Robotics & Internet-of-Things | جامعة الأمير سلطان PRINCE SULTAN UNIVERSITY

CS316: INTRODUCTION TO AI AND DATA SCIENCE

CHAPTER 8
CONVEX OPTIMIZATION

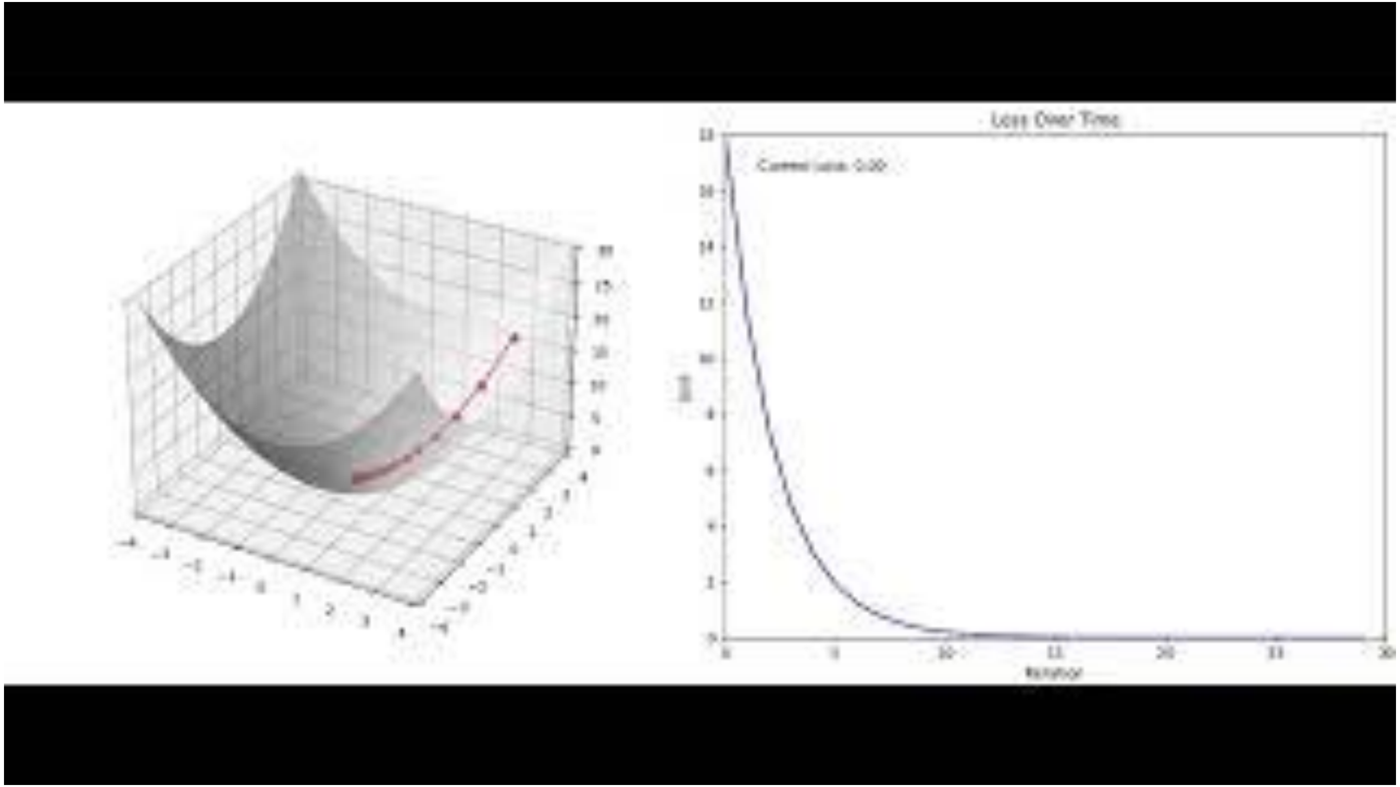
LECTURE
CONVEX OPTIMIZATION

Prof. Anis Koubaa

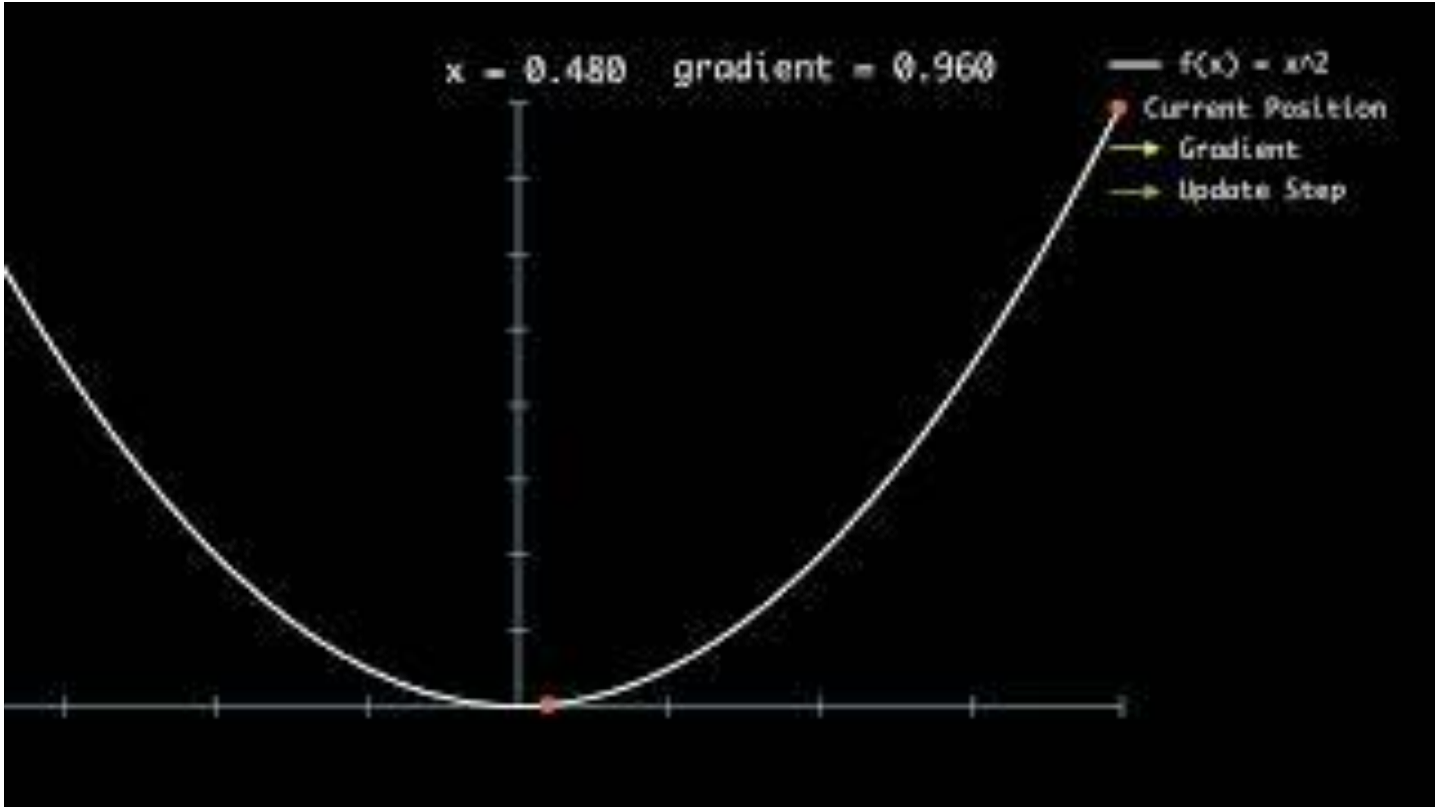
Nov 2024

www.riotu-lab.org

What is Convex Optimization?



What is Gradient Descent?



What is Optimization?

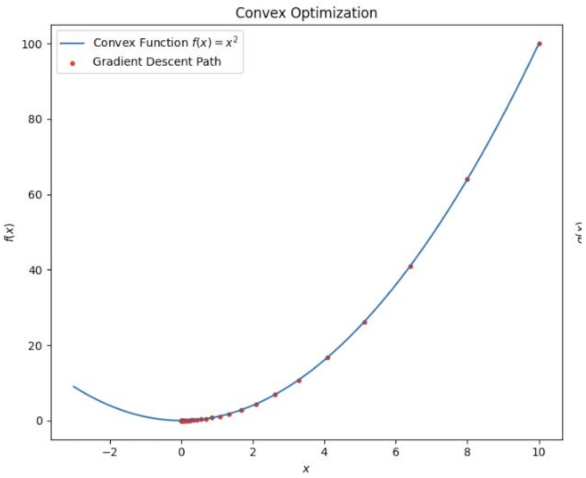
- **Definition of Optimization**
  - Optimization involves **selecting the best element** from a set of available alternatives.
  - In mathematical terms, this process is often associated with **finding the minimum or maximum** of a function.

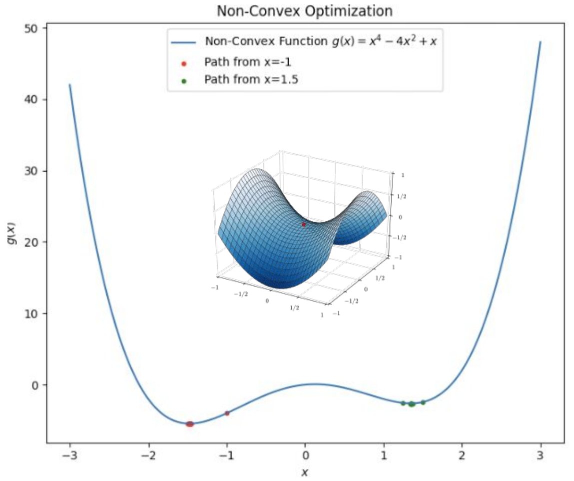
Importance in Data Science

- **Core Component**
  - Optimization is the **backbone of machine learning**.
  - Enables models to **learn from data** by systematically **improving performance** according to specified **metrics (i.e., loss function)**.
- **Objective Function**
  - **Machine learning** models are trained by **minimizing** or **maximizing** an **objective function**, also known as a **loss** or **cost** function.
  - This function measures the **error** or the discrepancy between the ***predicted*** values and the ***actual*** values in the training data.

Types of Optimization Problems

- **CONVEX OPTIMIZATION PROBLEMS**
  - **Definition:** An optimization problem where the **objective function** is a **convex function** and the feasible set is a convex set.
  - **Characteristics:** Unique **global minimum**; any **local minimum** is also a global minimum, simplifying the search for solutions.
  - **Example:** Least squares linear regression, where the function to minimize is a quadratic function of the parameters.
- **NON-CONVEX OPTIMIZATION PROBLEMS**
  - **Definition:** An optimization problem where the **objective function** or the feasible set is **non-convex**.
  - **Characteristics:** Potential for multiple **local minima** and possibly saddle points, making these problems more challenging to solve.
  - **Example: Neural network** training, where the **loss landscape** is **highly non-linear** and contains **many local minima**.

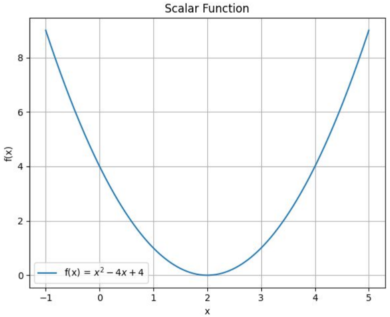




Convex Optimization Problems in Data Science

- **Linear Regression**
  - **Problem: Minimize** the **sum of squared residuals** (MSE) between observed values and values predicted by a linear model.
  - **Nature: Convex problem** as the objective function is a **quadratic function**, ensuring a single global minimum.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$



- **Logistic Regression:**
  - **Problem: Maximize** the **likelihood** of correctly predicting binary outcomes using a logistic function.
  - **Nature:** Convex problem due to the **log-likelihood** function being concave; minimization of its negative is a convex optimization problem.

$$L(y, \hat{p}) = -\left[y \log(\hat{p}) + (1 - y)\log(1 - \hat{p})\right]$$

CHAPTER 3
CONVEX OPTIMIZATION

LECTURE
CONVEX OPTIMIZATION

CONVEX FUNCTIONS

CS313: INTRODUCTION TO DATA SCIENCE

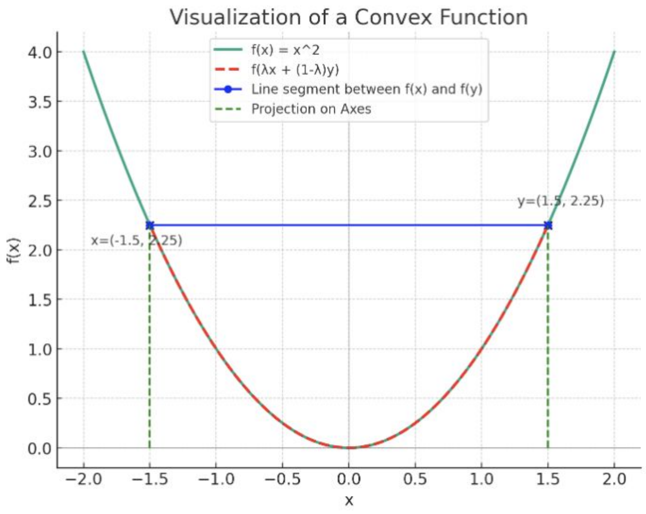
Prof. Anis Koubaa

Definition of Convexity

- **Concept Overview:**
  - **Convex Function:** A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if, for all $x, y \in \mathrm{dom}(f)$, and for any λ in the interval [0, 1],

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$$

- **Geometric Interpretation:**
  - The line segment connecting any two points on the graph of f does not lie below the graph at any point between these two points.
- **Importance in Optimization:**
  - Understanding convexity is critical as it simplifies optimization problems significantly by ensuring that every local minimum is a global minimum.

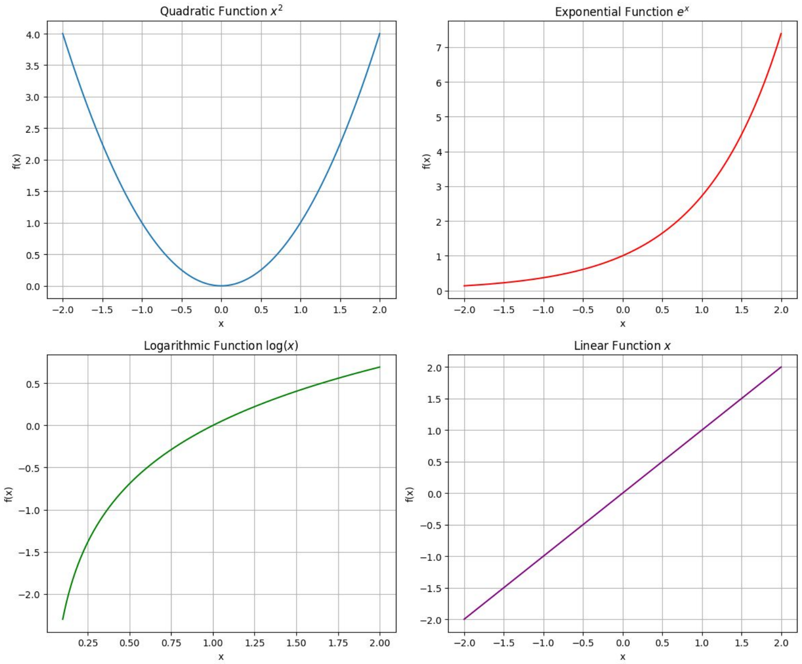


1. The curve $f(x) = x^2$, shown as a solid line, which represents the function over the interval from -2 to 2.
2. The line segment connecting the points $(x, f(x))$ and $(y, f(y))$ for $x = -1.5$ and $y = 1.5$, shown as blue points connected by a line. This line demonstrates the linear combination of $f(x)$ and $f(y)$.
3. The dashed red line, which plots $f(\lambda x + (1-\lambda)y)$ for λ in the interval [0, 1]. This represents the function value at the convex combinations of x and y.

As you can see from the plot, the segment (in blue) lies above the graph of the function $f(x) = x^2$ (in red), illustrating that $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$ for all λ between 0 and 1, which confirms that $f(x) = x^2$ is indeed a convex function. [>-]

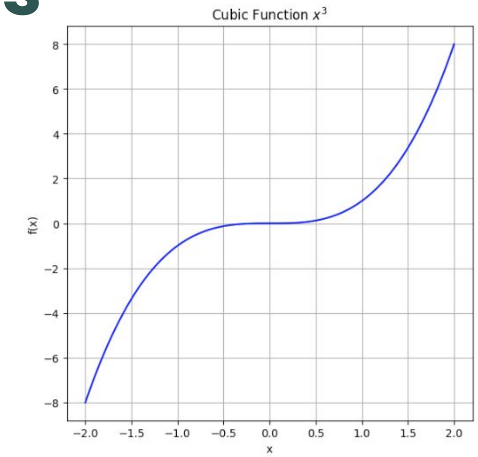
Examples of Convex Functions

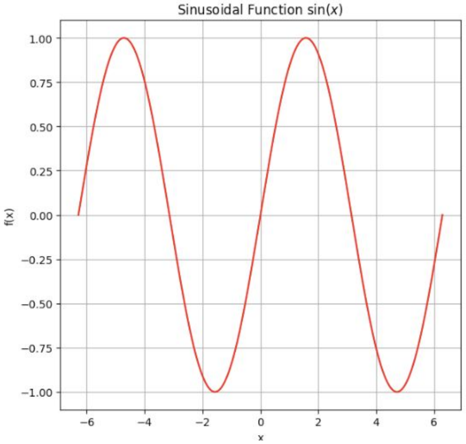
- **Quadratic Functions:**
  - Example: $f(x) = x^2$, which is convex because the second derivative $f''(x) = 2$ is always positive.
- **Exponential Functions:**
  - Example: $f(x) = e^x$, with its second derivative $f''(x) = e^x$ also being positive.
- **Logarithmic and Linear Functions:**
  - Example: $f(x) = \log(x)$ (convex over $x > 0$).
  - Linear Function Example: $f(x) = ax + b$ is convex and concave (it is linear).



Examples of Non-Convex Functions

1. **Cubic Function**: $f(x) = x^3$
   - This function has a point of inflection at $x = 0$, which means it changes curvature from concave to convex, making it non-convex as a whole.
2. **Sinusoidal Function**: $f(x) = \sin(x)$
   - A sinusoidal function oscillates between positive and negative values, with its peaks and troughs making it clearly non-convex, as the line segments connecting points across a peak or trough will lie below the curve.
3. **Absolute Value Function**: $f(x) = |x|$
   - Although it might appear linear and convex at first glance, the absolute value function has a sharp point at $x = 0$, which violates the smoothness condition required for convex functions. It is technically neither convex nor concave due to this cusp.
4. **Polynomial Function with Multiple Roots**: $f(x) = x^4 - x^2$
   - This function, due to its multiple turning points, exhibits both concave and convex intervals, making it non-convex.
5. **Exponential Minus Quadratic**: $f(x) = e^{-x} - x^2$
   - This function has both exponential decay and quadratic growth components, creating multiple inflection points and thus making it non-convex.





CHAPTER 3

CONVEX OPTIMIZATION

LECTURE

CONVEX OPTIMIZATION

Optimization techniques for convex functions

CS313: INTRODUCTION TO DATA SCIENCE

Prof. Anis Koubaa

Unconstrained vs. Constrained Optimization

Definitions & Key Features:

- **Unconstrained Optimization**:
  - **No Limits**: Optimizes a function $f(x)$ anywhere within its domain.
  - **Methods**: Uses simpler methods like Gradient Descent.
- **Constrained Optimization**:
  - **With Conditions**: Must satisfy additional constraints like $g(x) \leq 0$.
  - **Methods**: Requires complex techniques such as Lagrangian Multipliers or KKT Conditions.

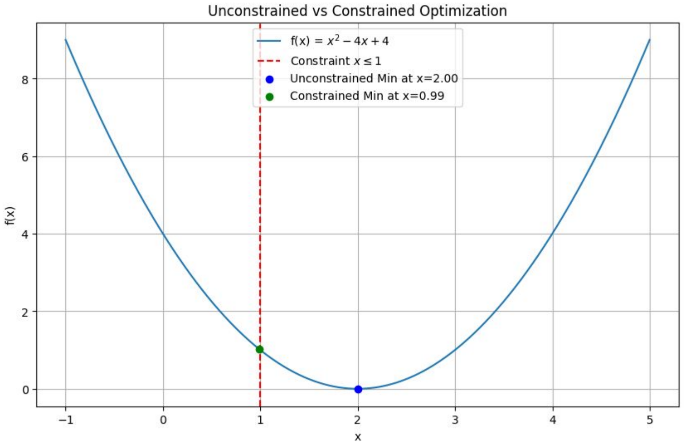
```
# Function to find the minimum using a simple gradient descent approach
def find_minimum_unconstrained(x_start, learning_rate, iterations):
    x = x_start
    for i in range(iterations):
        x -= learning_rate * df(x)
    return x

# Function to find the minimum considering the constraint x <= 1
def find_minimum_constrained(x_start, learning_rate, iterations):
    x = x_start
    for i in range(iterations):
        x_new = x - learning_rate * df(x)
        if g(x_new) <= 0:  # Check if the new x satisfies the constraint
            x = x_new
        else:
            break  # Stop if the constraint is violated
    return x
```

EXAMPLE

Function and Constraints

We'll use the function $f(x) = x^2 - 4x + 4$ for both cases:

1. **Unconstrained Optimization**: We'll find the minimum of the function over its entire domain.
2. **Constrained Optimization**: We'll add a constraint $g(x) = x - 1 \leq 0$, which means we're only allowed to find the minimum where $x \leq 1$.



Unconstrained vs. Constrained Optimization

```python
# Function to find the minimum using a simple gradient descent approach
def find_minimum_unconstrained(x_start, learning_rate, iterations):
    x = x_start
    for i in range(iterations):
        x -= learning_rate * df(x)
    return x

# Function to find the minimum considering the constraint x <= 1
def find_minimum_constrained(x_start, learning_rate, iterations):
    x = x_start
    for i in range(iterations):
        x_new = x - learning_rate * df(x)
        if g(x_new) <= 0:  # Check if the new x satisfies the constraint
            x = x_new
        else:
            break  # Stop if the constraint is violated
    return x
```

Contrast:

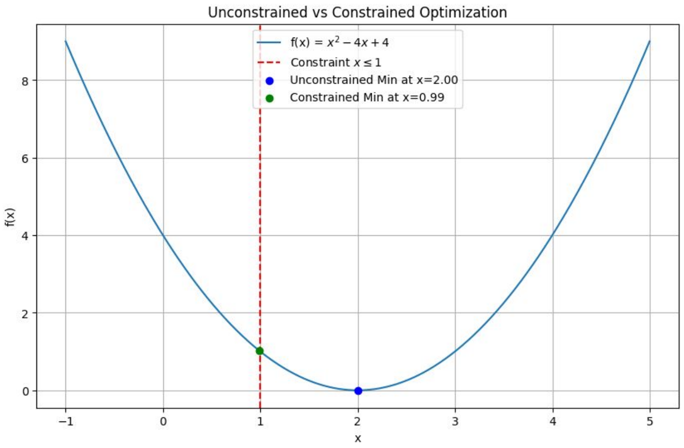
- **Freedom**: Unconstrained has complete freedom in variable choices; Constrained is limited by specific rules.
- **Solution Space**: Unconstrained searches the entire domain; Constrained focuses on the feasible set.

Practical Application:

- **Unconstrained**: Parameter optimization in algorithms.
- **Constrained**: Resource allocation within budget limits.

Conclusion:

- The choice between them depends on the problem constraints and the desired outcomes.



Unconstrained Optimization Techniques

1. **Gradient Descent**:
   - **Mathematical Concept**: Update formula:

     $$x_{\text{new}} = x_{\text{old}} - \alpha \nabla f(x_{\text{old}})$$

     Here, α is the step size, and $\nabla f(x)$ is the gradient or slope of the function at x.
   - **Simple Explanation**: Like walking downhill, this method takes steps proportional to the steepness of the hill to reach the lowest point. The steeper the hill, the bigger the step.

CHAPTER 3
CONVEX OPTIMIZATION

LECTURE
CONVEX OPTIMIZATION

Gradient Descent

CS313: INTRODUCTION TO DATA SCIENCE

Prof. Anis Koubaa

Intuition behind gradient descent

What is Gradient Descent?

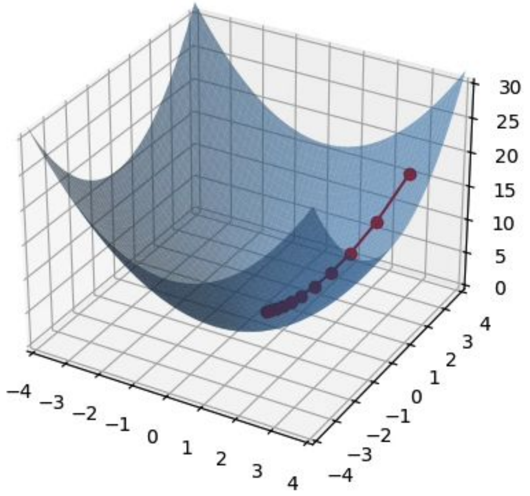
- **Gradient Descent** is an optimization algorithm used to minimize a function by iteratively moving towards the minimum value of the function.

How Does It Work?

- **Step-by-Step Process**:
  1. **Start** with an initial guess for the value of the parameter(s).
  2. **Calculate the Gradient**: Determine the gradient (the slope of the function) at the current point.
  3. **Update the Parameter(s)**: Adjust the parameter(s) in the direction opposite to the gradient to move towards the minimum.

$$x_{\text{new}} = x_{\text{old}} - \alpha \nabla f(x_{\text{old}})$$

Where α is the learning rate, controlling the step size.



Unconstrained vs. Constrained Optimization

Definition & Mechanism:

- **Gradient Descent**: Minimizes a function by updating variables in the direction opposite to the gradient.

$$x_{\text{new}} = x_{\text{old}} - \alpha \nabla f(x_{\text{old}})$$

Key Features:

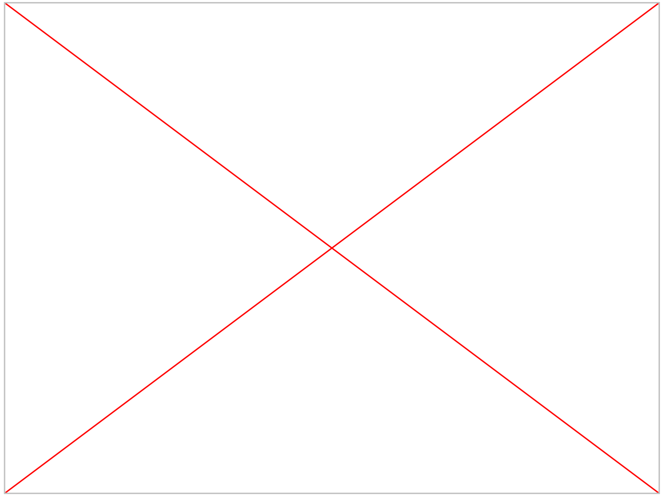
- **Simplicity**: Easy to implement; requires only the gradient computation.
- **Efficiency**: Directly targets the steepest path to reduce the function value.

Ideal for:

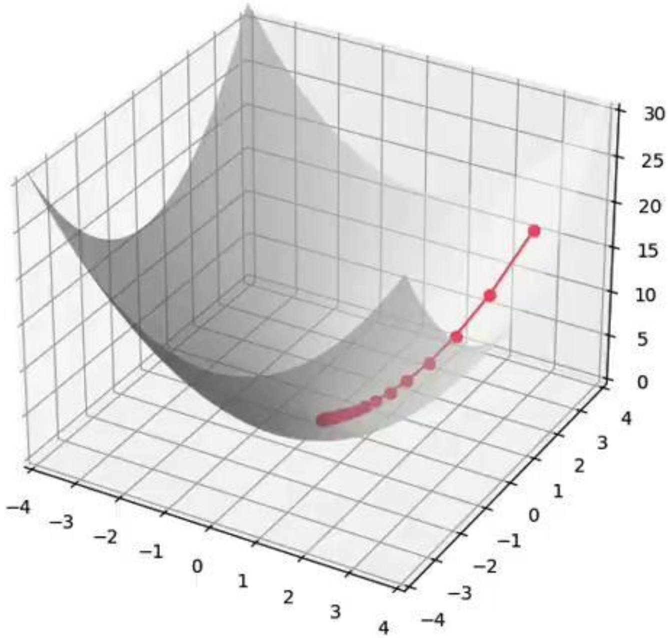
- **Unconstrained Scenarios**: No external conditions affect the descent process.

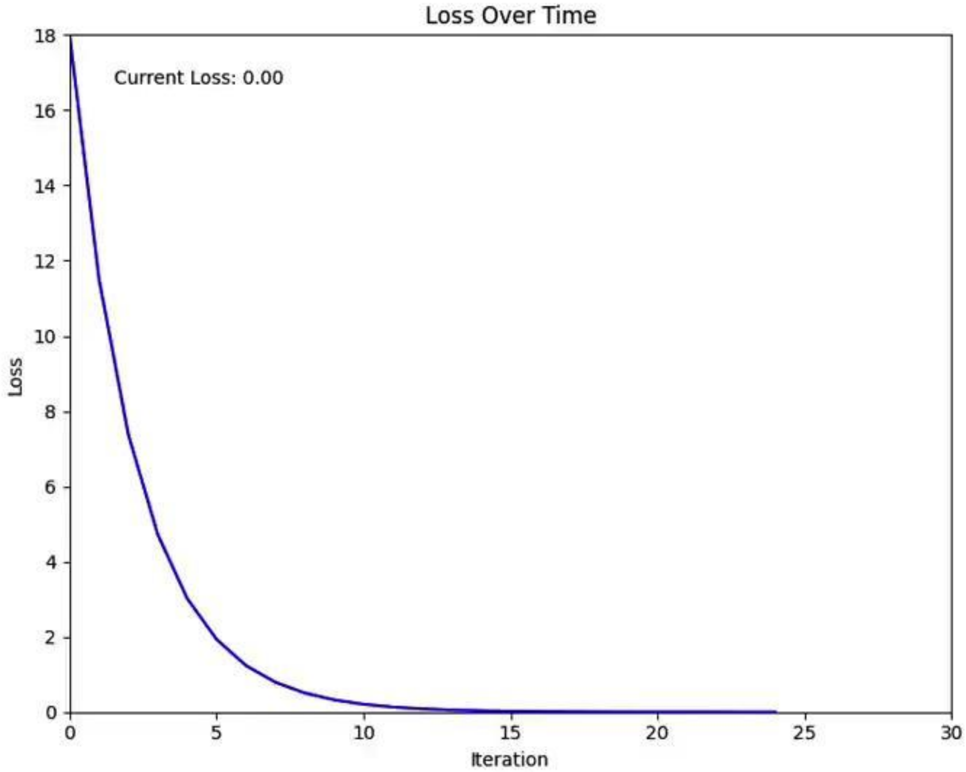
Applications:

- **Machine Learning**: Training models by minimizing error functions.
- **Economic Modeling**: Finding cost-effective strategies.

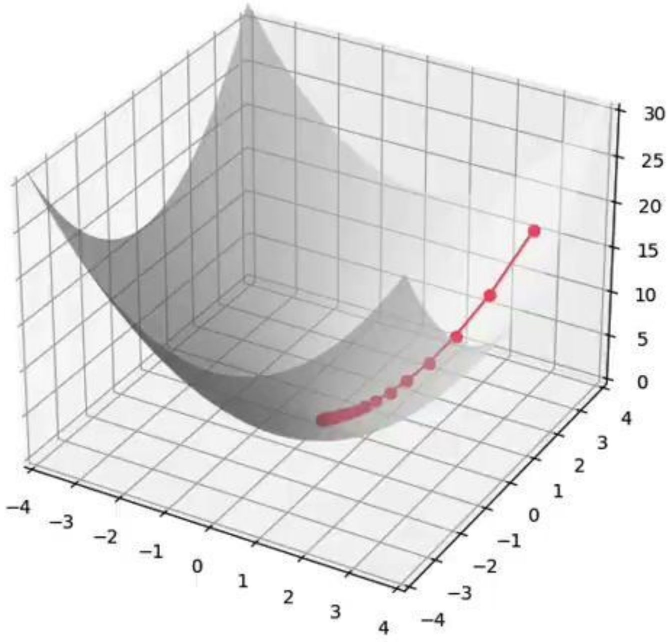


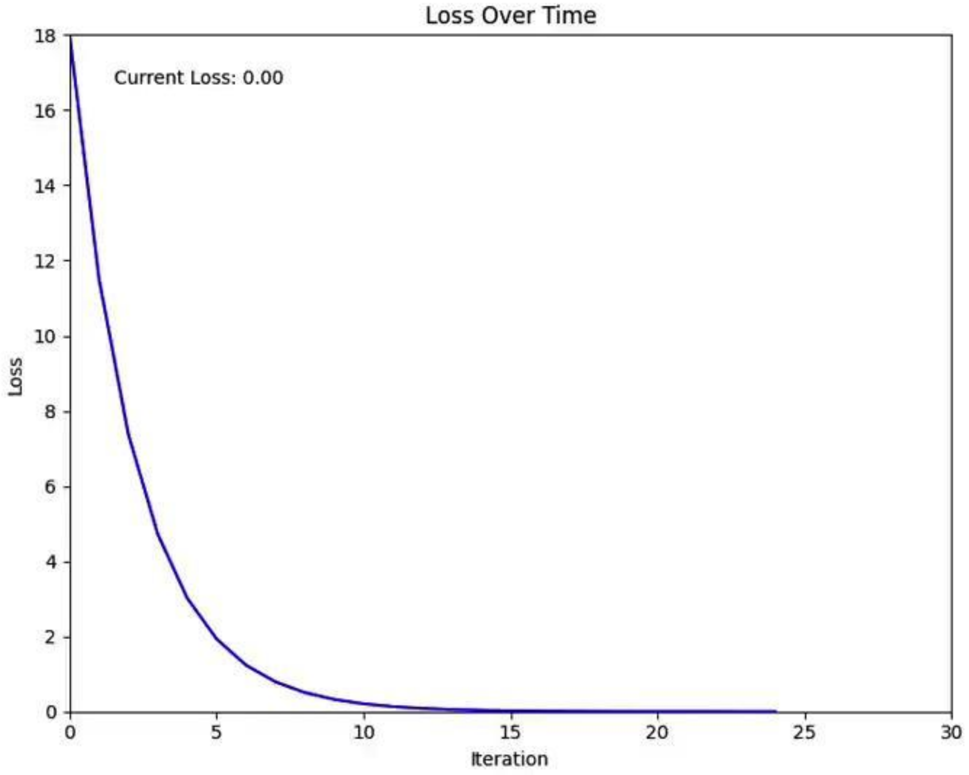
Unconstrained vs. Constrained Optimization





Unconstrained vs. Constrained Optimization





Mathematical Foundation of Gradient Descent

Objective:

- To minimize a function $f(x)$, where x can be a vector of parameters.

Derivation of the Update Rule:

1. **Taylor Expansion**:
   - To understand how the function f changes, we consider the Taylor expansion around a point x:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T H \Delta x$$

   Here, $\nabla f(x)$ is the gradient of f at x, and H is the Hessian matrix of second derivatives.

2. **Neglect Higher-Order Terms**:
   - For small changes Δx, the higher-order terms (like the Hessian term) become negligible, simplifying to:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x$$

Mathematical Foundation of Gradient Descent

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \quad \text{for all } x$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots \quad \text{for all } x$$

$$\tan x = \sum_{n=1}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)}{(2n)!} x^{2n-1} \quad = x + \frac{x^3}{3} + \frac{2x^5}{15} + \cdots \quad \text{for } |x| < \frac{\pi}{2}$$

$$\sec x = \sum_{n=0}^{\infty} \frac{(-1)^n E_{2n}}{(2n)!} x^{2n} \quad = 1 + \frac{x^2}{2} + \frac{5x^4}{24} + \cdots \quad \text{for } |x| < \frac{\pi}{2}$$

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \quad = x + \frac{x^3}{6} + \frac{3x^5}{40} + \cdots \quad \text{for } |x| \le 1$$

$$\arccos x = \frac{\pi}{2} - \arcsin x$$

$$= \frac{\pi}{2} - \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \quad = \frac{\pi}{2} - x - \frac{x^3}{6} - \frac{3x^5}{40} - \cdots \quad \text{for } |x| \le 1$$

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} \quad = x - \frac{x^3}{3} + \frac{x^5}{5} - \cdots \quad \text{for } |x| \le 1,\ x \ne \pm i$$

Mathematical Foundation of Gradient Descent

3. **Descent Direction**:
   - To decrease f, we choose Δx such that $f(x + \Delta x)$ is less than $f(x)$. The most effective direction to decrease f is opposite to the gradient, $\nabla f(x)$:

$$\Delta x = -\eta \nabla f(x)$$

   where η (eta) is a small positive scalar known as the learning rate.

4. **Update Rule**:
   - Substituting Δx in the simplified Taylor expansion:

$$f(x - \eta \nabla f(x)) \approx f(x) - \eta \nabla f(x)^T \nabla f(x)$$

   - Since $\nabla f(x)^T \nabla f(x)$ is always non-negative (it's the square of the gradient norm), the function value decreases if η is chosen properly.

5. **Gradient Descent Formula**:
   - The update rule for x to minimize f becomes:

$$x_{\text{new}} = x - \eta \nabla f(x)$$

   - Each iteration moves x in the direction that most reduces f.

Gradient Descent Algorithm

Algorithm 3 Gradient Descent

```
Input: Loss function f, gradient ∇f, initial weights w_init, learning rate α, tolerance tol, maximum iterations max_iters
Output: Optimized weights w
Initialize: w ← w_init
Initialize: iter ← 0
Initialize: converged ← False {Begin the optimization process}
while not converged and iter < max_iters do
    gradient ← ∇f(w) {Compute the gradient of the loss function with respect to weights}
    w ← w − α × gradient {Adjust weights to minimize the loss, moving against the gradient}
    if ||gradient|| < tol then
        {Check if the gradient is small enough to assume convergence}converged ← True
    end if
    iter ← iter + 1 {Update iteration counter}
end while
return w {Return the optimized weights} =0
```