Numerical Optimization

Numerical optimization is a field of applied mathematics concerned with finding the best possible solution (minimum or maximum) to a problem mathematically represented by an objective function.

key concepts:

- **Objective Function:** This function represents the quantity you want to optimize, either minimize (cost, error) or maximize (profit, efficiency).
- **Decision Variables:** These are the adjustable factors that influence the objective function. They are typically denoted by x (sometimes x1, x2, ..., xn for multiple variables).
- **Types of optimization problems:** There are different categories of optimization problems depending on the properties of the objective function and constraints (linear, nonlinear, convex, etc.).
- **Optimization algorithms:** Various algorithms exist for solving different types of optimization problems, each with its strengths and weaknesses (gradient descent, simplex method, etc.).

Objective

reaching the minimum error with the fewest iterations is a crucial goal in numerical optimization, especially for problems involving Deep Learning (DL) and Machine Learning (ML).

Importance of Minimizing Iterations:

- **Computational Efficiency:** Each iteration in an optimization algorithm requires computations. Reducing iterations translates to faster training times for your DL/ML models, saving time and resources.
- **Convergence Issues:** Sometimes, algorithms can get stuck in local minima (not the absolute minimum) if they run for too long. Fewer iterations can help mitigate this risk.

- **Practical Considerations:** In real-world applications, especially those involving resource constraints, minimizing training time is often essential.

steps in most numerical optimization algorithms:

- **Initialization:**
  - This is the starting point where you define the initial guess for the decision variables. This guess can be random, based on prior knowledge of the problem, or obtained from simpler calculations.
  - The quality of the initial guess can significantly impact the efficiency of the algorithm. A good guess can lead to faster convergence to the minimum.
- **Evaluation:**
  - You evaluate an objective function that represents the quantity you want to minimize (or maximize). This function takes the current values of the decision variables as input and outputs a single value representing the "goodness" of the current solution.
- **Direction of Update:**
  - Based on the current evaluation and possibly the history of previous evaluations, the algorithm determines a direction in which to move the decision variables. This direction should ideally lead towards a lower value of the objective function.
  - Different algorithms use different strategies to determine this direction. Some common approaches include using the gradient (the direction of steepest ascent/descent) or conjugate directions.
- **How to Update:**
  - Once the direction is determined, the algorithm decides by how much to move the decision variables along that direction. This step involves a concept called the "step size". Choosing an appropriate step size is crucial. A small step size might lead to slow convergence, while a large step size might jump past the minimum or even cause the algorithm to diverge.

- Many algorithms use line search techniques to find the optimal step size along the chosen direction.

Numerical optimization algorithms can be classified into two main categories based on the derivatives they use: first-order and second-order methods.

1. First-Order Methods:

- These algorithms rely only on the gradient of the objective function, which indicates the direction of steepest ascent/descent.
- They are simpler to implement and less computationally expensive compared to second-order methods.
- Common examples include:
  - Gradient Descent and its variants (Stochastic Gradient Descent, Momentum, Adam)
  - Coordinate Descent methods

2. Second-Order Methods:

- These algorithms utilize both the gradient and the Hessian of the objective function. The Hessian provides information about the curvature of the function, allowing for faster convergence to the minimum compared to first-order methods.
- However, they can be more complex to implement and computationally expensive due to the need to calculate the Hessian, which can be challenging for high-dimensional problems.
- Examples include:
  - Newton's Method
  - Quasi-Newton Methods (BFGS, L-BFGS)