Nature Inspired Computation
DSAI 403

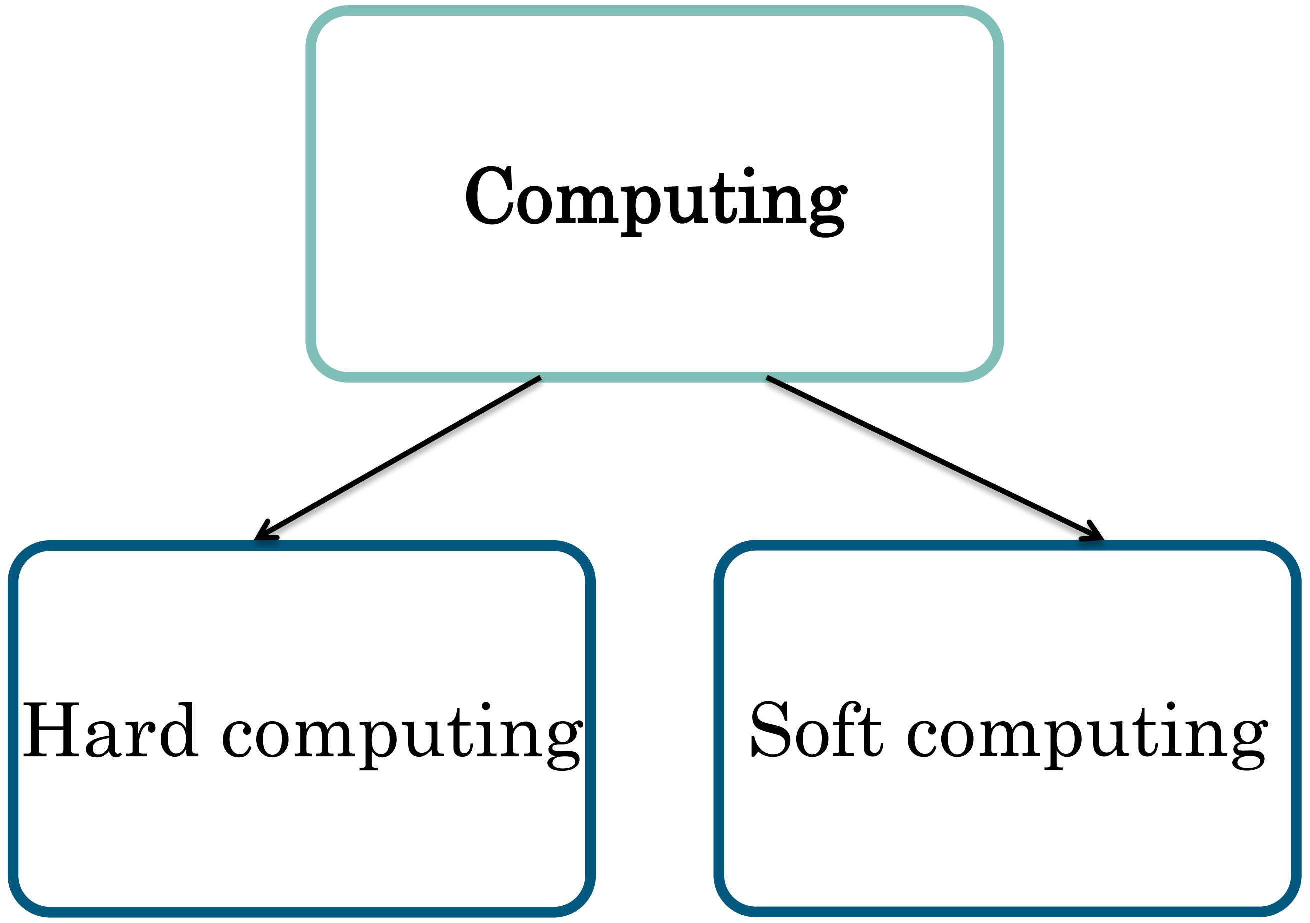
Assoc. Prof. **Mohamed Maher Ata**
Zewail city of science, technology, and innovation
momaher@zewailcity.edu.eg
Room: S028- Zone D

Computing

The two major problem-solving technologies include:

Hard computing:

- It is suitable for the problems, which are easy to model mathematically and whose stability is highly predictable.
- It relies on the principles of accuracy, certainty, and inflexibility.
- It deals with precise models where accurate solutions are achieved quickly.
- Problem: As most of the real-world problems are complex in nature and difficult to model mathematically
- Examples: Arithmetic operations: calculating (2 + 2 = 4), Shortest path algorithms (like Dijkstra's algorithm in graph theory), Sorting algorithms: bubble sort, quicksort.



Soft computing:

- The main goal of soft computing is to provide solutions to complex real-world problems which are not modeled or too difficult to model mathematically. It does not require an extensive mathematical formulation of the problem like hard computing.
- Examples: Weather prediction, **neural networks** learn patterns from imperfect images

Soft computing

- It may not be able to yield so much precise solution as that obtained by the hard computing.
- Different members of this family are able to perform various types of tasks:
  - Fuzzy Logic (FL) is a powerful tool for dealing with imprecision and uncertainty.
  - Artificial Neural Network (ANN) is a potential tool for learning and adaptation.
  - Genetic Algorithm(GA) is an important tool for search and optimization.

Hard computing vs. soft computing

	Hard computing	Soft computing
Model	It deals with precise models	It deals with approximate models
Methodologies	Binary logic and crisp systems	Fuzzy logic and probabilistic reasoning
Features	Accuracy, certainty and inflexibility	Approximation, uncertainty and flexibility
Nature	Deterministic	Stochastic
Data	Exact input data	Ambiguous and noisy data
Computation	Sequential computation	Can perform parallel computation

Summary:

1. **Hard Computing** is like a **calculator** → always gives exact answers (2 + 2 = 4).
2. **Soft Computing** is like a **human brain** → can make decisions even with incomplete information (e.g., recognizing a friend's face in a blurry photo).

Hybrid computing

- It is a **combination** of the conventional hard computing and emerging soft computing, such that a part of the complex real-world will be solved using hard computing and the remaining part can be tackled using soft computing to get advantages from both of them and eliminate their individual limitations.

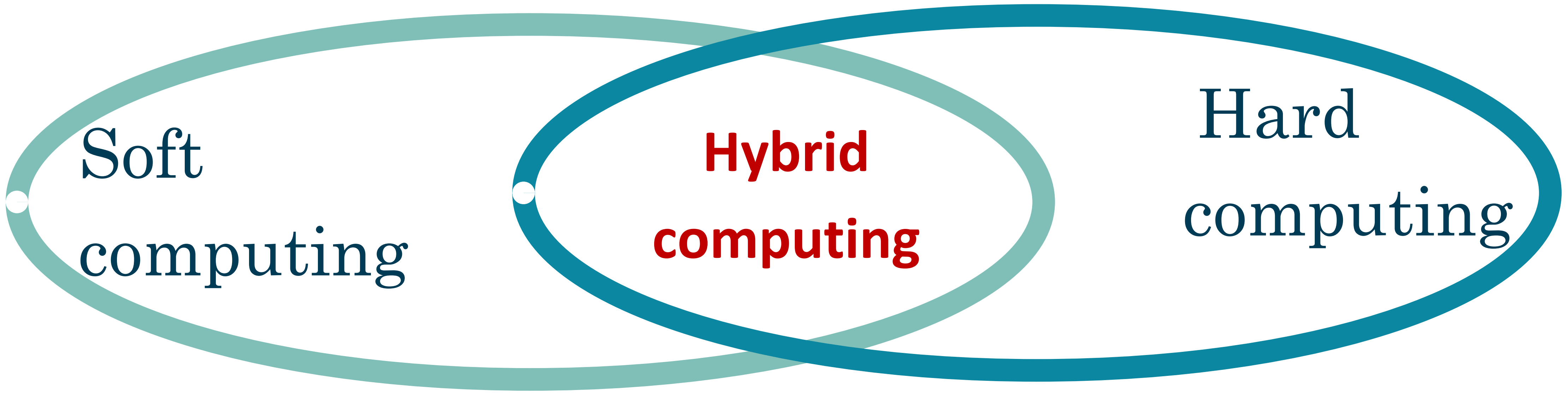
Examples:

1. **Medical Diagnosis Systems**

Hard computing: exact calculations of lab test values.
Soft computing: fuzzy/neural networks interpret vague symptoms (e.g., “feels weak,” “moderate fever”).
Together = more reliable diagnosis.

2. **Self-Driving Cars**

Hard computing: sensor fusion, exact distance calculations, control system equations.
Soft computing: AI decision-making, fuzzy rules for uncertain traffic conditions.



Before formal optimization techniques: reasoning

reasoning by feasibility (constraint-based reasoning) of a small assignment problem:

Assume that you have 3 pretrained models and 3 edge devices. Each device must run exactly **one model**. Our goal is to minimize total validation error across all devices while respecting device memory limits.

Assume that:

Models (size, validation error):

- M1: 30 MB, error = 6.0%
- M2: 90 MB, error = 4.0%
- M3: 150 MB, error = 3.5%

Devices (memory):

- Device A: 32 MB
- Device B: 128 MB
- Device C: 256 MB

Important rule: a model **cannot** be assigned to a device **if model size > device memory**.

Solution:

Step 1: Construct cost matrix:

	A (32MB)	**B (128MB)**	**C (256MB)**
M1 (30MB)	6	6	6
M2 (90MB)	infeasible	4	4
M3 (150MB)	infeasible	infeasible	3.5

Step 2: our objective is to minimize the sum of the three devices' validation errors.

Total error = error(A) + error(B) + error(C)

- Because M3 only fits on C, M3 must go to Device C.
- Because M2 cannot go to A, M2 must go to B. as C has been assigned before to model M3
- Therefore Device A must take M1

Total validation error = **6.0 + 4.0 + 3.5 = 13.5%**

The problems of reasoning

Scalability

- Reasoning works when the problem is very small (few variables).
- But as soon as the problem grows (e.g., 100 tasks, 50 machines), there are **too many possibilities** for the human brain to check.

No guarantee of optimality

- Human reasoning might find a “good-looking” solution, but you can’t be sure it’s the **best** one.

Complex constraints

- Real problems have many constraints (capacity, time, energy, costs).
- Hard to consider all of them just by hand.

Time

- Even if you could reason it out, it would take **too long** for large-scale problems

Before formal optimization techniques: Newton Raphson

- Find the roots for the following equation: $f(x) = x^3 + x - 9$
- Derivative: $f'(x) = 3x^2 + 1$
- Intermediate Theorem: $f(1) = 1^3 + 1 - 9 = -7$ $\quad f(2) = 2^3 + 2 - 9 = 1$
- Accordingly, roots lies in between 1 & 2.
- Initialize at $x_0 = 1$

$$\begin{aligned} x_0 &= 1 \\ f(x_0) &= 1^3 + 1 - 9 = -7 \\ f'(x_0) &= 3 * 1^2 + 1 = 4 \\ x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{-7}{4} = 2.75 \end{aligned}$$

$$\begin{aligned} x_1 &= 2.75 \\ f(x_1) &= 2.75^3 + 2.75 - 9 = 14.54 \\ f'(x_1) &= 3 * 2.75^2 + 1 = 23.6875 \\ x_2 &= x_1 - \frac{f(x_0)}{f'(x_0)} = 2.75 - \frac{14.54}{23.6875} = 2.136 \end{aligned}$$

$$\begin{aligned} x_2 &= 2.136 \\ f(x_2) &= 2.136^3 + 2.136 - 9 = 6.28 \\ f'(x_2) &= 3 * 2.136^2 + 1 = 14.6875 \\ x_3 &= x_2 - \frac{f(x_0)}{f'(x_0)} = 2.136 - \frac{6.28}{14.6875} = 1.71 \end{aligned}$$

$$\begin{aligned} x_3 &= 1.71 \\ f(x_3) &= 1.71^3 + 1.71 - 9 = -2.29 \\ f'(x_3) &= 3 * 1.71^2 + 1 = 9.7723 \\ x_4 &= x_3 - \frac{f(x_3)}{f'(x_3)} = 1.71 - \frac{-2.29}{9.7723} = 1.944 \end{aligned}$$

$$\begin{aligned} x_4 &= 1.944 \\ f(x_4) &= 1.94^3 + 1.94 - 9 = 0.24 \\ f'(x_4) &= 3 * 1.94^2 + 1 = 12.3 \\ x_5 &= x_4 - \frac{f(x_4)}{f'(x_4)} = 1.944 - \frac{0.24}{12.3} = 1.9245 \end{aligned}$$

$$\begin{aligned} x_5 &= 1.9245 \\ f(x_5) &= 1.9245^3 + 1.9245 - 9 = 0.05 \\ f'(x_5) &= 3 * 1.9245^2 + 1 = 12.111 \\ x_6 &= x_5 - \frac{f(x_5)}{f'(x_5)} = 1.9245 - \frac{0.05}{12.11} = 1.920 \end{aligned}$$

- **So, final root of the equation is approximately: 1.920**
- **This method is gradient dependent**
- **Take very long time**

Optimization

- Optimization is the process of finding the best solution out of all feasible solutions.
- There exists no specific method which solves effectively all optimization problems (**No free launch (NFL) theorems**).
- **NFL** states that " **no single algorithm is the best for all problems** ". Accordingly, no optimization algorithm is universally better than others; averaged over all possible problems, every algorithm performs the same.

This means:

- If an algorithm performs well on one class of problems, It must perform worse on another class to compensate. Any advantage an algorithm shows on some problems will be exactly balanced by worse performance on other problems.
- This tells us that when solving real problems, we must choose or design the algorithm to fit the specific problem’s structure, instead of expecting one universal method to always work best.

Example

Assume we have two models

- Model A: a decision tree.
- Model B: a neural network.

Assume we have two different datasets

- Dataset 1: A simple, rule-based dataset (e.g., “If it’s raining, carry an umbrella”). This will be easy for a decision tree algorithm.
- Dataset 2: A complex, nonlinear dataset (e.g., recognizing handwritten digits). This will be easier for a neural network algorithm.

Step 1: Model performance (we have 4 cases)

On Dataset 1:

Decision tree (Model A) → performs very well
Neural network (Model B) → performs poorly

On Dataset 2:

Decision tree (Model A) → performs poorly
Neural network (Model B) → performs very well

Step 2: Average over all possible problems

If we had to average performance across *all possible datasets* (from the simplest rules to the most complex patterns), the overall score of both models would be the same.

Conclusion:

The No Free Lunch theorem in AI means:

- *No single model (deep learning, SVM, decision tree, etc.) is always the best for every dataset.*
- Each algorithm wins on some problems but loses on others.
- That’s why we choose models based on the problem’s structure (e.g., CNNs for images, RNNs for sequences, trees for tabular data).

Formal Description

Find x^* such that: $$x^* = \arg\min_{x \in S} f(x)$$

Where:

- x: decision variable (vector in $\mathbb{R}^n$)
- f(x): cost, objective, or fitness function
- S: feasible set (may include constraints)

For Example:

$f(x) = (x-3)^2, x \in S$

- Minimum value of f(x) is 0
- The point where this happens $x = 3$

Accordingly,

$\min f(x) = 0 \quad and \arg\min f(x) = 3$

Note that:

Optimization ≠ Always Minimization:

- Minimize cost or loss
- Maximize performance (accuracy, profit)

$$x^* = \arg\max_{x \in S} f(x)$$

Types of Optimization Problems

Linear vs. Nonlinear Optimization

Linear Optimization

- Objective & constraints are linear.
- Solved efficiently (e.g., Simplex method).

Example:

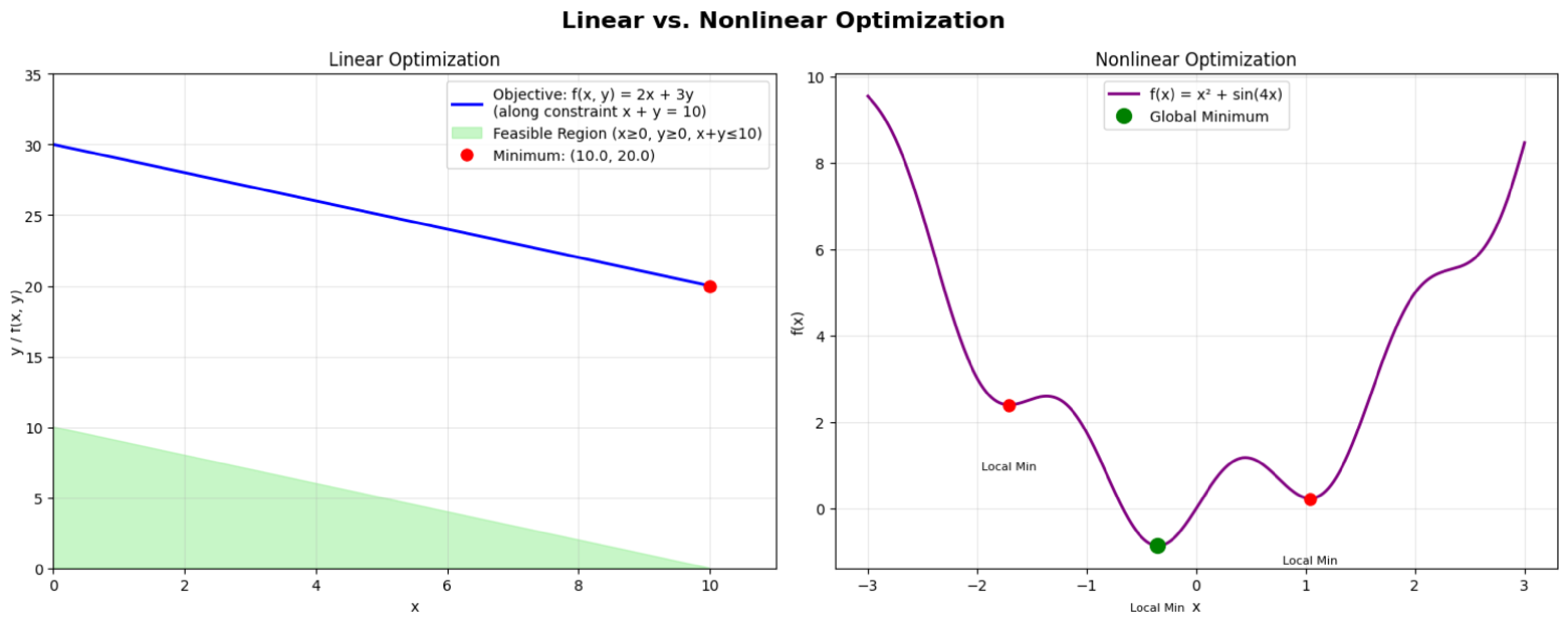
$$min\ f(x, y) = 2x + 3y\ \ s.t.\ x + y \leq 10$$

Nonlinear Optimization

- Objective or constraints are nonlinear.
- May have multiple local minima.

Example:

$$min\ f(x) = x^2 + \sin(x)$$



Types of Optimization Problems

Continuous vs. Discrete Optimization

Continuous **Optimization**

- Variables take values in continuous range.

Example:

$$min\ f(x) = x^2,\ x \in R$$

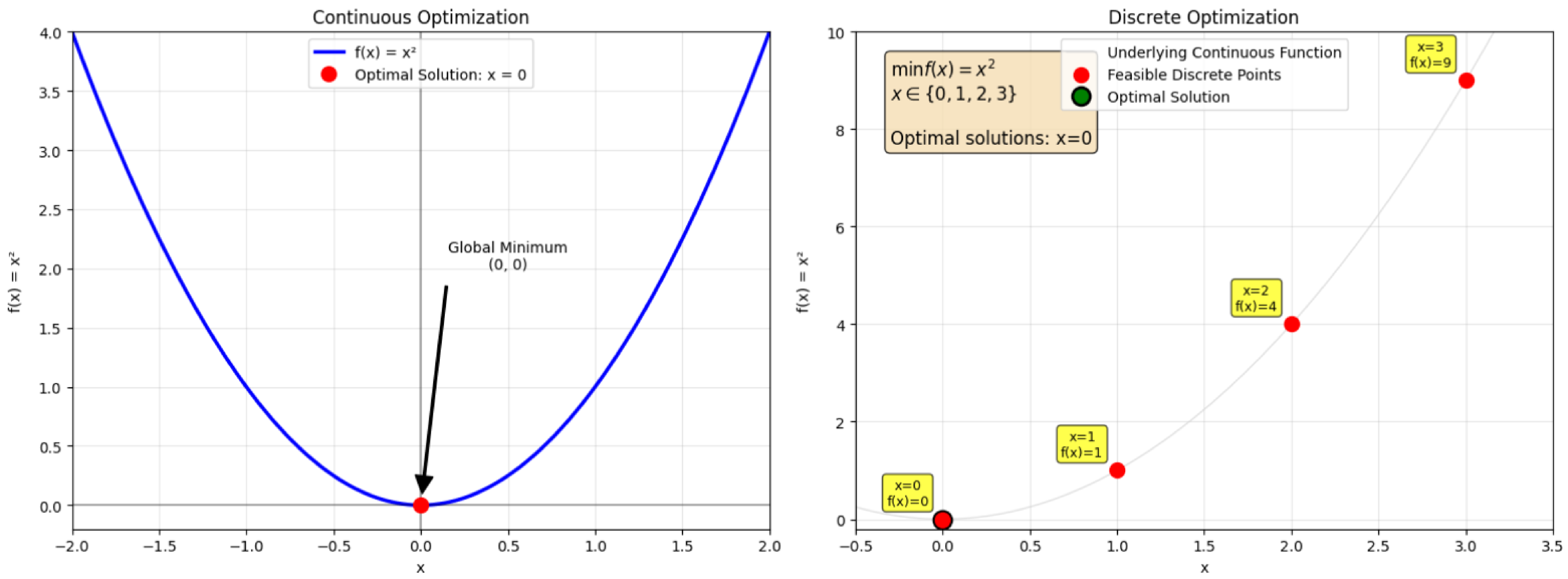
Discrete **Optimization**

- Variables restricted to integers or finite sets.

Example:

$$min\ f(x) = x^2,\ x \in \{0,1,2,3\}$$

Continuous vs. Discrete Optimization



Types of Optimization Problems

Constrained vs. Unconstrained Optimization

Constrained **Optimization**

- Variables must satisfy conditions.

Example:

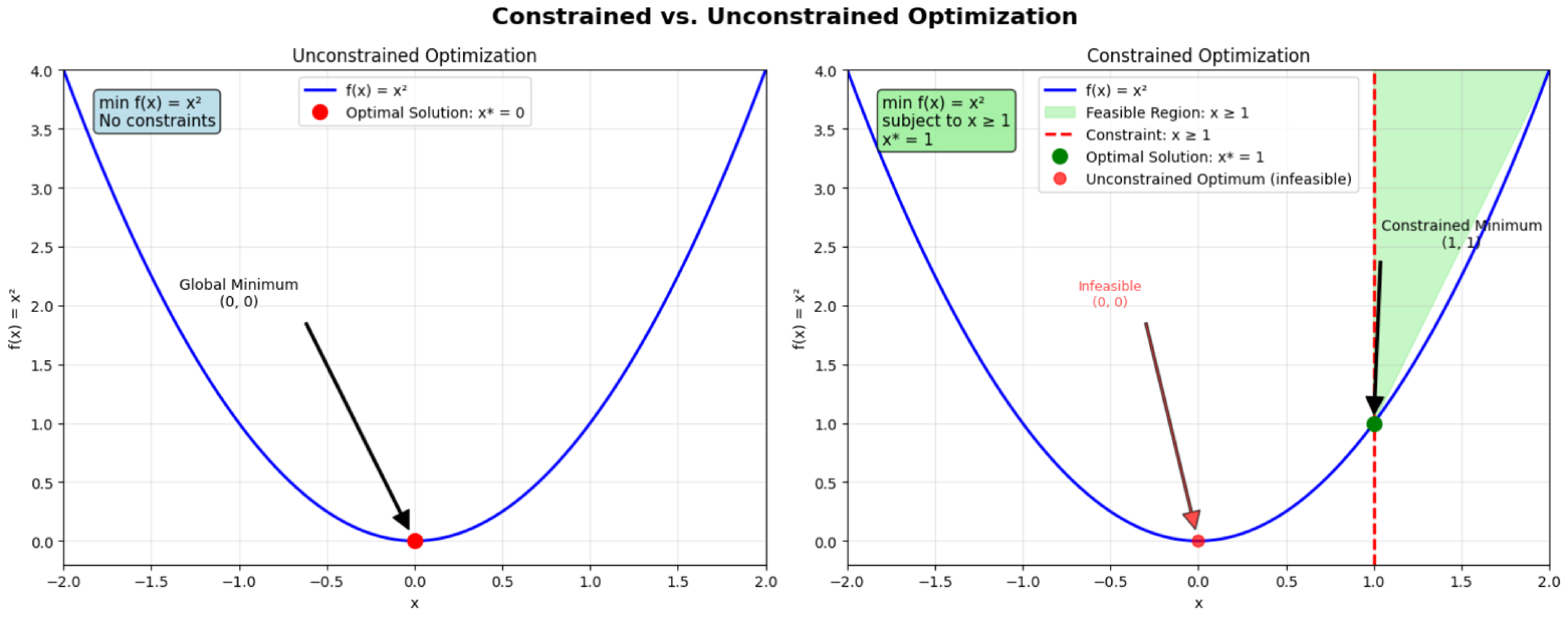
$$min\ f(x) = x^2,\ \mathbf{s.t.}\ x \geq 1 \Rightarrow x^* = 1.$$

Unconstrained **Optimization**

- No restrictions on variables.

Example:

$$min\ f(x) = x^2 \Rightarrow x^* = 0.$$



Types of Optimization Problems

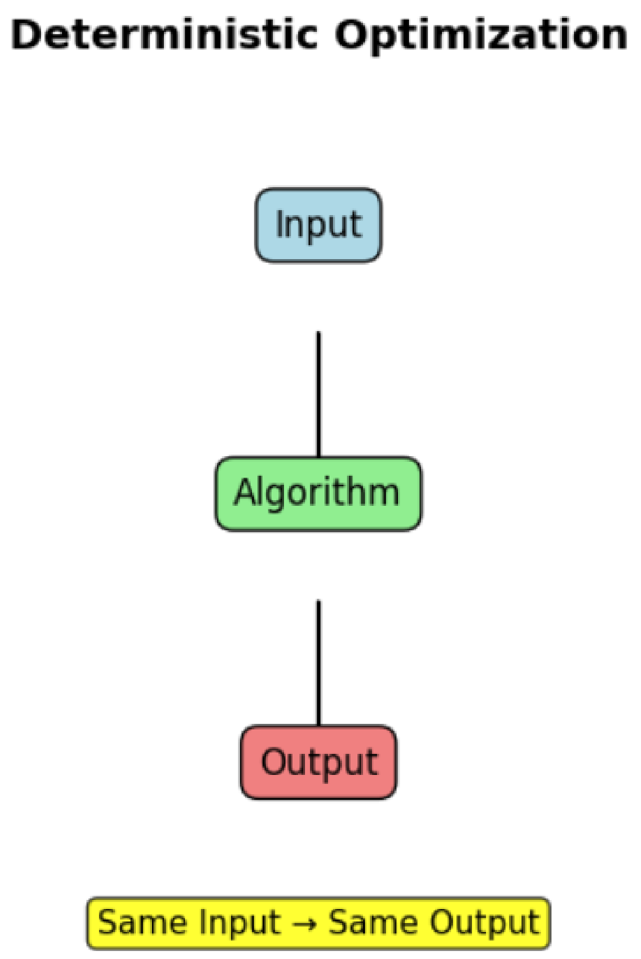
Deterministic vs. Stochastic Optimization

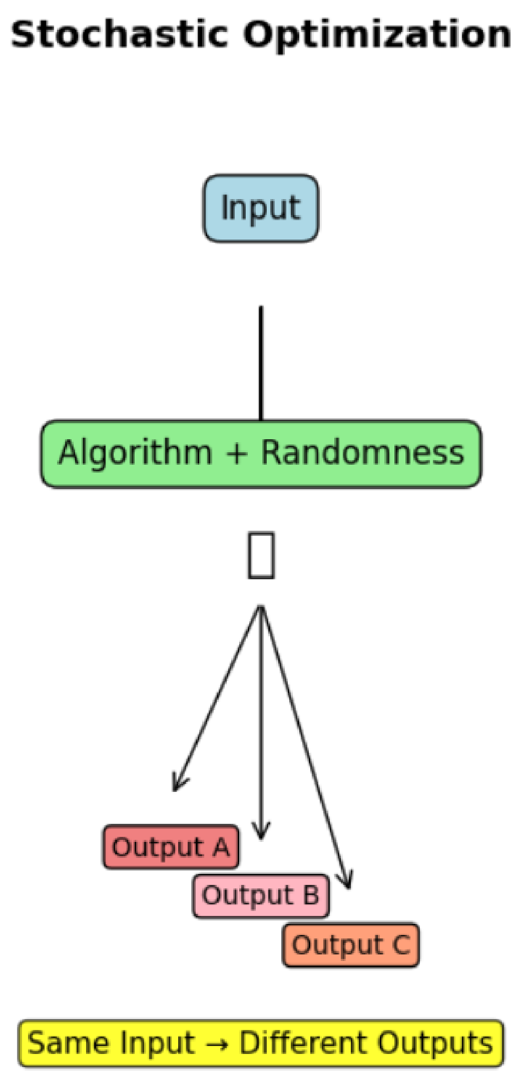
Deterministic

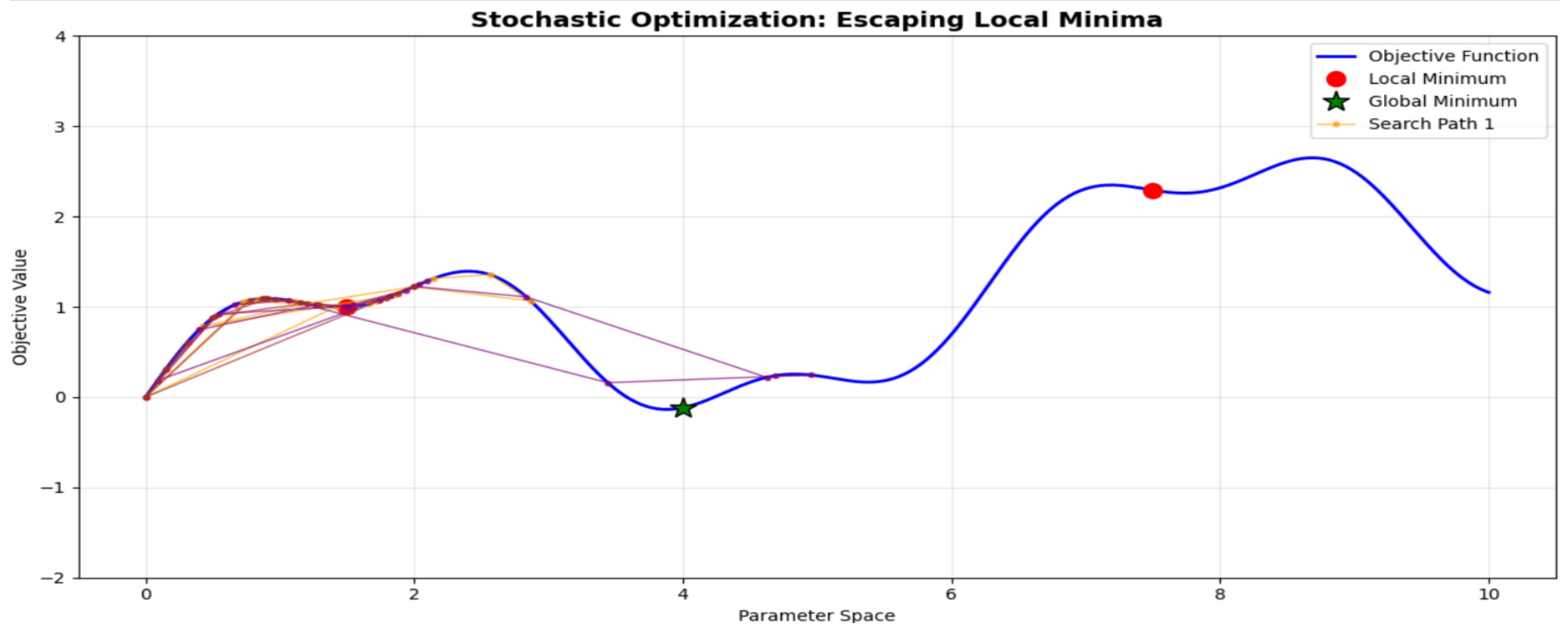
- Same input → same solution.
- Predictable, reproducible.
- Example: Linear Programming.

Stochastic

- Incorporates randomness.
- Solutions may vary between runs.
- Useful for noisy/complex spaces.
- Example: Genetic Algorithm







Optimization methods

1. Exact method

- Advantage: it can guarantee to find the accurate and optimal solution, such as dynamic programming, branch-and-bound, and integer linear programming.
- Disadvantage: in the worst case, if the problem size is increased, the time complexity would be increased exponentially.
- It is suitable for small- scale optimization problems.

2. Approximate method (our target in the course)

- It can not guarantee to find the accurate solution.
- It obtains approximate solutions in an acceptable and reasonable time.
- It is suitable for large-scale optimization problems.

Heuristic and Metaheuristic (As an approximate methods)

	Heuristic	**Metaheuristic**
Scope	Problem-dependent	Problem-independent (general framework)
Flexibility	Works well for one domain	Can be adapted to many domains
Optimality	Often finds *good* solutions, not guaranteed optimal	Aims for near-optimal, no guarantee
Solution type	Approximate	Approximate
Examples	Nearest neighbor, greedy methods, rule-based methods	GA, PSO, ACO, SA, Tabu Search
Speed	Fast	Moderate (but scalable)
Inspiration	Human intuition, simple rules	Nature-inspired, physics-inspired, or strategy-based

Families of Metaheuristic Algorithms

By Define the family of metaheuristic we need to :

- Compare major metaheuristic algorithms
- Understand how they work
- Know when to use each algorithm

Metaheuristics can be broadly classified into 3 major families:

Family	Inspiration Source	Examples	Key Features
Evolutionary-based	Biological evolution & natural selection	Genetic Algorithm (GA), Evolution Strategies (ES), Differential Evolution (DE)	- Uses crossover & mutation operators - Population-based - Good at exploration
Swarm-based	Collective behavior of animals/insects	Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Firefly Algorithm	- Agents cooperate & share information - Mimics real-world swarm intelligence - Balances exploration & exploitation
Physics/Chemistry-inspired	Physical & chemical processes	Simulated Annealing (SA), Gravitational Search Algorithm (GSA), Harmony Search, Chemical Reaction Optimization	- Inspired by physical/chemical laws - Uses probabilistic moves - Often simple but powerful for escaping local minima

Why Metaheuristic Instead of Gradient Descent?

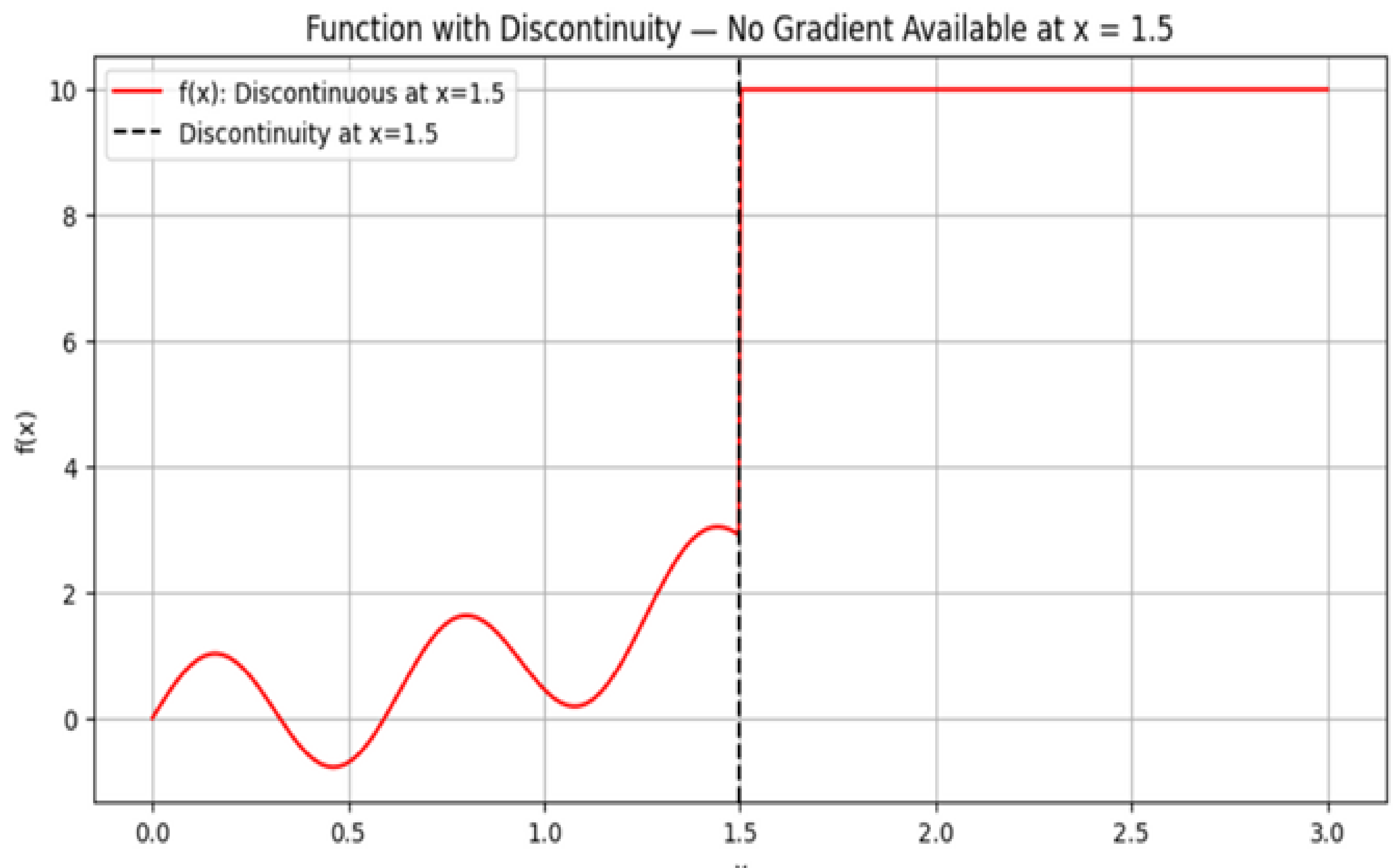
1	**Non-differentiable Functions**	▪ Gradient descent needs derivatives. ▪ Metaheuristics work with discrete or jump functions. ▪ **Example**: Choosing number of neurons (20 → 100 is a jump, not smooth)
2	**Local Minima & Saddle Points**	▪ Gradient can get stuck in a poor valley. ▪ Metaheuristics explore broadly and can escape. ▪ **Example**: Training a neural network may stop at 85% accuracy (local minimum), while swarm search continues to higher accuracy
3	**Multi-modal Problems**	▪ Many peaks/valleys mislead gradients. ▪ Metaheuristics keep multiple solutions at once. ▪ **Example**: Image segmentation cost functions often have many false minima.
4	**Curse of Dimensionality**	▪ In high dimensions, search space grows exponentially. ▪ Gradient descent struggles, swarm explores better. ▪ **Example**: Optimizing 100 hyperparameters for deep learning
5	**Exploration vs. Exploitation Balance**	▪ Gradient = local refinement only. ▪ Metaheuristics = global + local. ▪ **Example**: Genetic Algorithm explores new CNN structures, while fine-tuning improves them
6	**No Need for Gradient Information**	▪ Some functions are black-box, no gradient available. ▪ Metaheuristics only need function evaluations. ▪ **Example**: Tuning SVM kernel parameters (no analytic derivative).

1) Non-differentiable Functions

Let's define an objective function $f(x)$ as: $f(x) = \begin{cases} x^2 + \sin(10x) & \text{if } x < 1.5 \\ 10 & \text{otherwise} \end{cases}$

This function:

- Has a discontinuity at $x = 1.5$ so Gradient at $x = 1.5$ is Undefined. This is due to that left and right limits are not equal: $\lim_{h\to 0^-} \frac{f(1.5)-f(1.5-h)}{h} \neq \lim_{h\to 0^+} \frac{f(1.5+h)-f(1.5)}{h}$
- Cannot be differentiated symbolically at that point
- Gradient-based methods fail, while Metaheuristics still work by just evaluating $f(x)$ at multiple points
- Models real-world simulation outputs (e.g., cost of a robot arm movement)



Example with simple AI model for non differentiable case:

We need to tune two hyperparameters for a neural network to minimize validation loss after 5 epochs:

- x_1 =dropout rate, continuous: $x_1 \in [0.1, 0.5]$
- x_2 =number of hidden units, discrete: $x_2 \in \{16, 32, 64\}$

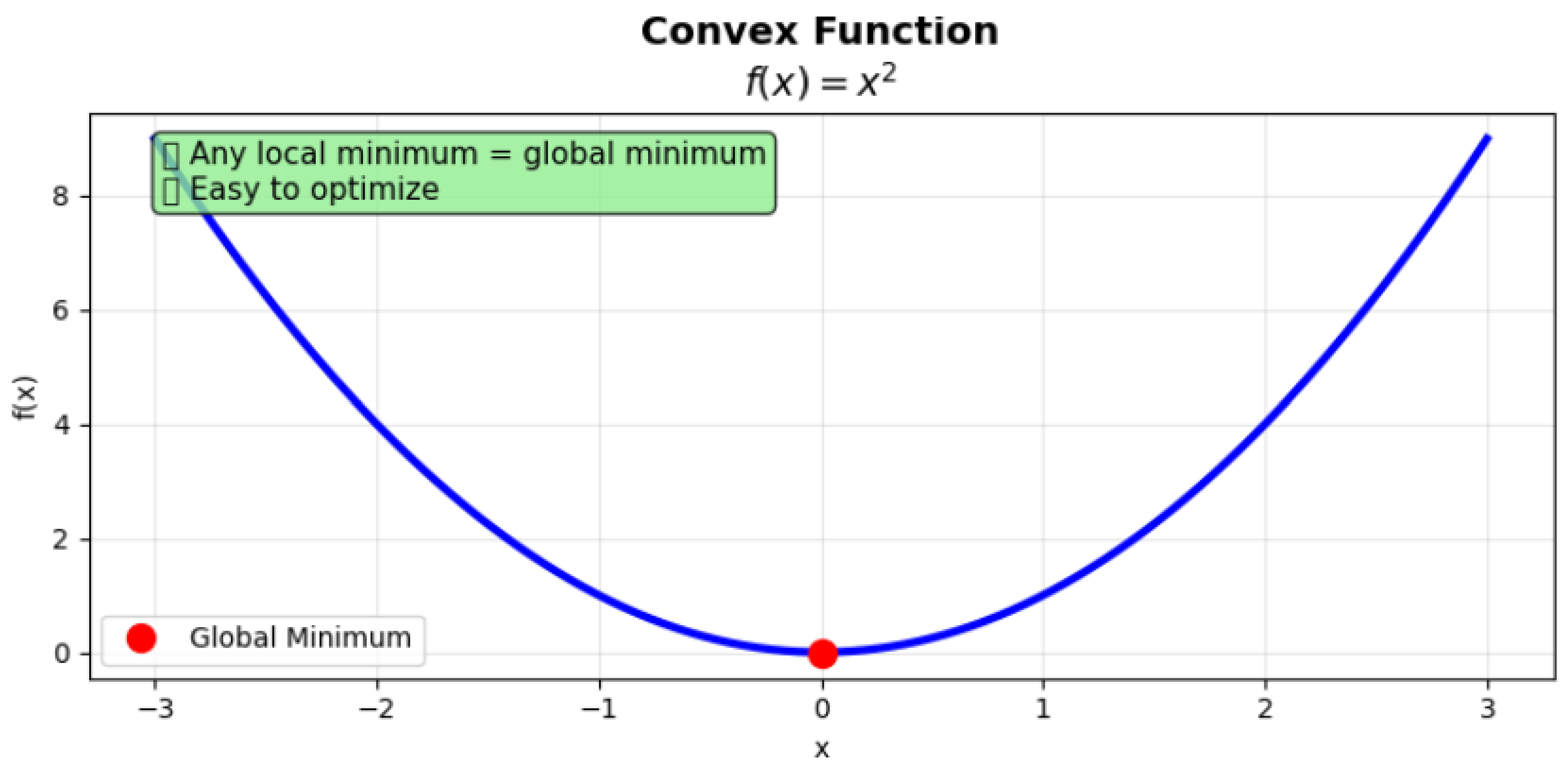
Why Gradient Methods Fail:

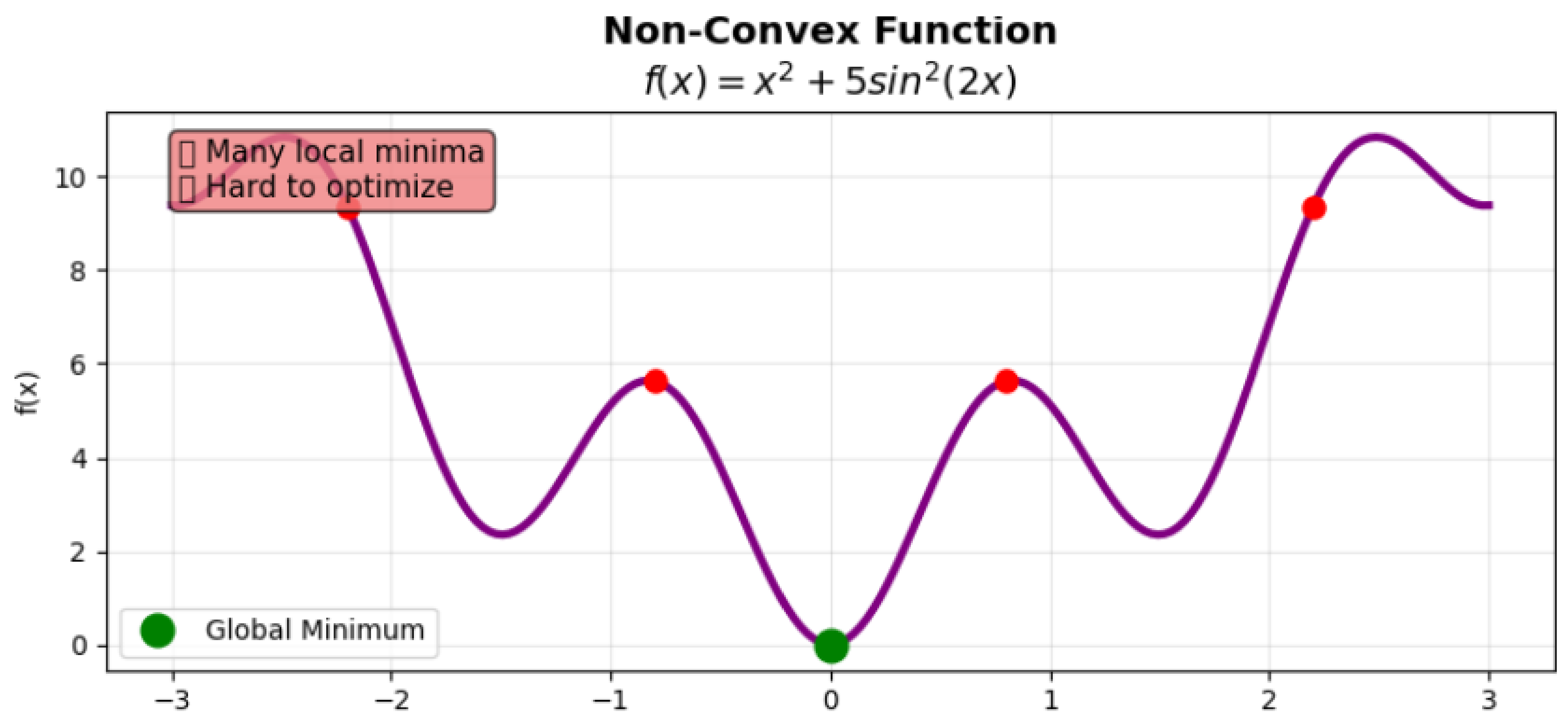
- Can't differentiate with respect to **discrete units** (16 → 32 → 64 are categorical jumps).
- Validation loss is noisy, so even with respect to dropout, gradients are unstable.
- Example of invalid gradient step:
$$[\text{dropout}, = 0.2\ \textit{units} = 32] \rightarrow [\textit{dropout}, = 0.3\ \textit{units} = 64]$$
- This is a jump, not a smooth path.

2) Local Minima & Saddle Points

- **Convex function** → Bowl-shaped, easy
- Any Local minima **=** global minimum
- Gradient descent always succeeds, no matter where you start
- Example: $\boldsymbol{f(x) = x^2}$

- **Non-Convex Function** →Wavy, tricky.
- May have multiple local minima.
- Gradient descent may get stuck in a local valley , missing the true bottom at x=0.
- Example: $\boldsymbol{f(x) = x^2 + 5\,sin^2(x)}$

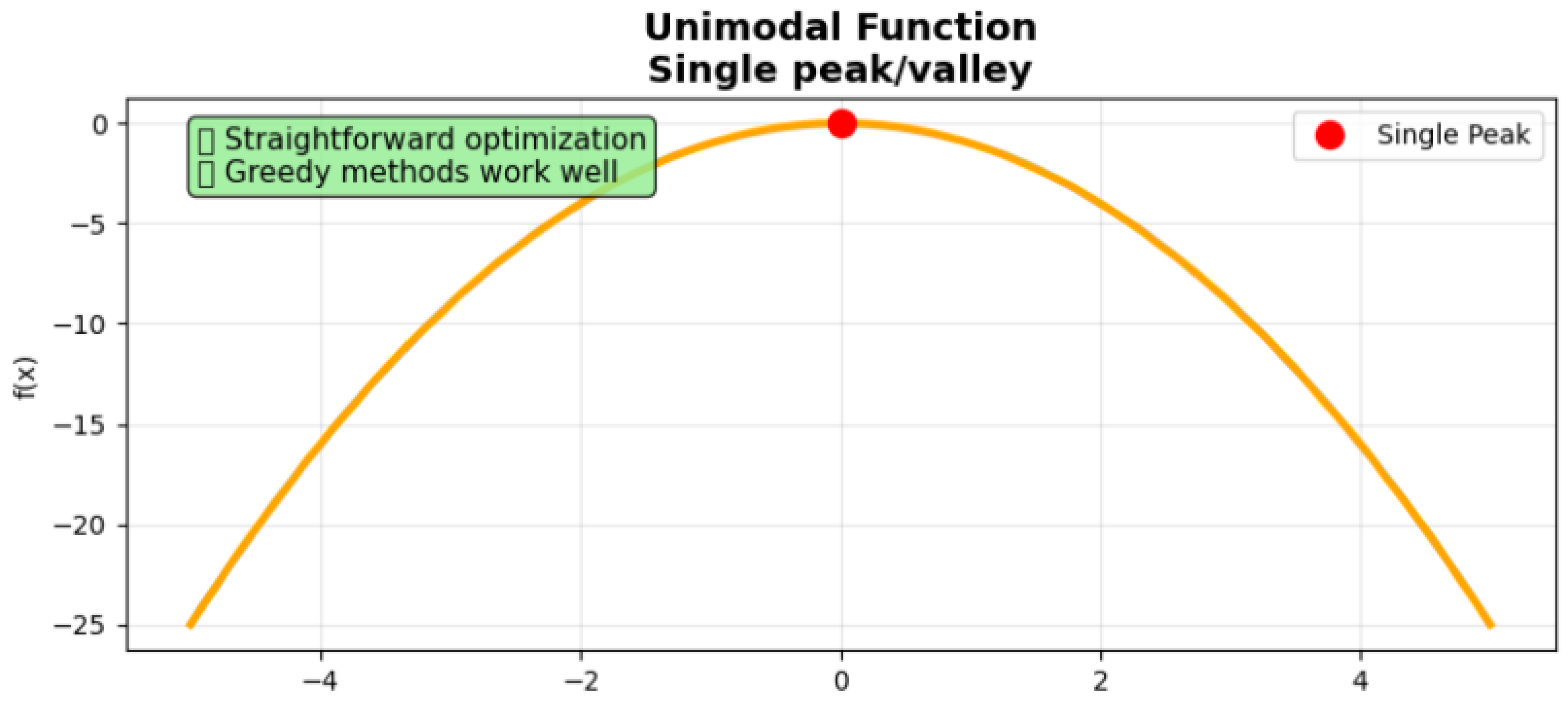


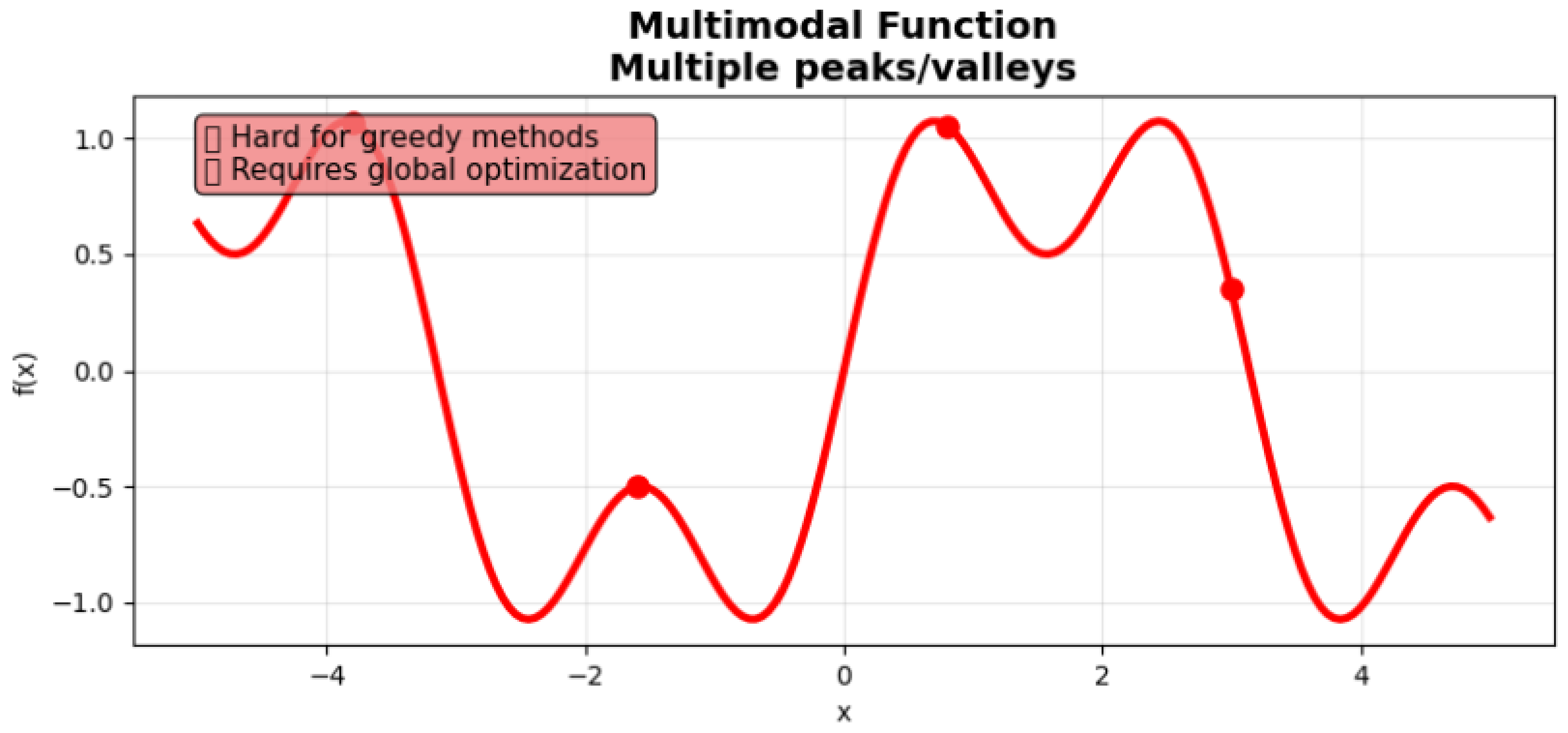


3) Multi-modal Problems

- Unimodal: **One peak/trough** → easy to optimize
- In unimodal functions, the slope always points toward the global minimum.

- Multimodal: **Many minima/maxima** → hard for greedy methods
- In multimodal functions, the slope may point toward a wrong valley → gradient descent gets trapped.





4) Curse of Dimensionality

- As the number of dimensions increases, the **search space grows exponentially**, making optimization harder.
- Suppose we want to tune 5 hyperparameters:

Hyperparameter	Number of choices	values
Number of neurons	5	{50, 100, 150, 200, 250}
Learning rate	3	{0.01, 0.001, 0.0001}
Batch size	3	{16, 32, 64}
Optimizer	3	{SGD, Adam, nAdam}
Activation	3	{ReLU, Tanh, Sigmoid}

- **Case studies:**

Case 1	1D Case (only 1 Hyperparameter): for example, only tuning learning rate	▪ just 3 options. ▪ Easy to search
Case 2	5D Case (all 5 hyperparameters together)	▪ $5 \times 3 \times 3 \times 3 \times 3 = 405$ ▪ much bigger

- Now, suppose we add more dimensions (say 10 hyperparameters) and suppose each has 5 options:
- $combinations = 5^{10} = 9765625$
- Impossible to check them all with gradient-based methods.

5) Exploration vs. Exploitation

In metaheuristic algorithms, two competing forces govern how we search through the solution space:

1. Exploration

- **Definition: Searching new areas** globally across the solution space (try *different areas*).
- **Goal:** To **discover new regions** that may contain better solutions.

2. Exploitation

- **Definition: improving known good areas** of the solution space. Intensively searching near known good solutions (improve the *current best*) .
- **Goal:** To **refine** and converge toward the local/global optimum.

Exploration vs. Exploitation in Metaheuristics

Example:

$$f(x) = x^2 + \sin(3x)\,, and\ x \in \{-2, -1, 0, 1, 2\}$$

- This is **non-convex** because the sine term creates “wiggles” (local minima and maxima).
- our target is to find only **one global minimum**.
- On non-convex functions, gradient descent may fail because it gets stuck in local minima. That’s why we use optimization methods like heuristics and metaheuristic, which balance exploration (global search) and exploitation (local improvement).”

Step 1: Exploration (searching widely):

test the following random points $\{-2, 0, 2\}$ from the given set

$$f(-2) = 4 + \sin(-6) = 4.3$$
$$f(0) = 0 + \sin(0) = 0$$
$$f(2) = 4 + \sin(6) = 3.72$$

Exploration shows that x=0 looks promising (value 0), but we don’t know if it’s the best.

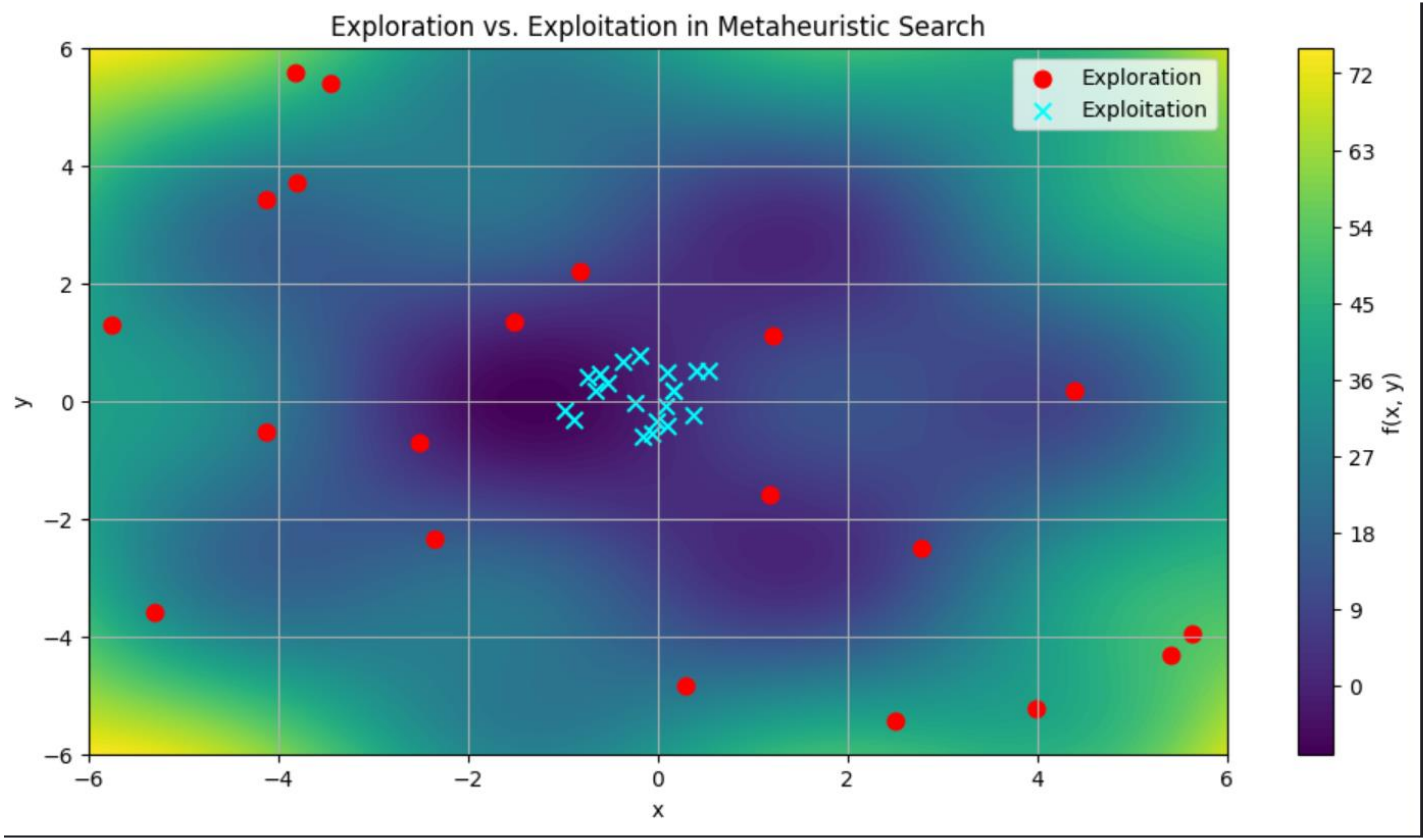
Step 2: Exploitation (refine locally near x=0):

Check neighbors near $x = 0$; let us say $\{-1, 1\}$

$$f(-1) = 1 + \sin(-3) = 0.859$$
$$f(1) = 1 + \sin(3) = 1.14$$

Local refinement or exploitation shows that $x = 0$ really is the best in this neighborhood.

How the behavior differs on the same function landscape on both



As AI engineers today:

1. We can design machine learning and deep learning models.
2. We can also make these models more interpretable and explainable.

But the big question is:

1. How can we apply **Metaheuristics** to optimize these models?
2. What is the **core idea** behind this approach?

Example:

We have a small neural network to classify images (say, MNIST digits). We want to **find the best learning rate** and **number of neurons in hidden layer** to **maximize** accuracy. Assume that:

Learning rate $x_1 \in [0.001, 0.1]$

Hidden neurons $x_2 \in [10, 100]$

The function to optimize:

$$Accuracy = f(learning\ rate, \quad hidden\ neurons)$$

Solution:

Step (1): initialize

Start with 3 random values:

Value	Learning rate	Hidden neurons
V1	0.01	20
V2	0.05	50
V3	0.02	80

Step 2: Evaluate Fitness

Train network and get accuracy:

Value	Learning rate	Hidden neurons	Accuracy
V1	0.01	20	85%
V2	0.05	50	88%
V3	0.02	80	82%

$\longrightarrow$ **Best case so far:** V2

Step 3: Update Positions (Exploration + Exploitation)

Move **toward the best solution** + some random exploration:

$$\mathrm{x}_{\mathrm{new}} = \mathrm{x}_{\mathrm{current}} + \alpha \times (x_{best} - x_{current}) + random\ noise$$

Where:

- α: step coefficient, how to **move toward the best solution**. assume α=0.1, this value is a small step but stable (if you increase this value, this would lead to no exploration)
- Random noise: simulates the **unpredictability of nature** (like a bird not flying in a perfectly straight line, or molecules moving randomly). we can assume:

V1	$noise_{Lr} = +0.0008$	$noise_{neurons} = +2$
V2	$noise_{Lr} = -0.0003$	$noise_{neurons} = +1$
V3	$noise_{Lr} = +0.009$	$noise_{neurons} = -6$

- In our example, best case was V2 (learning rate=0.05 and number of hidden neurons=50)
- Now, sub in update position equation:

Value	**Old Learning rate**	**New Learning rate**	**Old Hidden neurons**	**new Hidden neurons**
V1	0.01	$0.01 + 0.1 \times (0.05 - 0.01) + 0.0008 = 0.0148$	20	$20 + 0.1 \times (50 - 20) + 2 = 25$
V2 (best)	0.05	$0.05 + 0.1 \times (0.05 - 0.05) - 0.0003 = 0.0497$	50	$50 + 0.1 \times (50 - 50) + 1 = 51$
V3	0.02	$0.02 + 0.1 \times (0.05 - 0.02) + 0.009 = 0.032$	80	$80 + 0.1 \times (50 - 80) - 6 = 71$

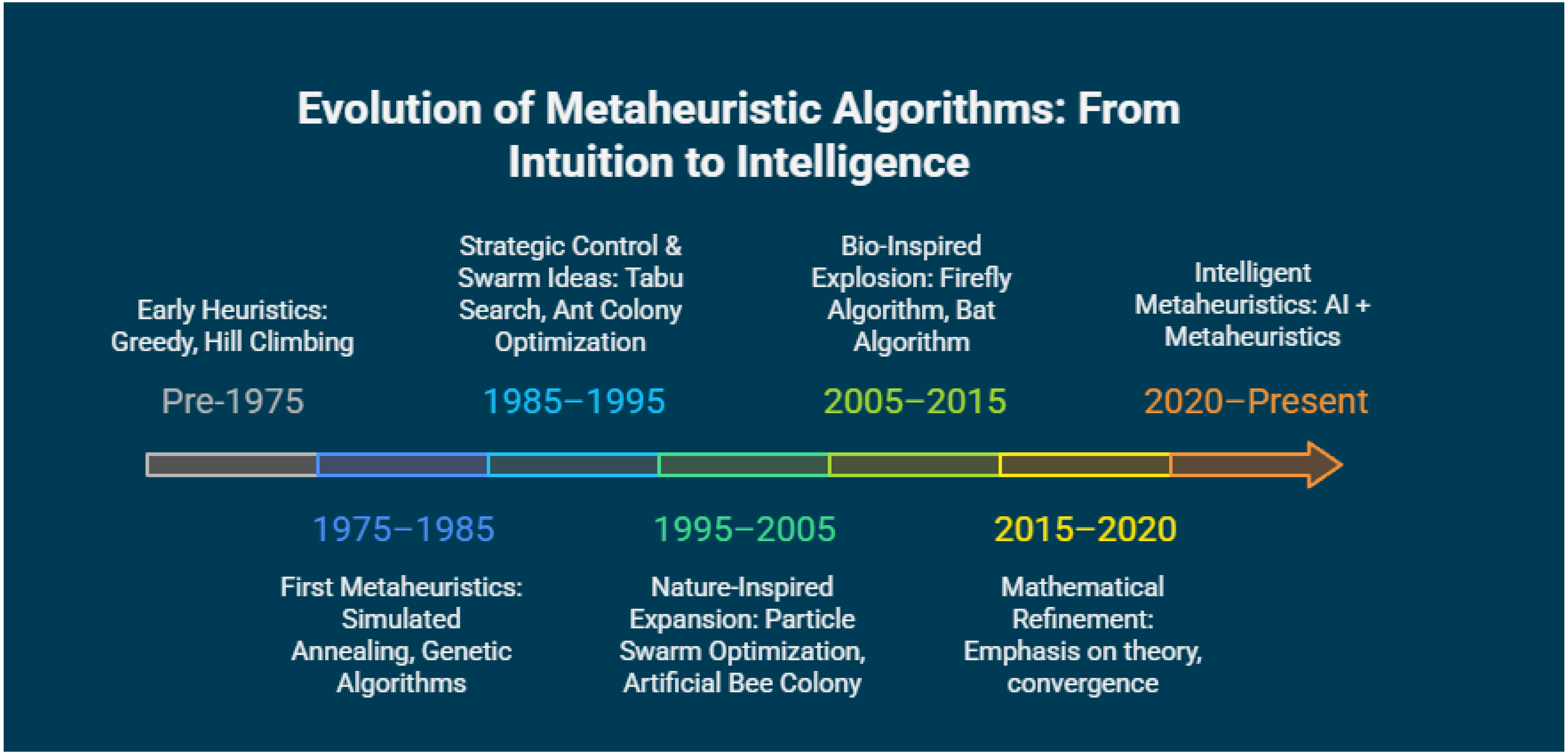
Step 4: Repeat Evaluation

Value	**Learning rate**	**Hidden neurons**	**Accuracy**
V1	0.0148	25	87%
V2	0.0497	51	89%
V3	0.032	71	85%

Step 5: Repeat updating until accuracy stops improving

- If we assume that accuracy is not improving after the first iteration, then best hyperparameters found: Learning rate ≈ 0.0497, Hidden neurons ≈ 51
- Accuracy ≈ 88–89%

Evolution of Metaheuristic


Evolution of Metaheuristic Algorithms: From Intuition to Intelligence
Early Heuristics: Greedy, Hill Climbing
Pre-1975
Strategic Control & Swarm Ideas: Tabu Search, Ant Colony Optimization
1985–1995
Bio-Inspired Explosion: Firefly Algorithm, Bat Algorithm
2005–2015
Intelligent Metaheuristics: AI + Metaheuristics
2020–Present
1975–1985
First Metaheuristics: Simulated Annealing, Genetic Algorithms
1995–2005
Nature-Inspired Expansion: Particle Swarm Optimization, Artificial Bee Colony
2015–2020
Mathematical Refinement: Emphasis on theory, convergence

Our Course

z	Lecture	Assessments
Week 1	▪ Introduction to Optimization & Metaheuristic	
Week 2	▪ Local Search & Hill Climbing	
Week 3	▪ Simulated Annealing (SA)	Start the project
Week 4	▪ Tabu Search & Memory Structures	
Week 5	▪ Genetic Algorithms (GA) and Genetic Programming (GP)	Quiz 1
Week 6	▪ Particle Swarm Optimization (PSO)	
Week 7	▪ Ant Colony Optimization (ACO) & Whale optimization algorithm	Discussion of phase 1 of the project
Week 8	Midterm	
Week 9	▪ Firefly Algorithm & Cuckoo Search	
Week 10	▪ Bat Algorithm & Flower Pollination Algorithm	
Week 11	▪ Multi-objective Optimization (MOO)-Part(I)	
Week 12	Multi-objective Optimization (MOO)- Part(II)	
Week 13	▪ Hyper-Heuristics	Quiz 2
Week 14	▪ Automated Algorithm Selection	Delivering and discussion of the final project ▪ Phase 2 ▪ final documentation + notebooks
Week 15	▪ Real time applications for nature inspired computation	Publishing final course work
Week 16	Final term	

Grading system

	Total number	Grade of each topic	Total weight
Lap attendance	10	1	10
Lap practice (LP)	10	1	10
Quizzes	2	5	10
Projects	1	10	10
Total weight			**40**
Midterm			20
Final term			40

References

1. E. Cuevas, A. Chavarin-Fajardo, C. Ascencio-Piña, and S. Garcia-De-Lira, *Optimization Strategies: A Decade of Metaheuristic Algorithm Development*. Cham, Switzerland: Springer, 2025. doi: 10.1007/978-3-031-81013-8
2. E. Cuevas, D. Zaldívar, and M. Pérez-Cisneros, *New Metaheuristic Schemes: Mechanisms and Applications*. Cham, Switzerland: Springer, 2024. doi: 10.1007/978-3-031-45561-2
3. B. Chopard and M. Tomassini, *An Introduction to Metaheuristics for Optimization*. Cham, Switzerland: Springer, 2018. doi: 10.1007/978-3-319-93073-2
4. M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*, 2nd ed. Boston, MA: Springer, 2010. doi: 10.1007/978-1-4419-1665-5