathbb{R}$) are continuous. Between any two real numbers, no matter how close they are, there are infinitely many other real numbers (e.g., between 1 and 2, there are values like 1.1, 1.01, 1.001, and so on).

Continuous vs Discrete state spaces

Continuous



Discrete

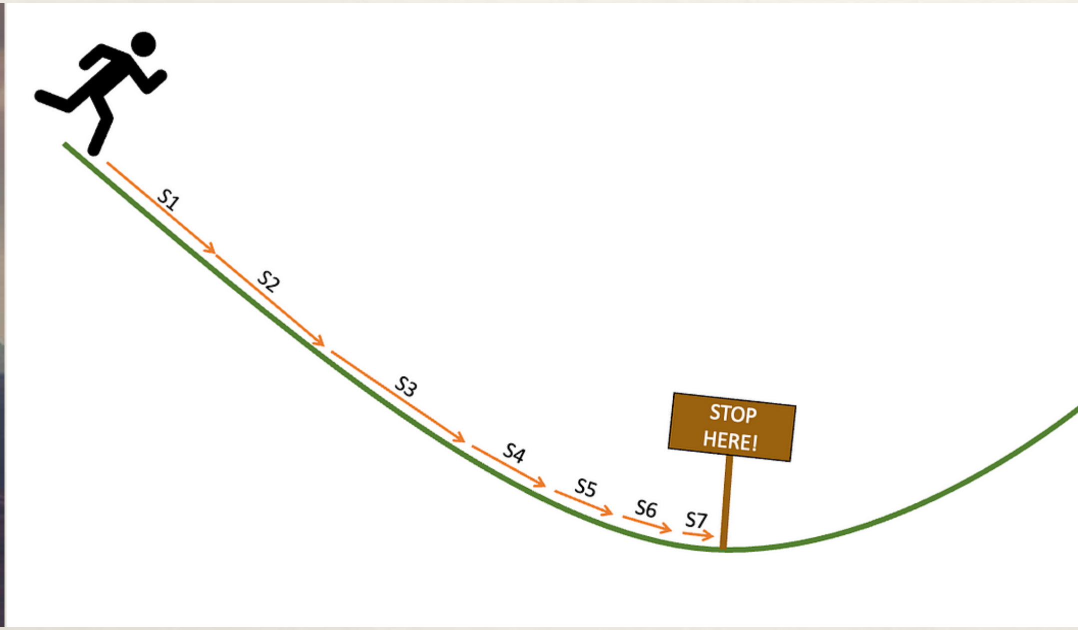


Continuous state spaces

- In continuous state spaces, the information describing a state is given as a list of real numbers, a **vector**
  - a direction and a distance in the continuous state space
- For example, assume we want to optimise the growing conditions for a plant by setting temperature and humidity
- Each state of the environment consists of two real numbers in this case, one for temperature and one for humidity
- We can use x_1 and x_2 to refer to these two real numbers and $\mathbf{x}$ to refer to the vector comprising both
- The objective function will be a continuous function f depending on x_1 and x_2, also called **variables** in this case
- We would like to optimise $f(x_1,x_2)$, also written succinctly as $f(\mathbf{x})$, to find a good state of the environment

A journey down the hill

- Imagine you're **standing at the top of a hill**.
    - You can't see the bottom, but you can feel the **slope** under your feet.
    - You want to **move downhill**.
    - You **measure the slope** beneath you and take a step in the direction where the slope is **the steepest downhill**.

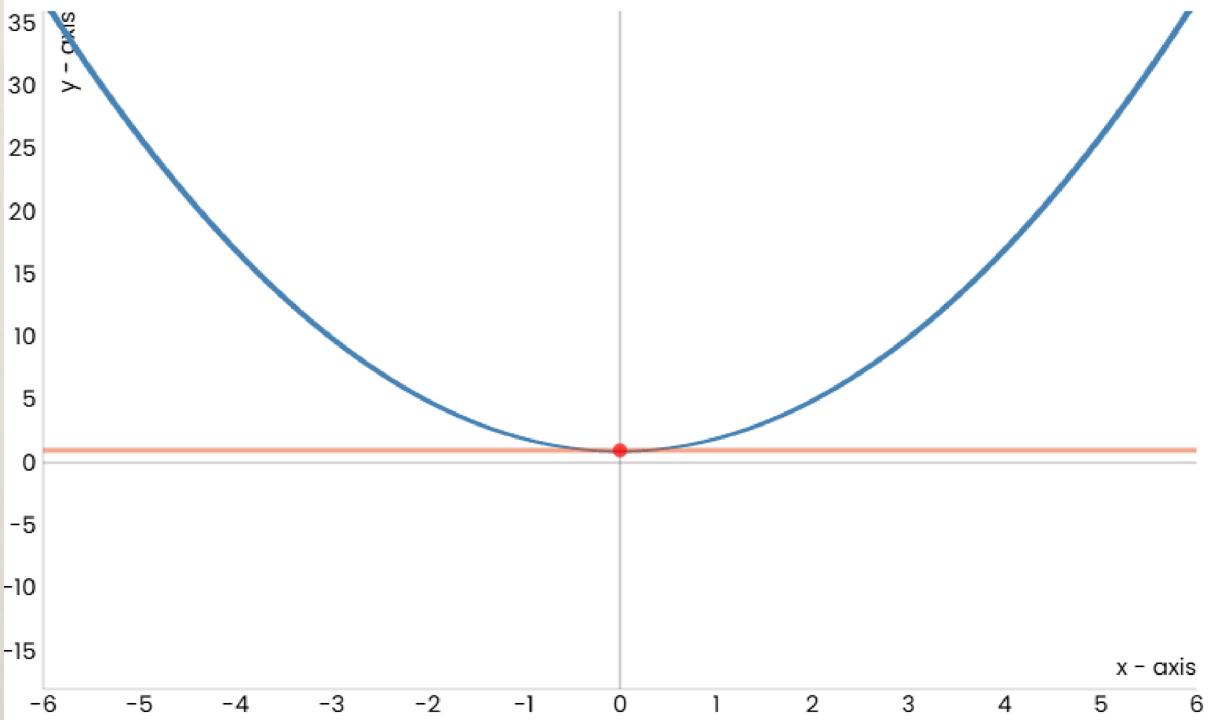


https://pyimagesearch.com/2021/05/05/gradient-descent-algorithms-and-variations/

https://medium.com/@kaineblack/gradient-descent-simply-explained-75b11732f20a

A mathematical hill

- Let's define a **simple hill** as the function: $f(x) = x^2+1$
- **Shape of the hill:** This is a **U-shaped curve**.
- The **lowest point** is at $x=0$, where $f(0)=1$.
- The further you move from $x=0$, the **higher the function gets**.



Understanding the slope (gradient)

- Gradient: The gradient tells us the slope of the function at any point.

- The gradient is given by the **derivative** of the function, which can be found algorithmically using rules from calculus

- In our example, the gradient is the derivative of $f(x)$: $f'(x)=2x$

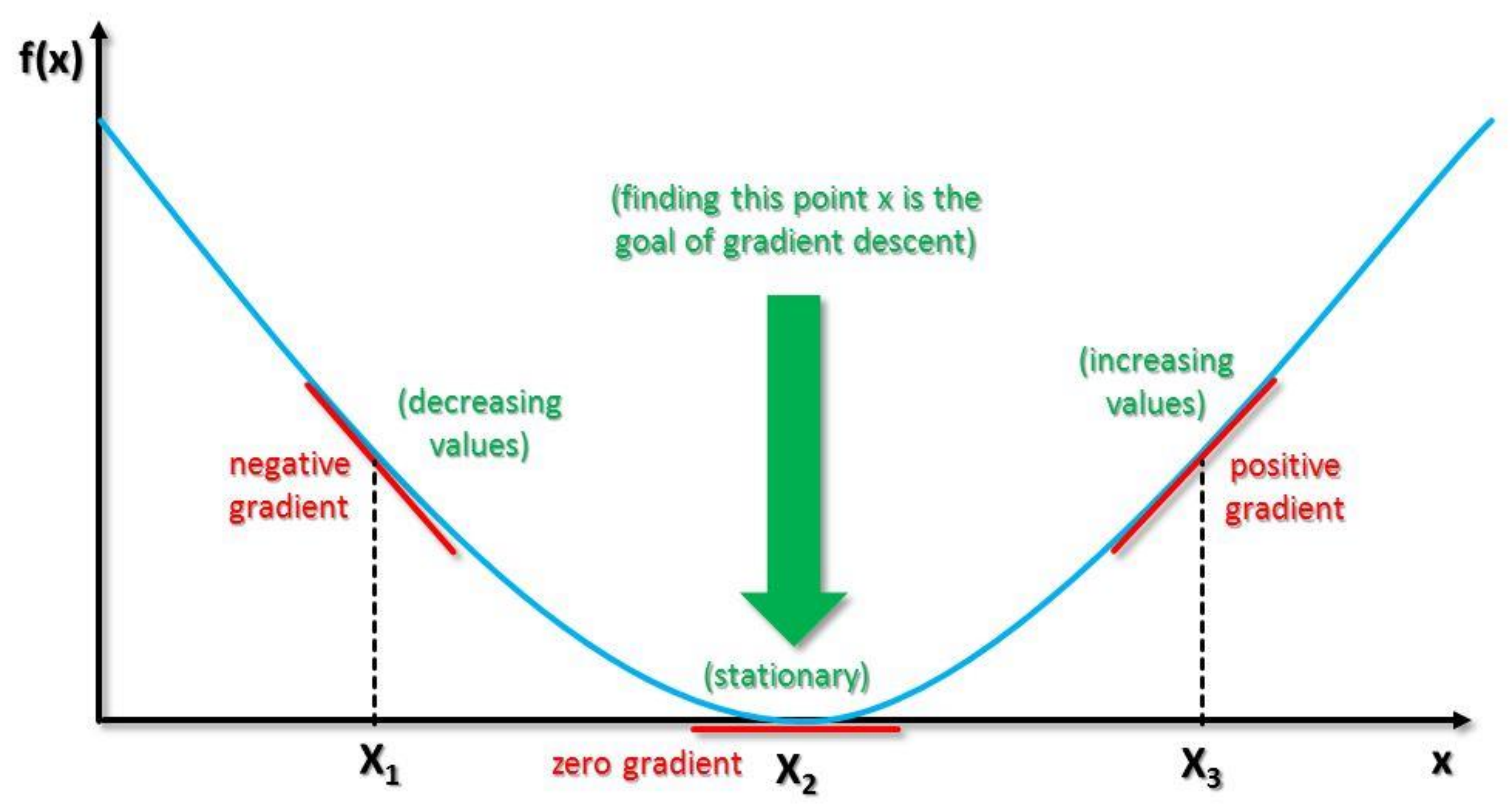
- The gradient tells us:
  - The direction of steepest ascent (i.e., uphill).
  - To go downhill, we move in the opposite direction.

- The gradient of a function is thus positive where the function is increasing and negative where the function is decreasing

- At $\boldsymbol{x=3}$, $\boldsymbol{f'(3)=6}$, meaning the gradient is **positive**, and we need to move left (opposite of the gradient).

- At $\boldsymbol{x=-3}$, $\boldsymbol{f'(-3)=-6}$, meaning the gradient is **negative**, and we need to move right.

Understanding the slope (gradient)



http://www.big-data.tips/gradient-descent

Gradient descent

- Method to find local optima of a differentiable function, ***f(x)***
- The idea of gradient descent is to update your position by taking a step in the opposite direction of the gradient.
- Intuition: gradient tells us direction of greatest increase, negative gradient gives us direction of greatest decrease
  - Take steps in directions that reduce the function value
- Definition of derivative guarantees that if we take a small enough step in the direction of the negative gradient, the function will decrease in value

The Gradient descent formula

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

- where γ_t is the t^{th} step size (sometimes called learning rate)
- $\nabla f(x_t)$ is the gradient at the current position i.e., x_t
- $x_{(t+1)}$ is the updated position

The Gradient descent algorithm

Gradient Descent Algorithm:

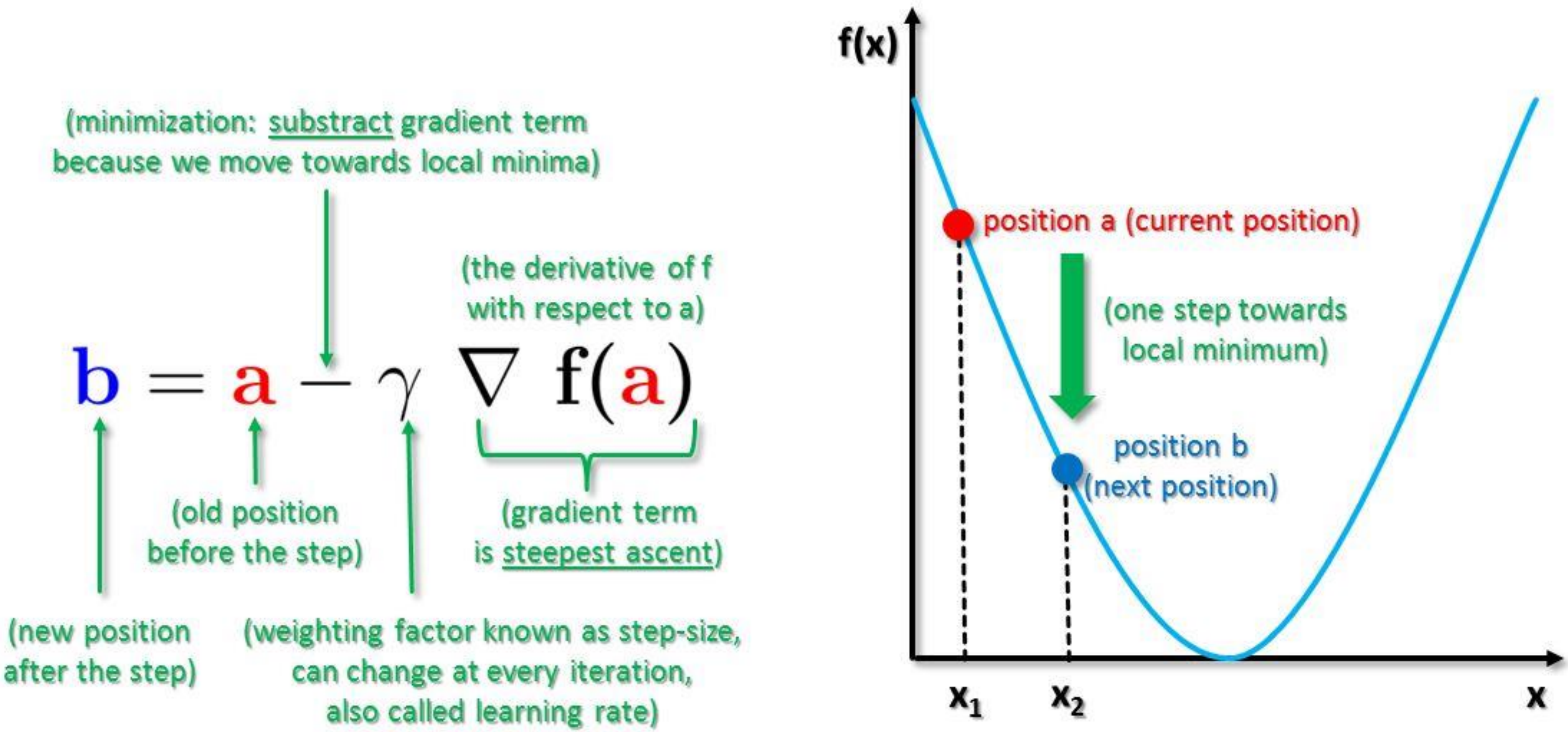
- Pick an initial point x_0

- Iterate until convergence

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

Possible Stopping Criteria: iterate until
$\| \nabla f(x_t) \| \leq \epsilon$ for some $\epsilon > 0$

The Gradient descent formula

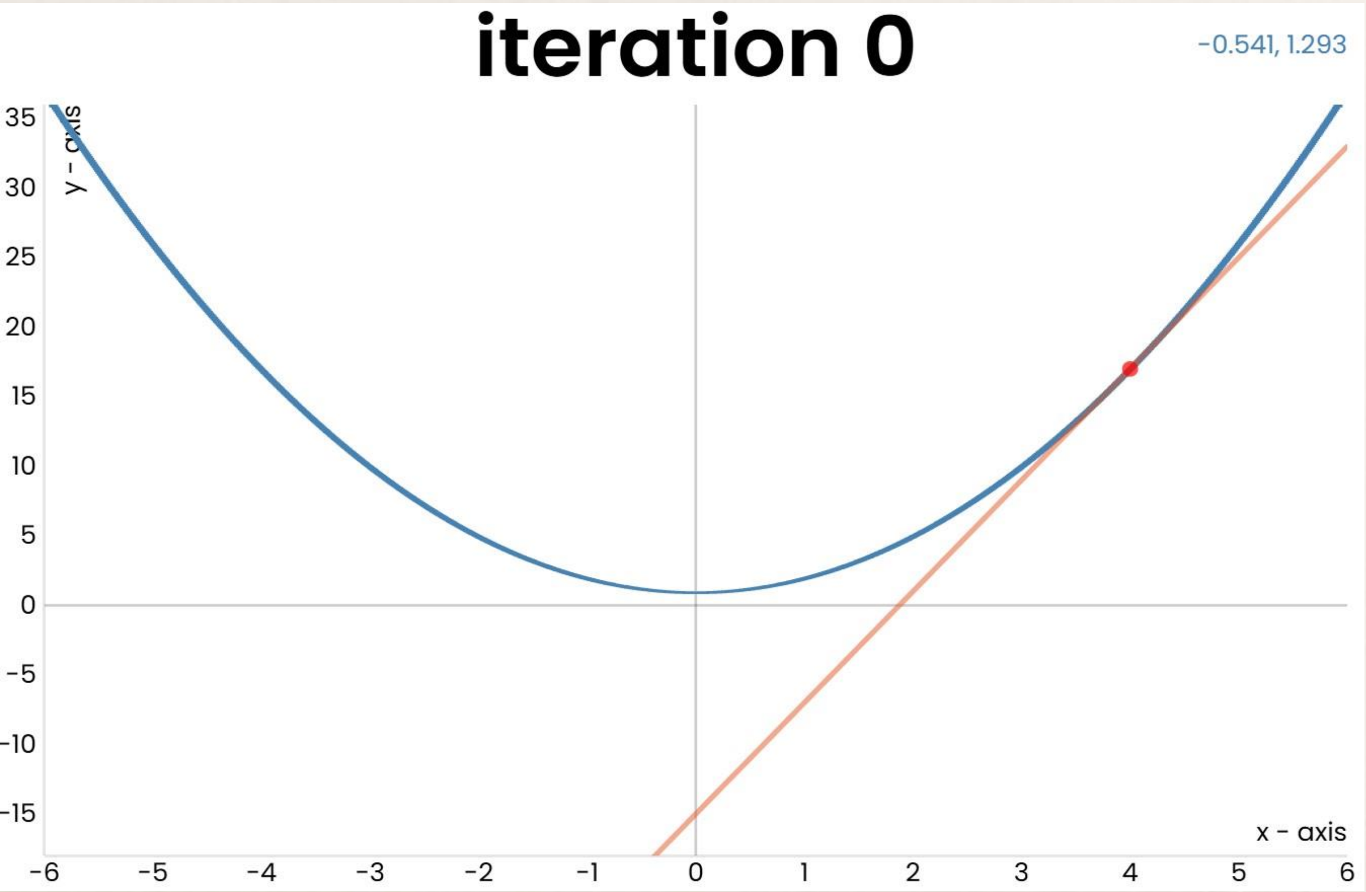


http://www.big-data.tips/gradient-descent

Gradient descent in one-dimensional state space

- Example objective function: x^2+1
- Derivative: $2x$
- Learning rate: 0.1
- Initial state: 4
- Update rule: $x \leftarrow x - 0.1 \times 2x$

Gradient descent in one-dimensional state space

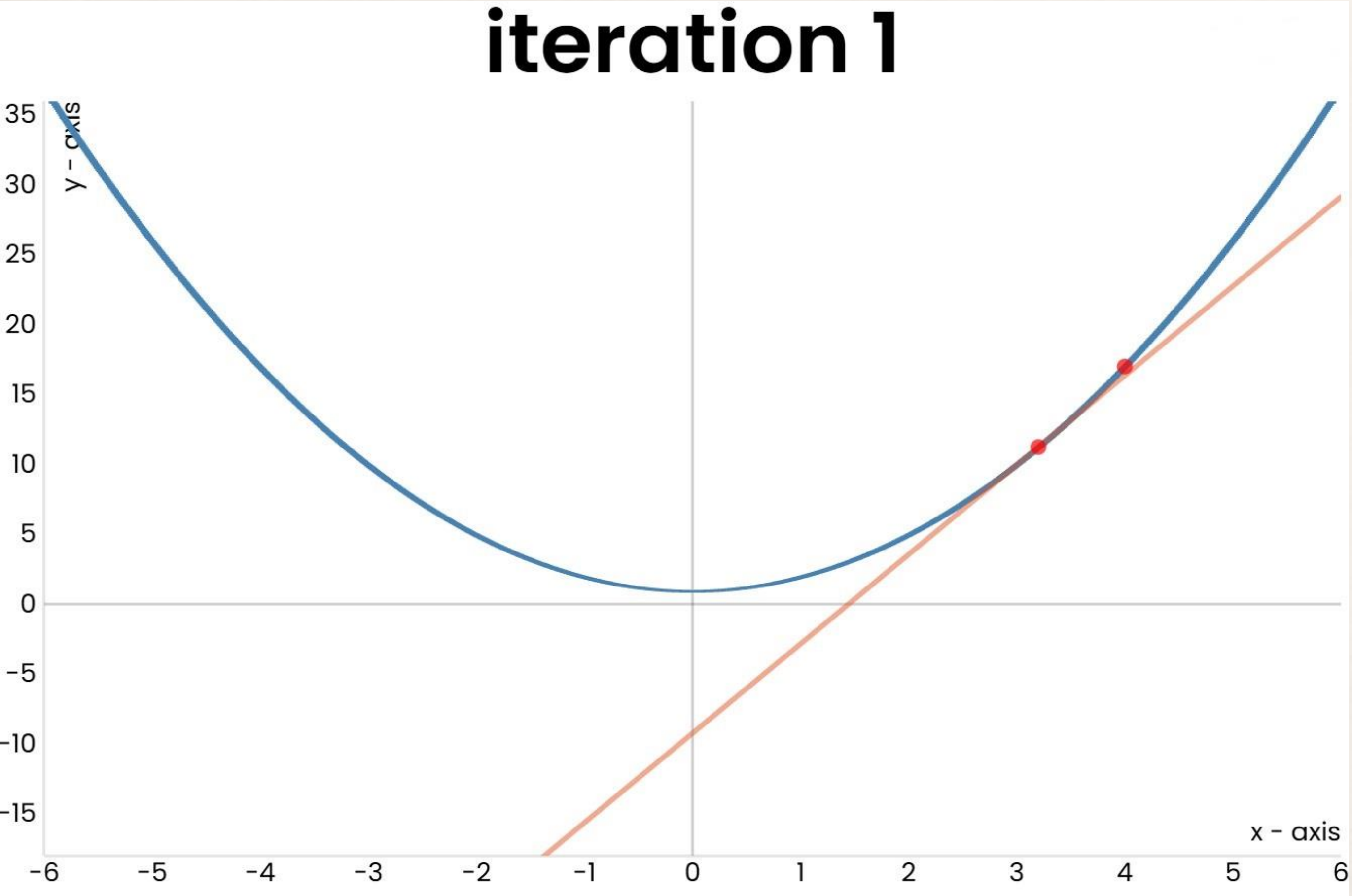


$f(x) = x^2+1$

Learning rate: 0.1

$x_0 = 4$

Gradient descent in one-dimensional state space



$f(x) = x^2+1$

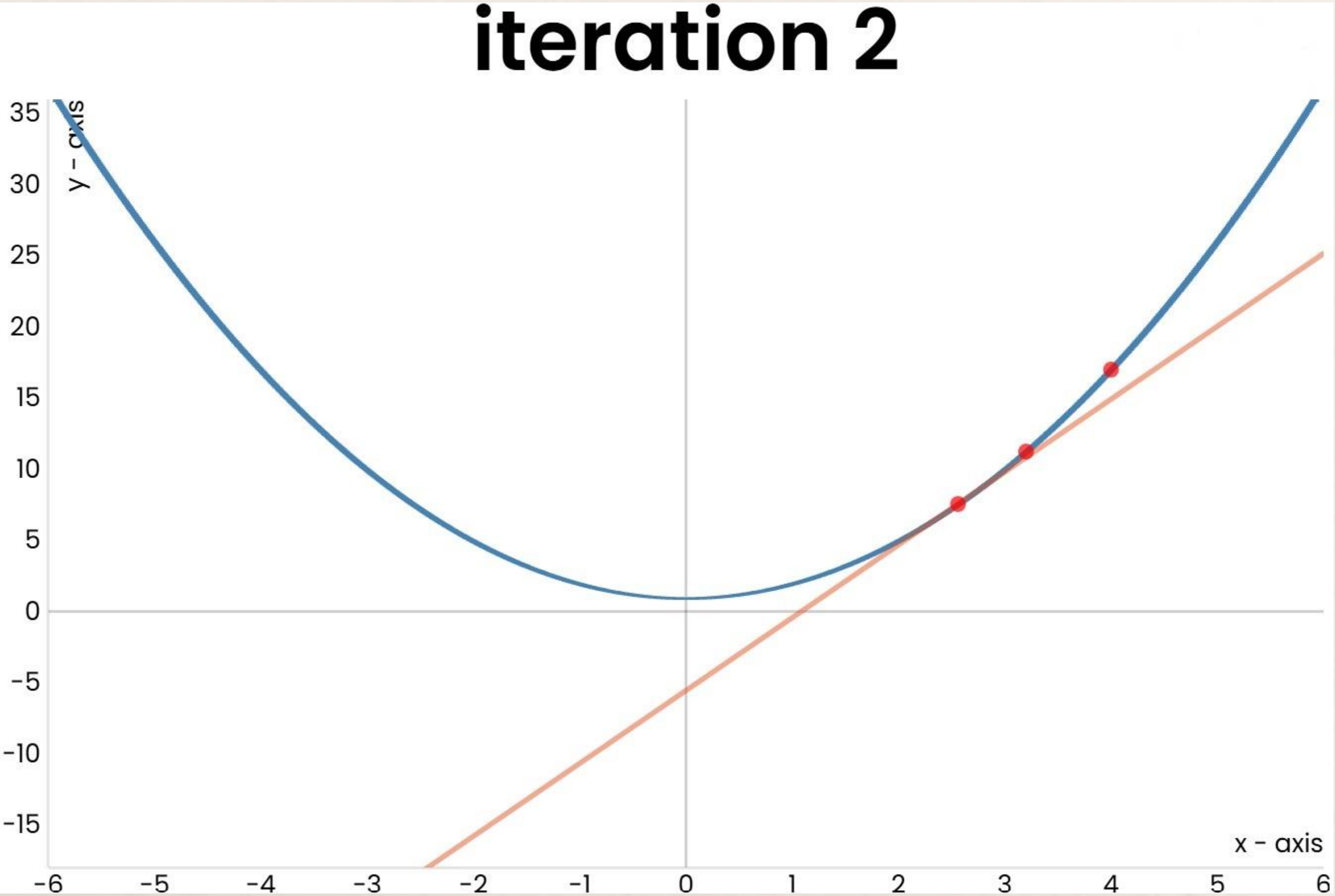
Learning rate: 0.1

$x_0 = 4$

$x_1 \leftarrow 4 - 0.1 \times 2(4)$

$x_1 \leftarrow 3.2$

Gradient descent in one-dimensional state space



$f(x) = x^2+1$

Learning rate: 0.1

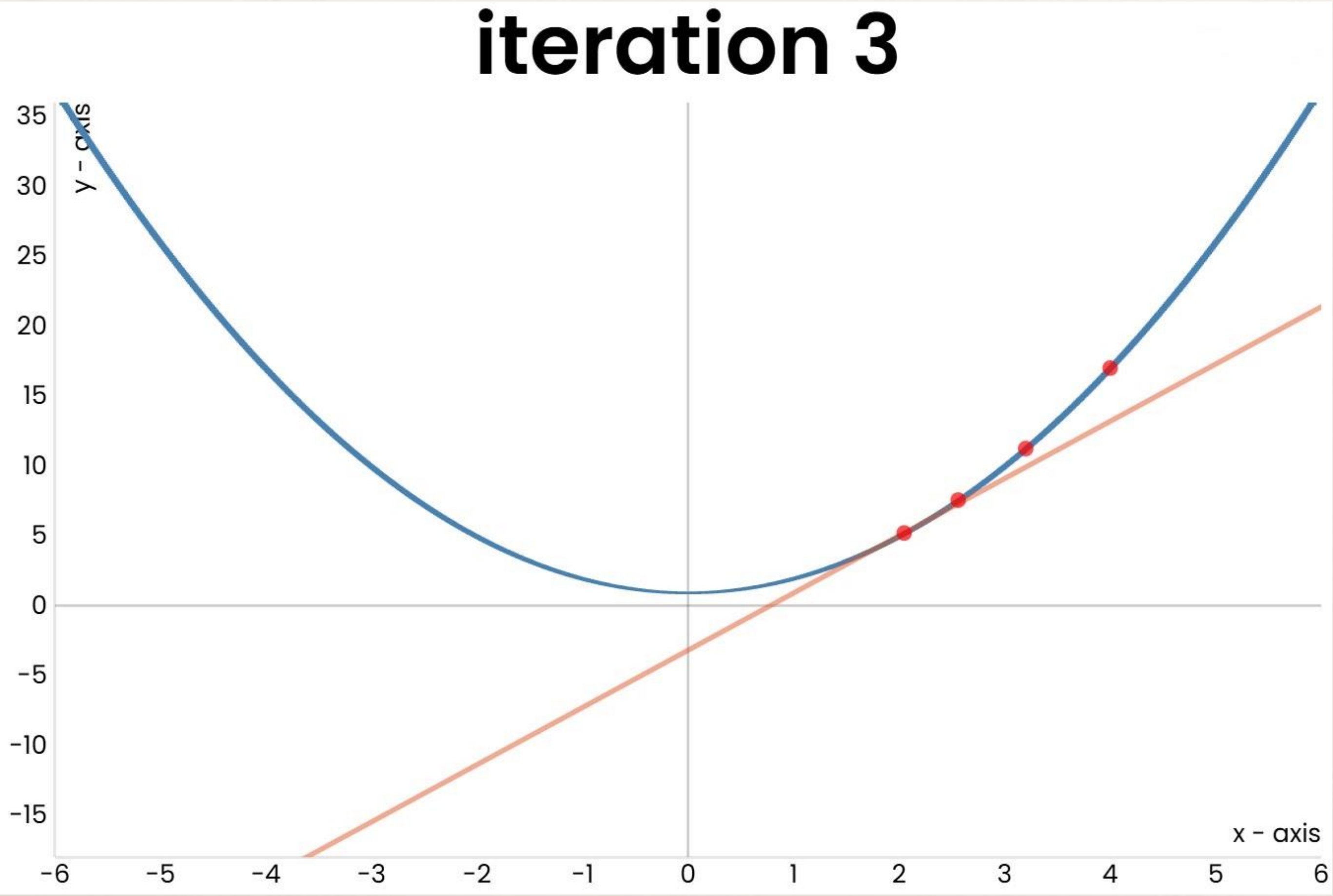
$x_0 = 4$

$x_1 = 3.2$

$x_2 \leftarrow 3.2 - 0.1 \times 2(3.2)$

$x_2 \leftarrow 2.56$

Gradient descent in one-dimensional state space



$f(x) = x^2+1$

Learning rate: 0.1

$x_0 = 4$

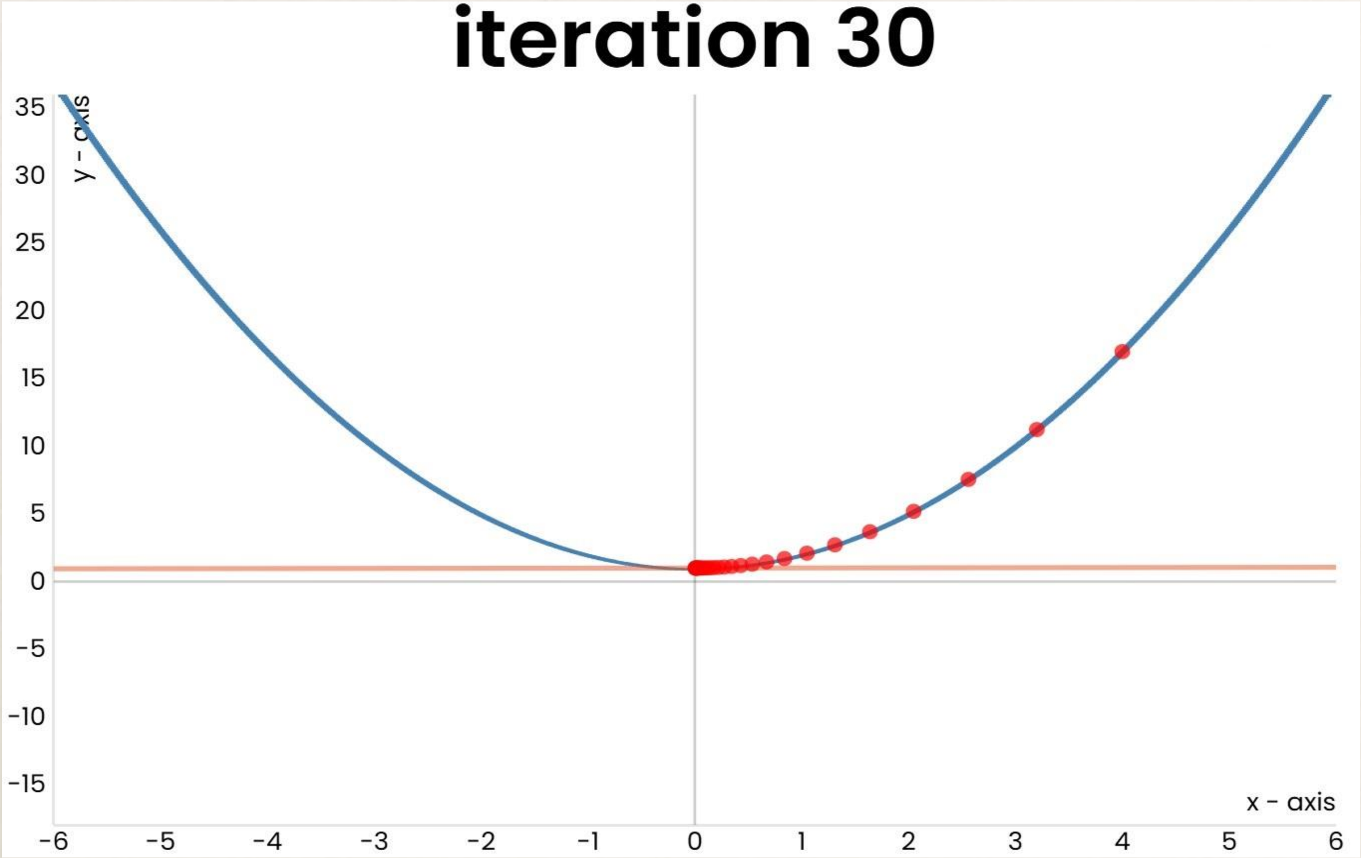
$x_1 = 3.2$

$x_2 = 2.56$

$x_3 \leftarrow 2.56 - 0.1 \times 2(2.56)$

$x_3 \leftarrow 2.048$

Gradient descent in one-dimensional state space



$f(x) = x^2+1$

Learning rate: 0.1

$x_0 = 4$

$x_1 = 3.2$

$x_2 = 2.56$

$x_3 = 1.6384$

$x_4 = 1.31072$

$x_5 = 1.048576$

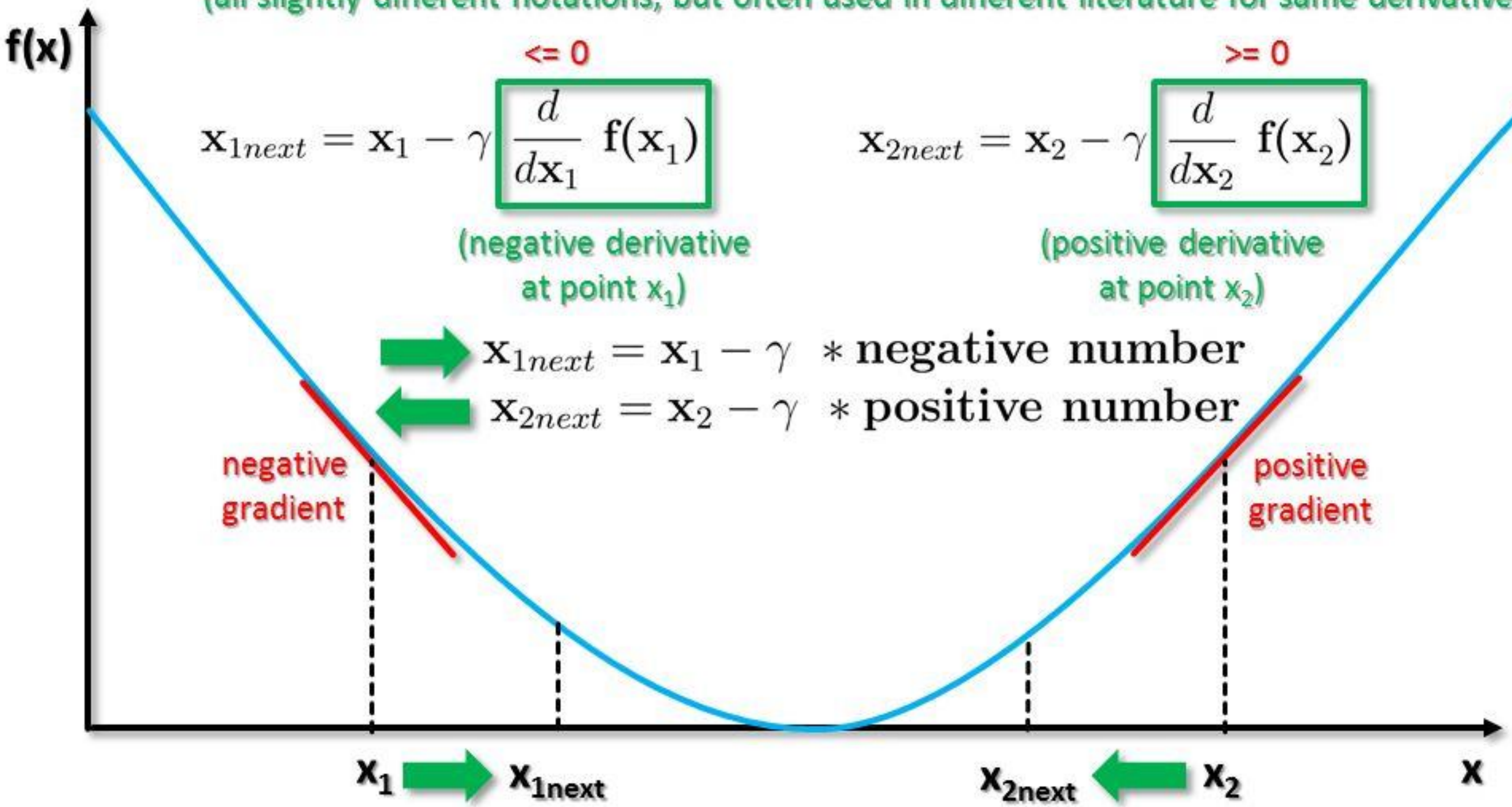
-
-
-
-

$x_{29} = 0.00495176$

Gradient descent in one-dimensional state space

$$\mathbf{b} = \mathbf{a} - \gamma \nabla \mathrm{f}(\mathbf{a}) \qquad \mathbf{b} = \mathbf{a} - \gamma \frac{\partial}{\partial \mathbf{a}} \mathrm{f}(\mathbf{a}) \qquad \mathbf{b} = \mathbf{a} - \gamma \frac{d}{d\mathbf{a}} \mathrm{f}(\mathbf{a})$$

(all slightly different notations, but often used in different literature for same derivative term)



http://www.big-data.tips/gradient-descent

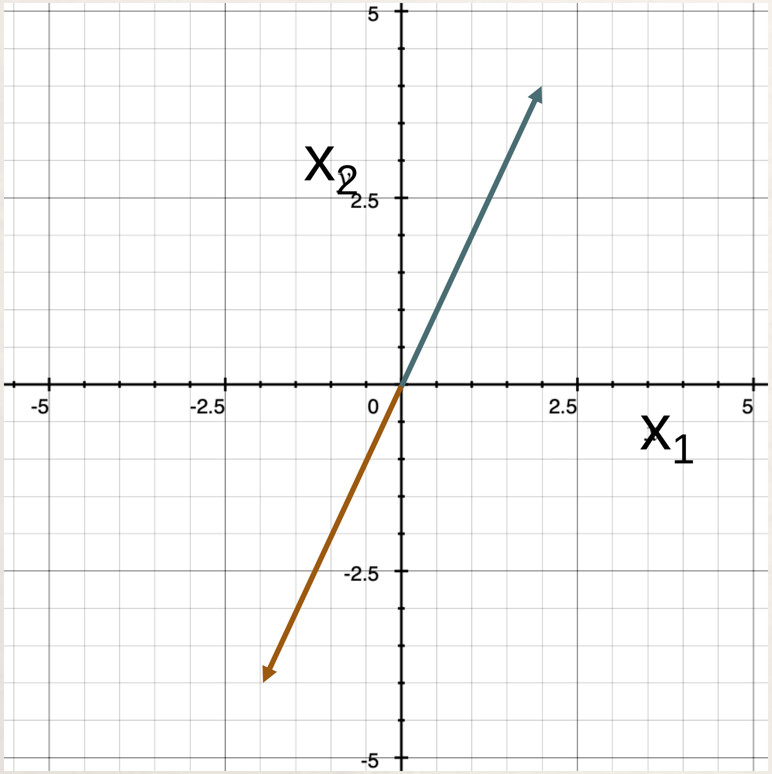
More than one dimension

- In most practical problems, states are represented by a vector **x**
- In this case, the gradient of $f(\mathbf{x})$ at **x** is also a vector, representing the direction of **steepest ascent**, denoted by $\nabla f(\mathbf{x})$
- Example objective function in 2D state space: $f(x_1, x_2) = x_1^2 + x_2^2$
- The gradient vector (the vector of **partial derivatives**) has the following two elements: $\nabla f(x_1, x_2)_1 = 2x_1$ and $\nabla f(x_1, x_2)_2 = 2x_2$
- To apply gradient descent here, we modify each of the two components of the state separately using its partial derivative:

$$x_1 \leftarrow x_1 - 0.1 \times 2x_1$$
$$x_2 \leftarrow x_2 - 0.1 \times 2x_2$$

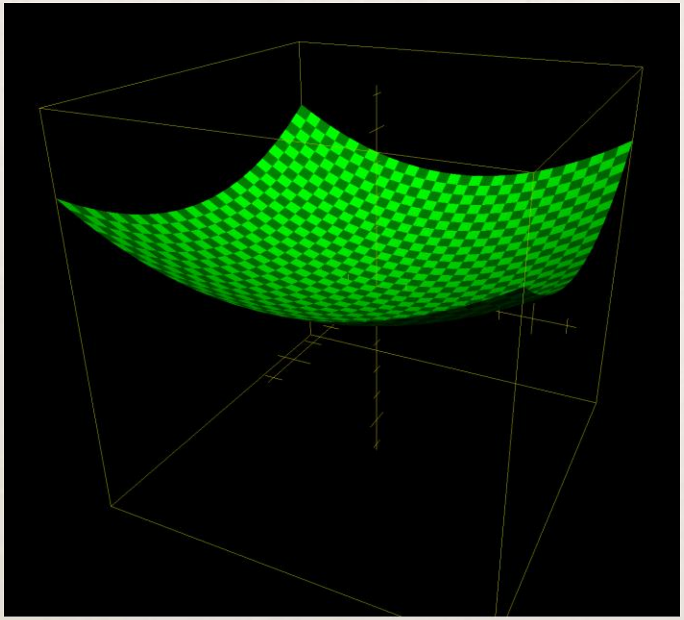
A vector represents a direction

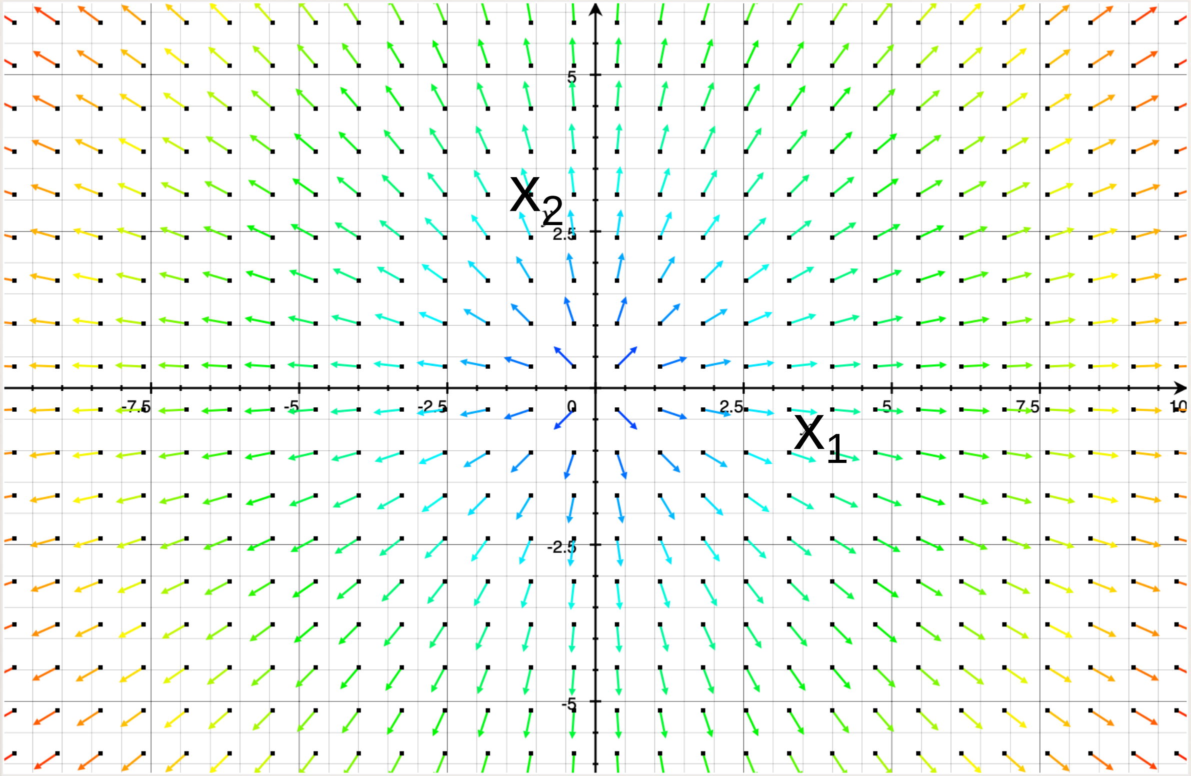
- Showing the directions represented by the vector (2, 4) and the vector (-2, -4):



Showing the gradients as a vector field

- We can compute the gradient for a grid of points in state space and show the corresponding directions using a **vector field**

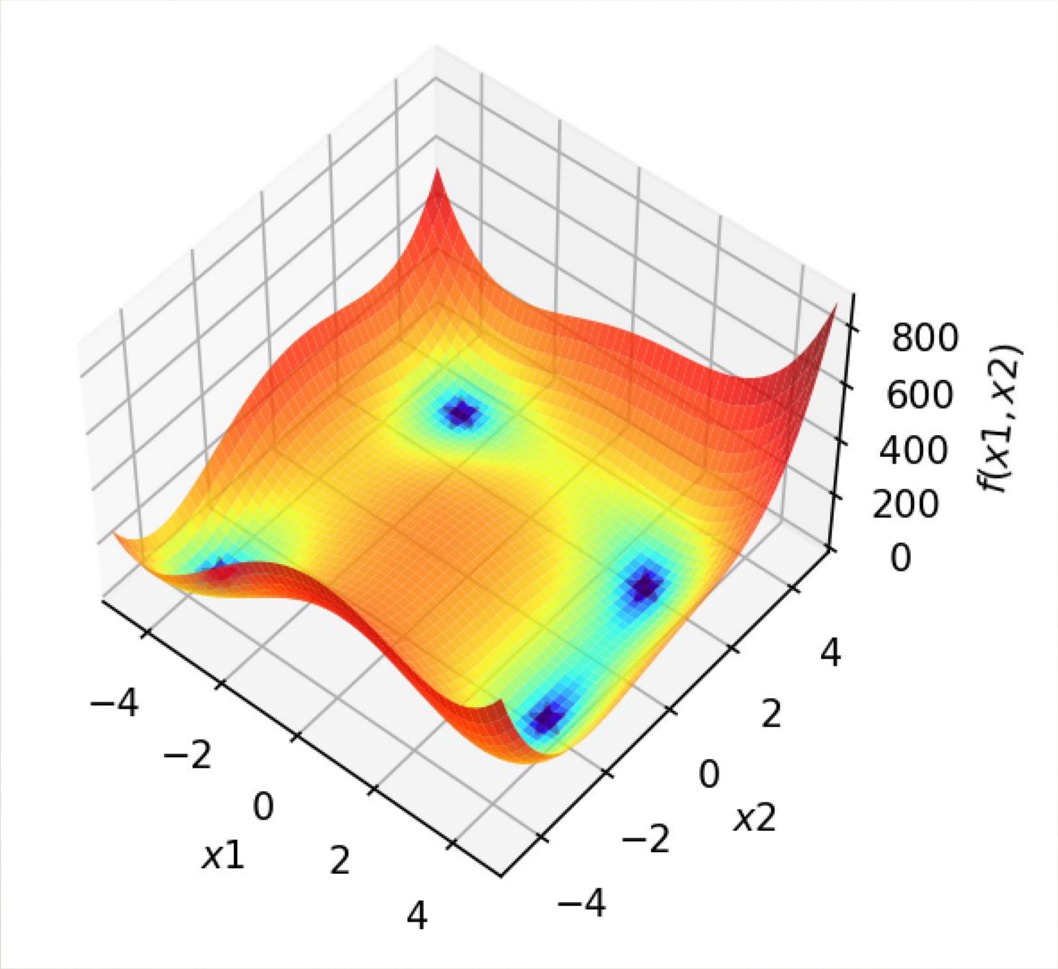




- Every vector shows the gradient, i.e., the direction of **steepest ascent,** at the corresponding point in state space

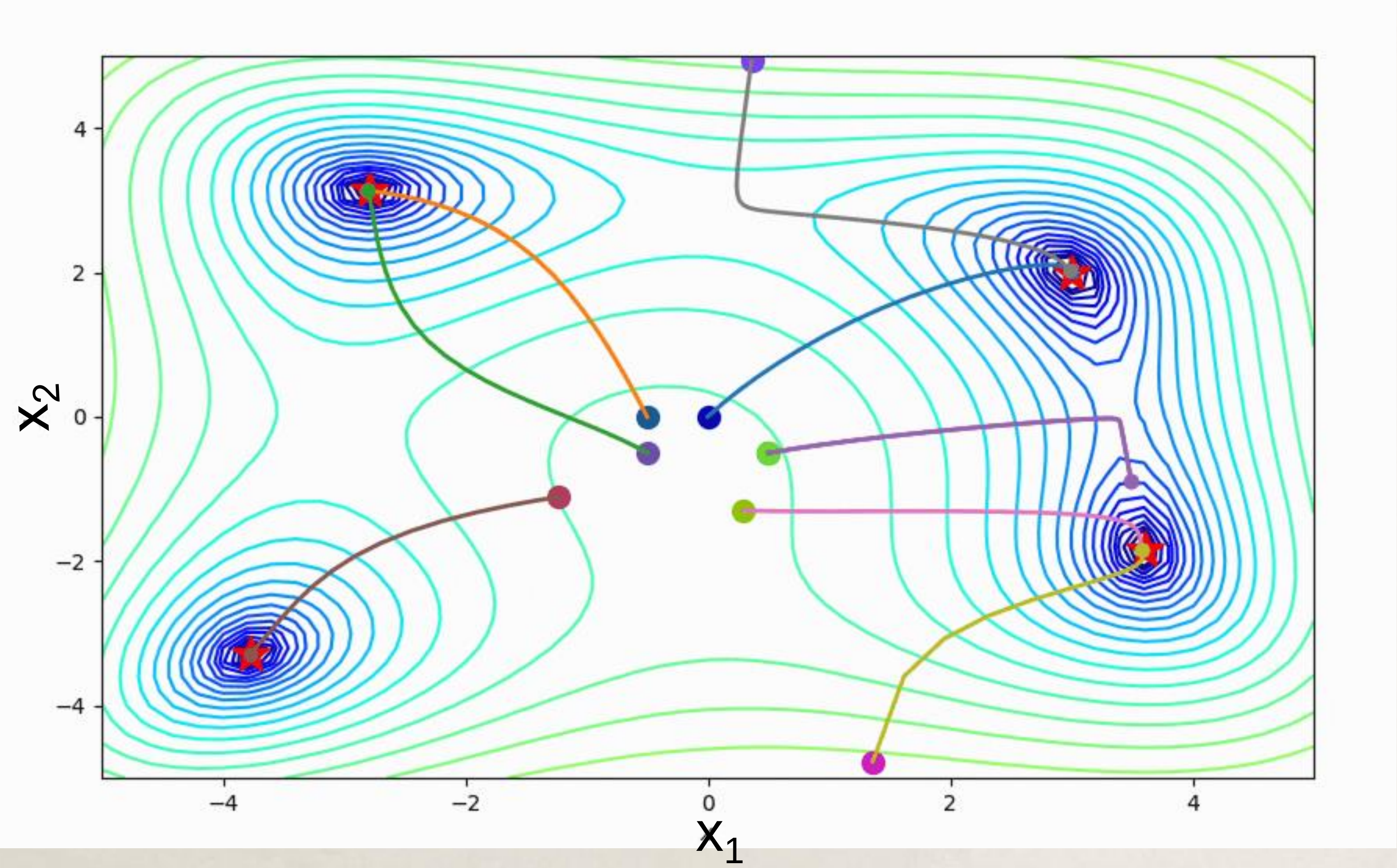
A function with multiple local minima

- Himmelblau's function is $f(\mathbf{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$



Gradient descent on this function

- Gradient descent for eight "random" starting points



Going beyond basic gradient descent

- Rather than using a constant learning rate, we can use a **line search** to find the appropriate step size along the current gradient
- The gradient (first-order information) only gives us information about the slope
    - Tells us the slope (direction of steepest ascent).
- We may also be able to use information about the **curvature** (second-order information) of the function to try to speed up the search
- The curvature is given by the **Hessian** matrix: the matrix of second-order partial derivatives
    - **Curvature (Hessian)**: Tells us how the slope itself changes (i.e., how the terrain is curved). Using curvature allows for more efficient and adaptive steps in the optimization process.

Going beyond basic gradient descent

- A classic **second-order** optimisation method based on the inverse of the hessian matrix is the **Newton-Raphson** method
  - Uses both the gradient and the Hessian to take smarter steps in optimization.
- If this is too expensive, it may be possible to approximate the inverse Hessian, yielding a **quasi-Newton** approach
  - Approximates the Hessian to make the method computationally feasible while still using curvature information for faster convergence.
- An optimisation problem may also involve **constraints**

References

- https://uclaacm.github.io/gradient-descent-visualiser/
- CS50's Introduction to Artificial Intelligence with Python 2020 (https://cs50.harvard.edu/ai/2024/)