SME3006 Machine Learning – 2025 Fall

Hyperparameter tuning and gradient-free optimization

INHA UNIVERSITY

Optimization problem

> Finding the minimizer of a function subject to constraints:

$$\underset{x}{\text{minimize}}\, J(x)$$

objective function

$$s.t.\, f_i(x) \leq 0,\ \ i = \{1, \dots, k\}$$
$$h_j(x) = 0,\ \ j = \{1, \dots, l\}$$

constraints

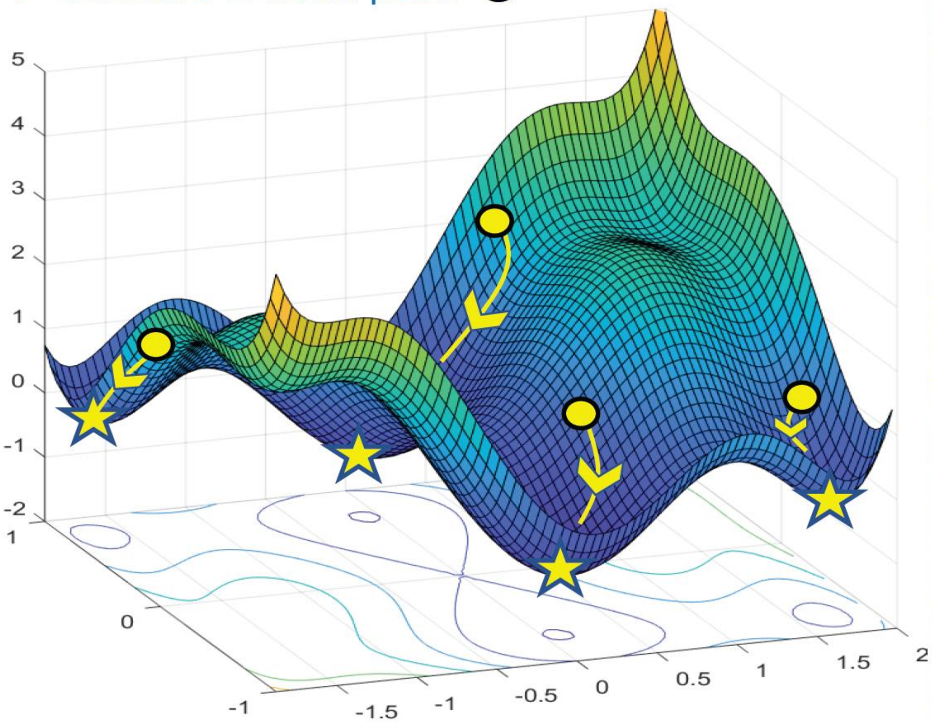
- Linear regression: $\underset{w}{\text{minimize}} \|Xw - y\|^2$
- Classification (logistic regression): $\underset{w}{\text{minimize}} \sum_{i=1}^{n} \log(1 + \exp(-y_i x_i^\top w))$
- Maximum likelihood estimation: $\underset{\theta}{\text{maximize}} \sum_{i=1}^{n} log p_\theta(x_i)$
- K-means: $\underset{\mu}{\text{minimize}} \sum_{j=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$

...

Optimality conditions

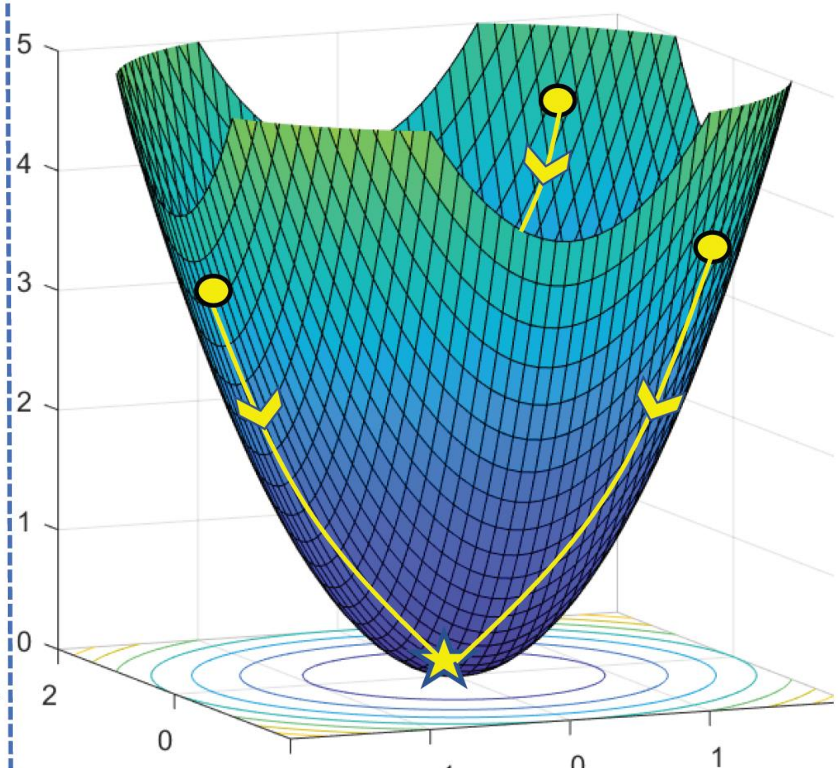
Nonconvex Optimization

- Multiple local minima ★
- Sensitive to initial point ●



Convex Optimization

- Unique minimum: global/local



Hyperparameter tuning

> Gradient-free (derivative-free) optimization
  - When gradients cannot be computed
    - discontinuous or non-smooth design space (discrete or mixed-integer variables)
    - hyperparameters of machine learning models
      e.g. learning rate, kernel type, regularization choice
    - black-box simulators with no analytic derivates
  - When following gradients is inefficient or unreliable
    - computing/approximating gradient is too costly relative to function evaluations
    - objectives/constraints are noisy or stochastic
    - complex landscapes with many local minima where gradients mislead search
  - Gradient-free methods have a higher chance of finding the global optimum

Hyperparameter tuning

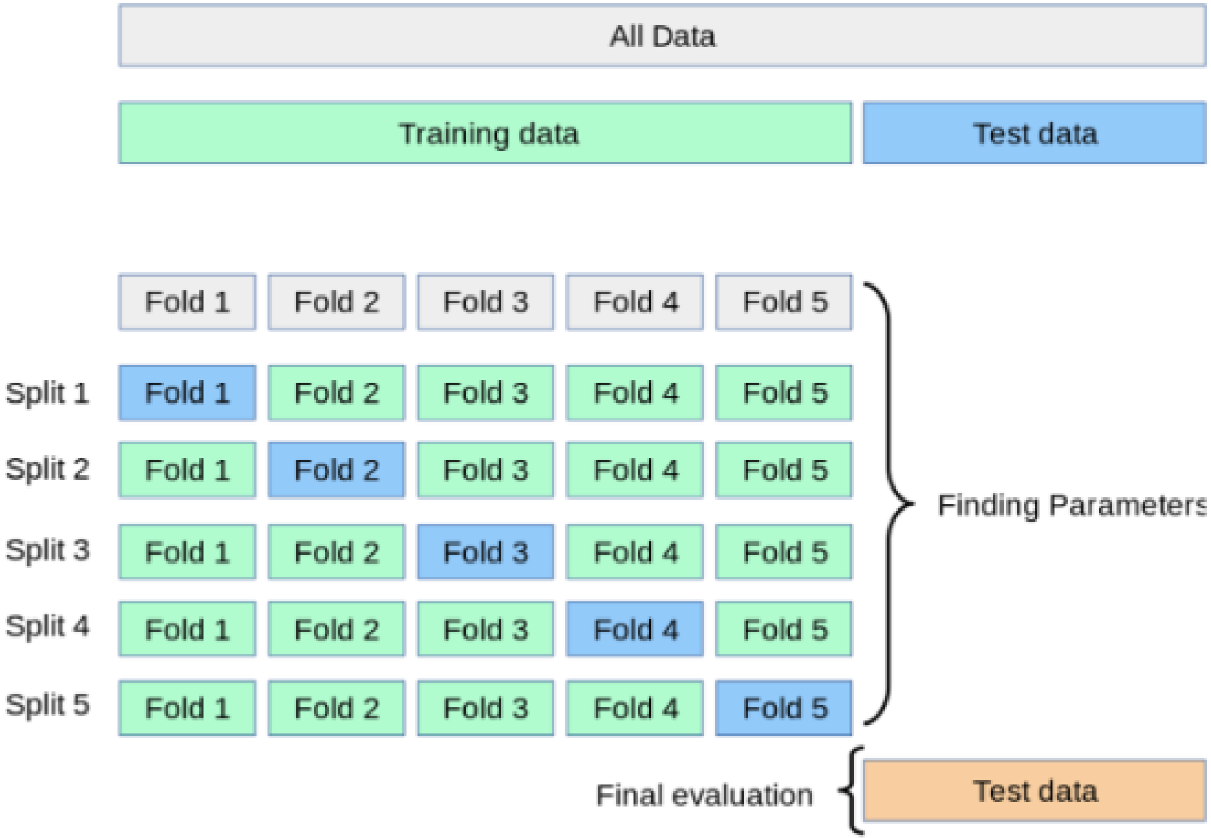
- Model selection
  - which input features to include
  - what preprocessing to do
  - what machine learning method to use

- Hyperparameter tuning
  - for k-nearest neighbors, hyperparameters include:
    - k
    - metric (e.g. Euclidean distance)

- We use cross-validation and pick the model / hyperparameter with the smallest test error

Hyperparameter tuning

> Cross-validation
  - splitting the data into multiple train/validation sets

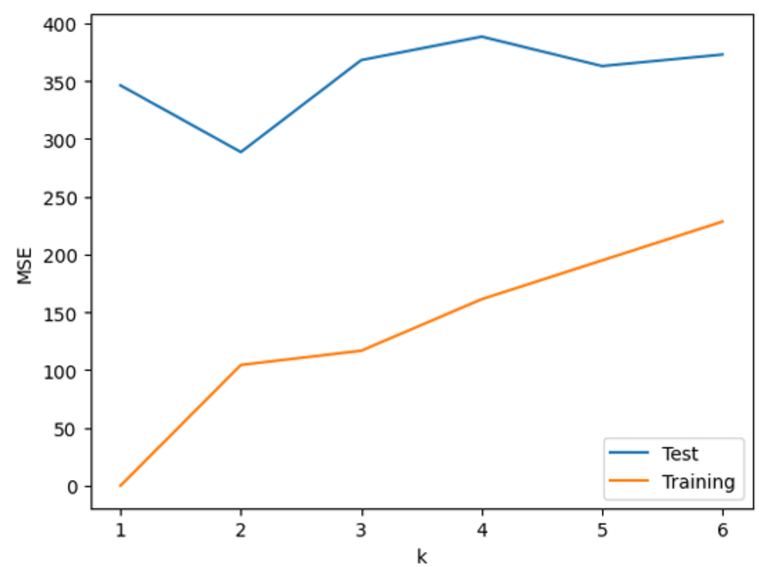


Hyperparameter tuning

> Wine quality prediction

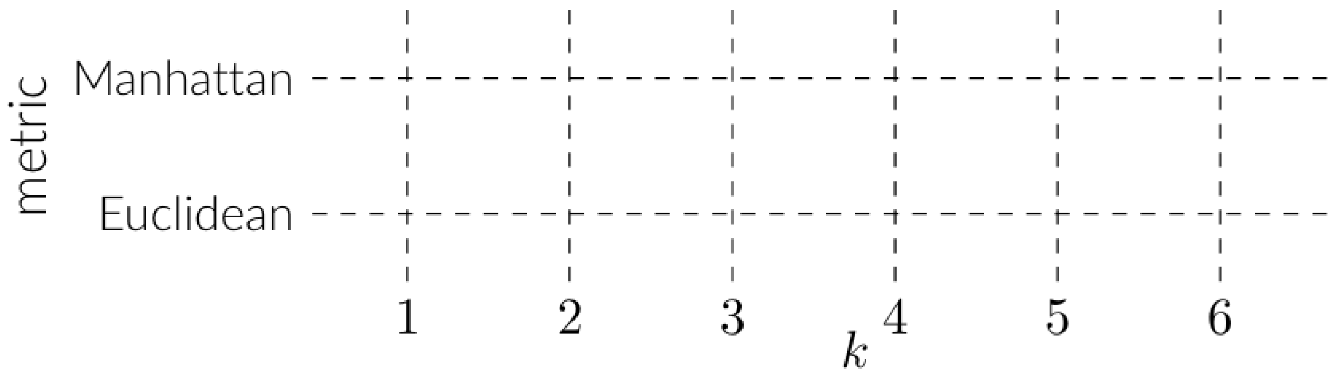
Hyperparameter tuning

> Wine quality prediction
  - Model selection
  - which input features to include
    - winter rain, summer temp, harvest rain, September. temp, age
  - Hyperparameter tuning
  - for k-nearest neighbors, hyperparameters include:
    - k
    - metric (e.g. Euclidean distance)



Hyperparameter tuning

> Grid search
  - We need to try all 12 combinations on the following grid:



  - 2 possible scaler (standard scaler, minmax scaler)
  - there were 5 input features in the original data
  - $2^5 = 32$ combinations of features we need to try

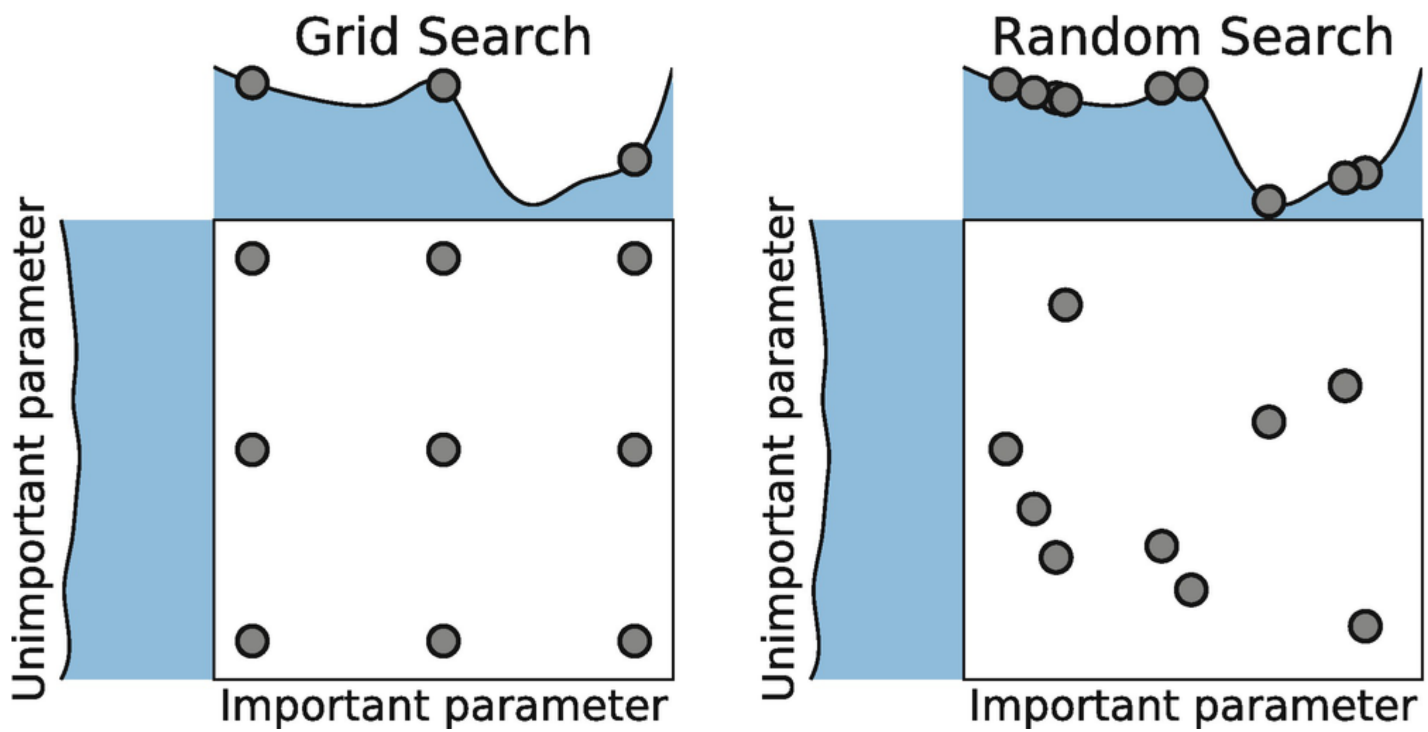
  - In total, 32*12*2=768 models

Hyperparameter tuning

- For large datasets, it is impossible to try every combination of models and parameters
- So instead we use heuristics, which do not guarantee the best model but tend to work well in practice
  - Randomized search: try random combinations of parameters
  - Coordinate optimization
    - start with guesses for all parameters
    - try all values for one parameter (holding the rest constant) and find the best value of that parameter
    - cycle through the parameters
  - …

Random search

- Explores by choosing a new position at random after each iteration
  - purely random across the search space in each step



Random search

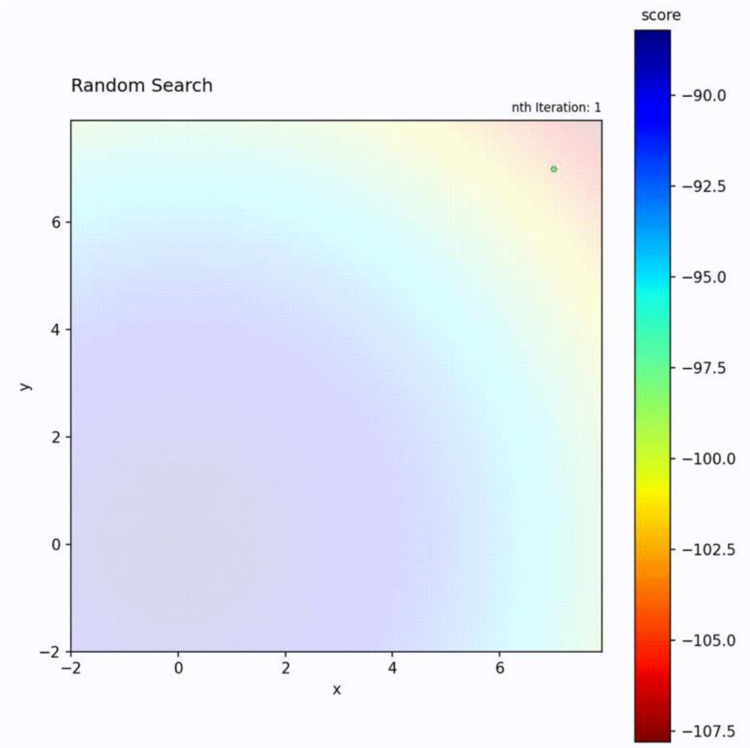
- Appropriate for solving deterministic objectives
  - often have promising theoretical properties
    - convergence in probability
    - hitting probability: to land in an ϵ-optimal region with probability at least p

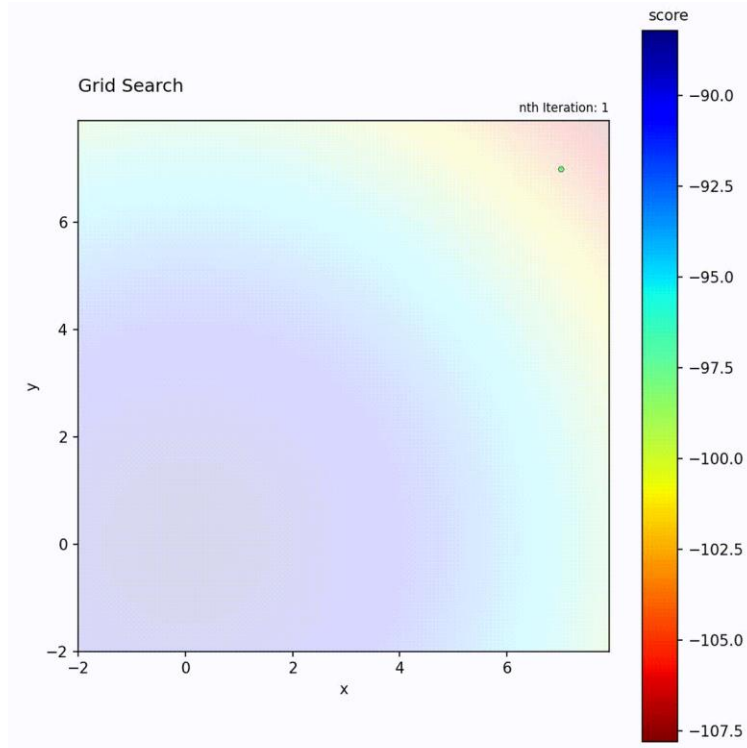
  - Pros
    - extremely easy to implement
    - no smoothness/continuity assumed
    - perfect parallelizable (i.i.d. samples)
    - all points can be generated in advance

  - Cons
    - convergence rate is slow (compared to methods that exploit continuity/gradients)
    - requires large N to achieve good probability guarantees

Random search

> RS samples points independently in each iteration, while grid search systematically explores the search space along predefined directions





Random search

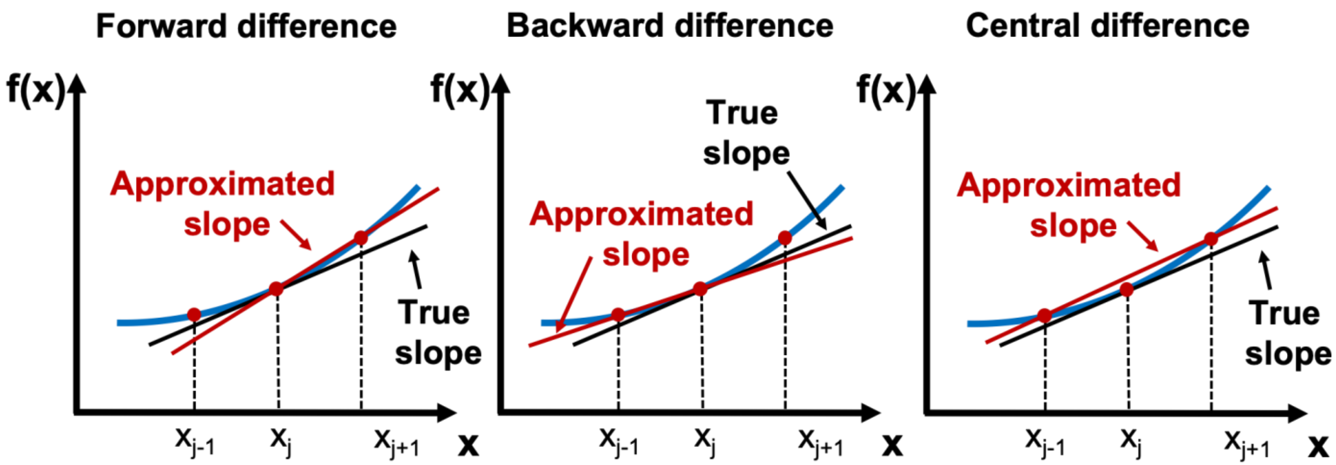
- How to advance the random search?
  - RS explores blindly
    - ignores state space structure and history
  - Once we collect information, we can guide the search

  - Gradient-free
    - when true gradient is unavailable (or noisy)
    - approximate with sampled perturbations

  - Derivative-free
    - do not compute or approximate gradient at all
    - rely only on function values to guide sampling
      (search more in promising regions)

Gradient-free optimization

> Finite difference – coordinate perturbations
  - estimate partial derivatives by probing each axis
  - gradient estimate: $f_j' = \frac{f(x+he_j)-f(x)}{h}$ or $f_j' = \frac{f(x+he_j)-f(x-he_j)}{2h}$

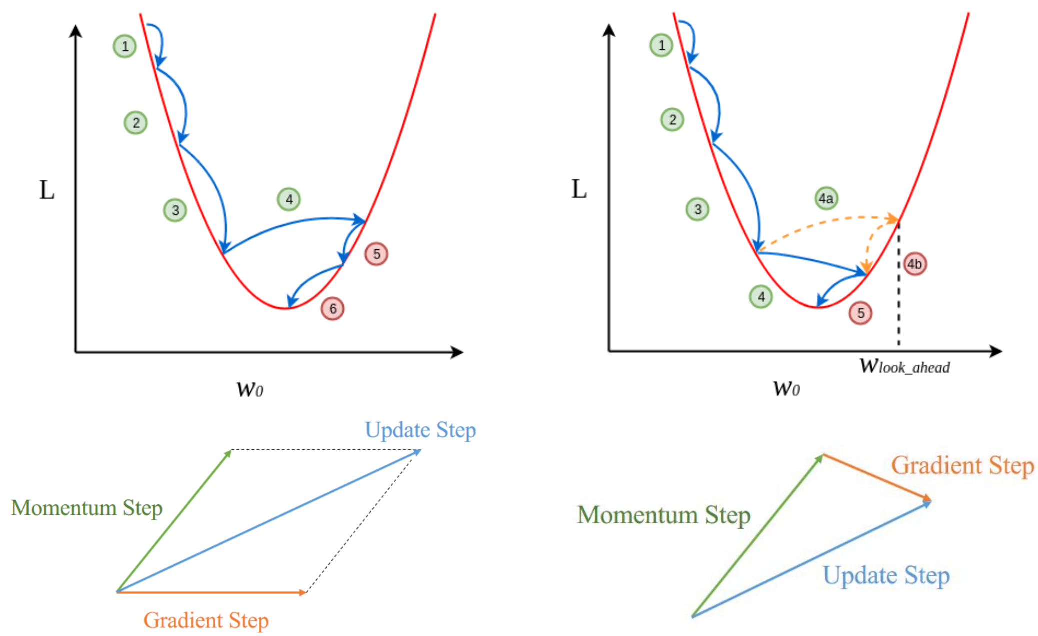


  - simple and transparent
  - scales poorly with dimension; sensitive to noise/non-differentiability

Gradient-free optimization

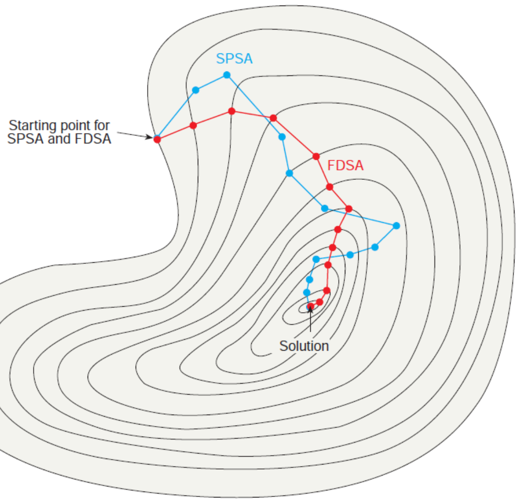
- Evolution strategy
  - sample around the current mean with Gaussian noise
  - use reward-weighted averaging to estimate a gradient
  - more sample efficient than FD in high dimension
  - Nesterov’s random search
    - use momentum from past updates to accelerate

recall Nesterov’s momentum



Gradient-free optimization

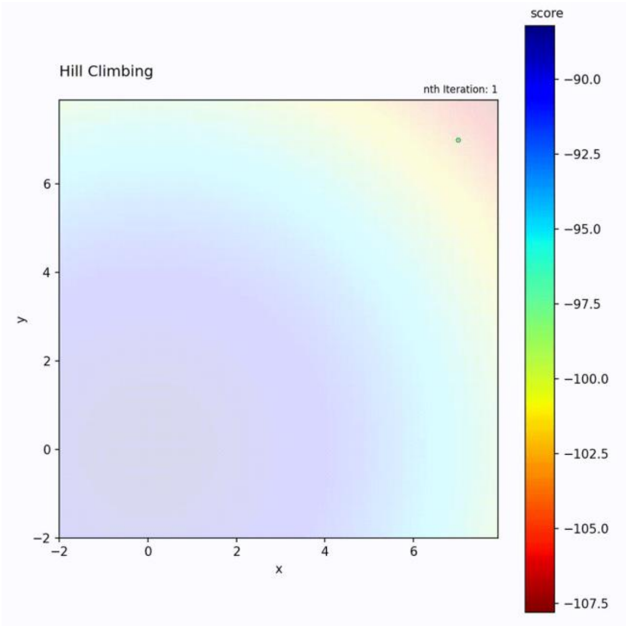
> Simultaneous Perturbation Stochastic Approximation (SPSA)
  - estimate a full gradient with only two function evaluations
  - generate a random perturbation vector $\Delta \in \{\pm 1\}^d$
  - Gradient estimate: $\hat{g}_j = \frac{f(x+c\Delta)-f(x-c\Delta)}{2c\Delta_j}$
  - extreme case of gradient approximation
    - ultra sample-efficient per iteration; great for very high-dim., noisy problems
    - higher variance than ES;

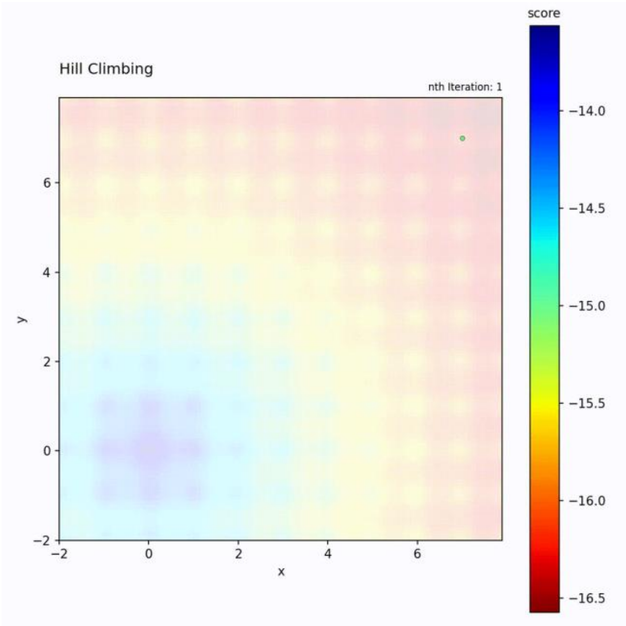


Derivative-free local search

> Derivative-free
  - methods that do not explicitly estimate a gradient, but still improve solutions via local moves

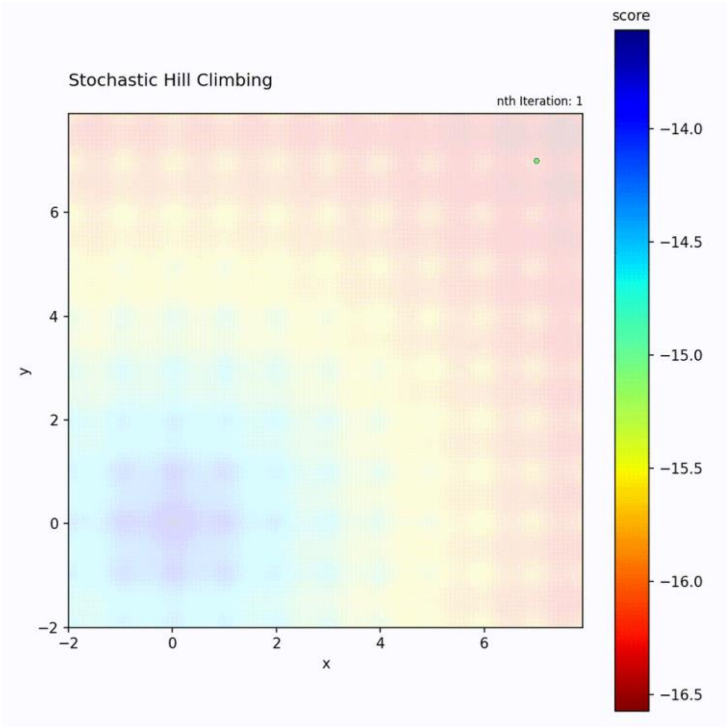
> Hill climbing
  - greedy local improvement: probe neighbors, move if better
  - evaluate the score of n neighbors in an epsilon environment and moves to the best one

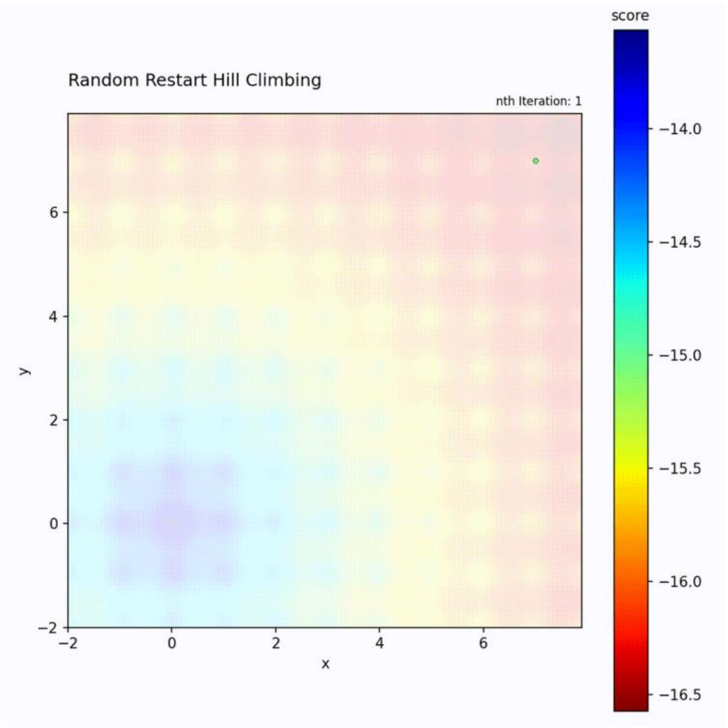




Derivative-free local search

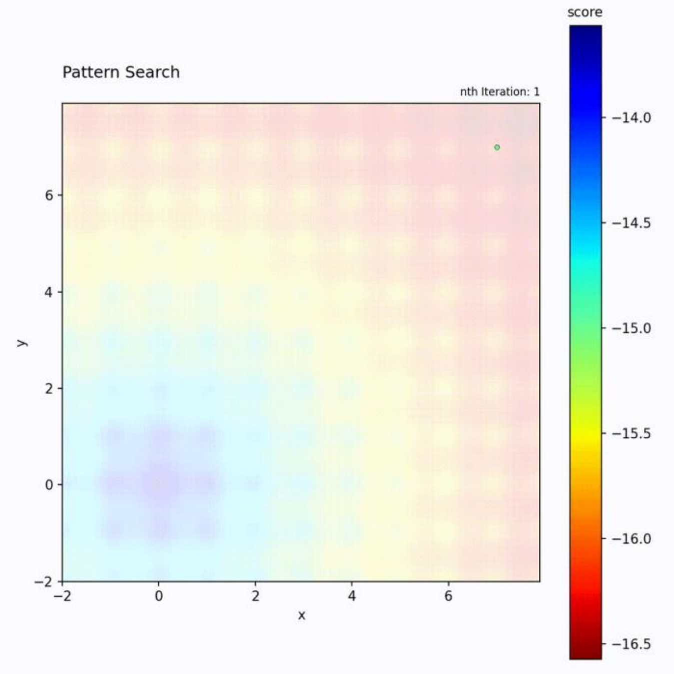
- Hill climbing variations
  - stochastic: adds a probability to move to a worse position
  - random restart: moves to a random position after n iterations.





Derivative-free local search

- Pattern search
  - probe a set of directions at a given step size
  - if any direction yields improvement, move and possibly enlarge steps otherwise shrink step and try again
  - robust to noise and non-smoothness
  - can be inefficient in ill-conditioned landscapes



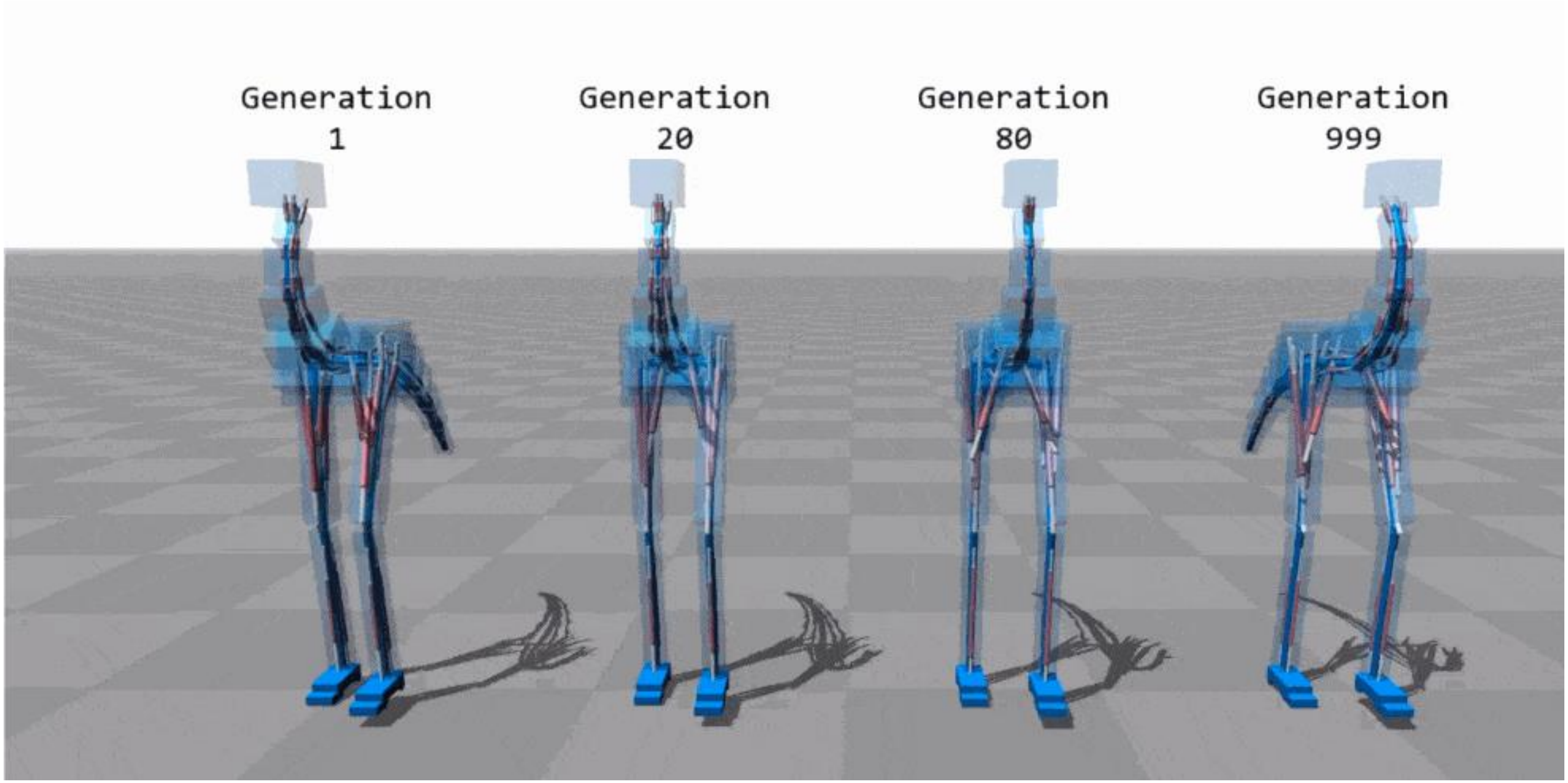
Population-based metaheuristics

> Local search
  - explores only the immediate neighborhood of the current solution
  - very efficient for smooth, low-dimensional problems
  - easily trapped in local optima
  - stochastic / random restart methods are partially global but still limited

> Global search
  - need methods that systematically explore the entire search space
  - population-based and distribution-based approaches

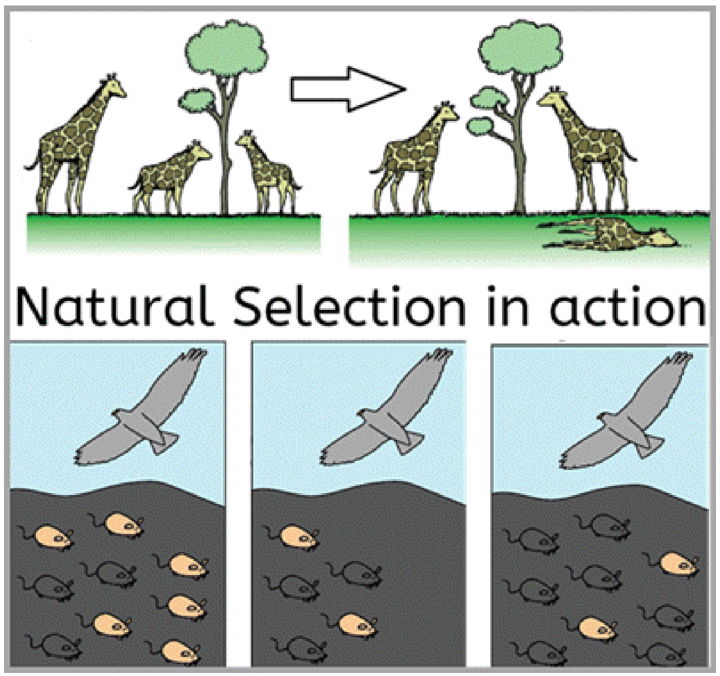
Population-based metaheuristics

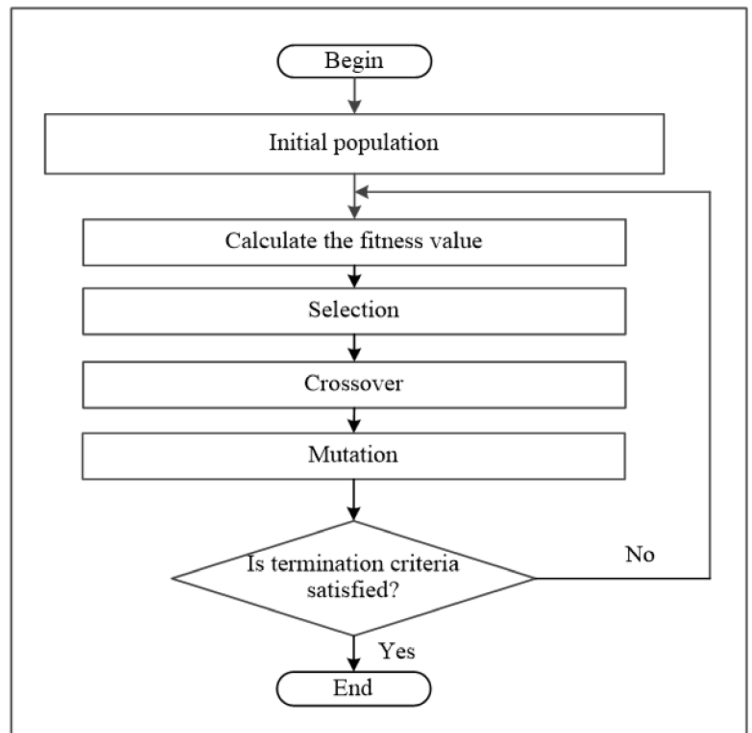
> Genetic algorithm



Population-based metaheuristics

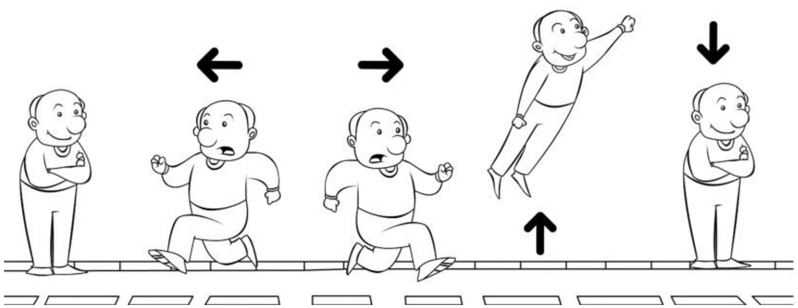
> GA is a heuristic search that mimics the natural evolution
  - it is a particular class of evolutionary algorithms inspired by evolutionary biology such as mutation, selection, and crossover
  - select the best, discard the rest

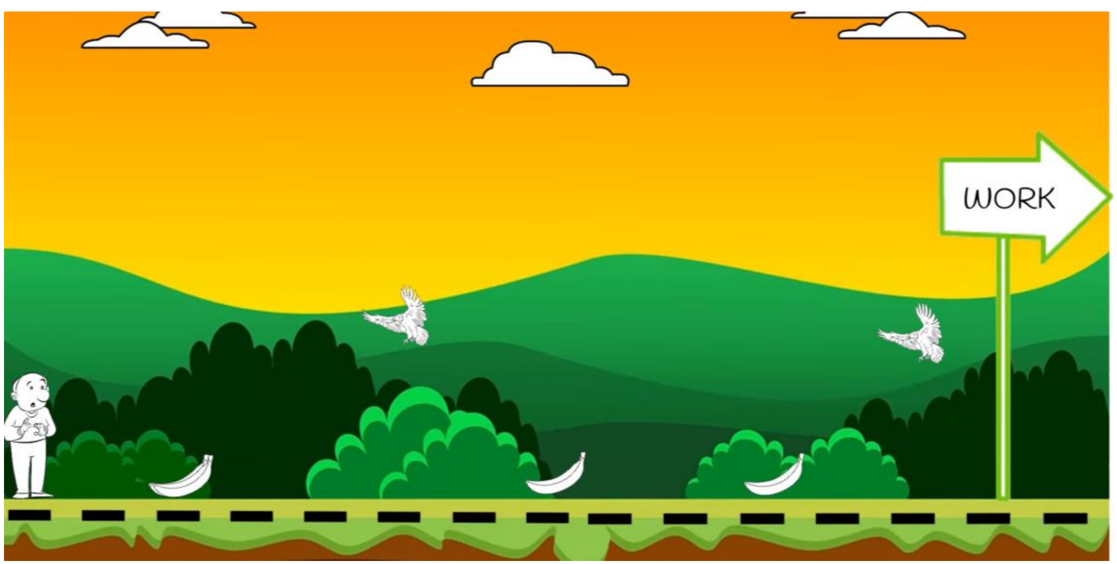




Population-based metaheuristics

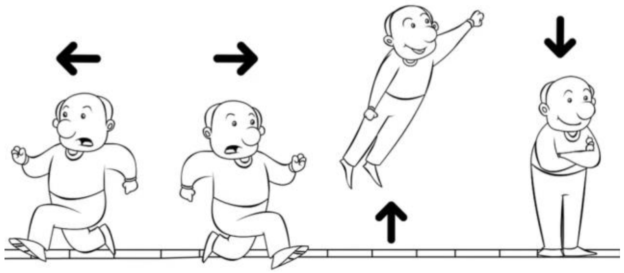
> Find a path to work safely



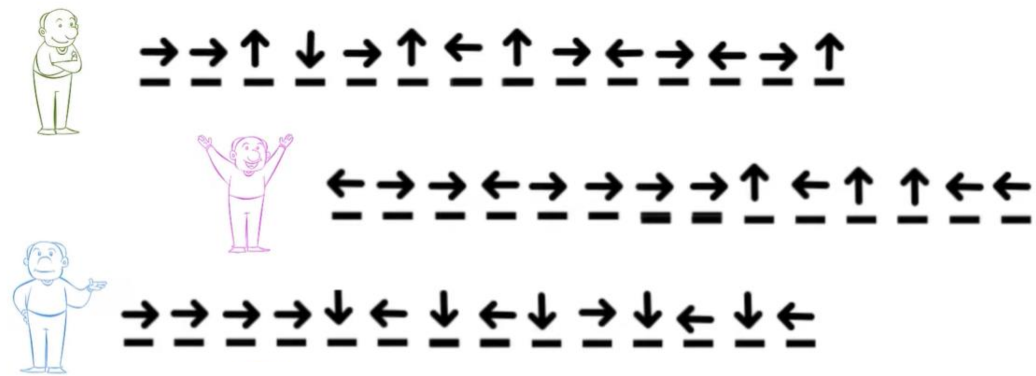


Population-based metaheuristics

> GA – binary encoding



00 01 10 11



01011011011**0**0010…
000101**00**01010101…
01**0**101011100**1**100…
…

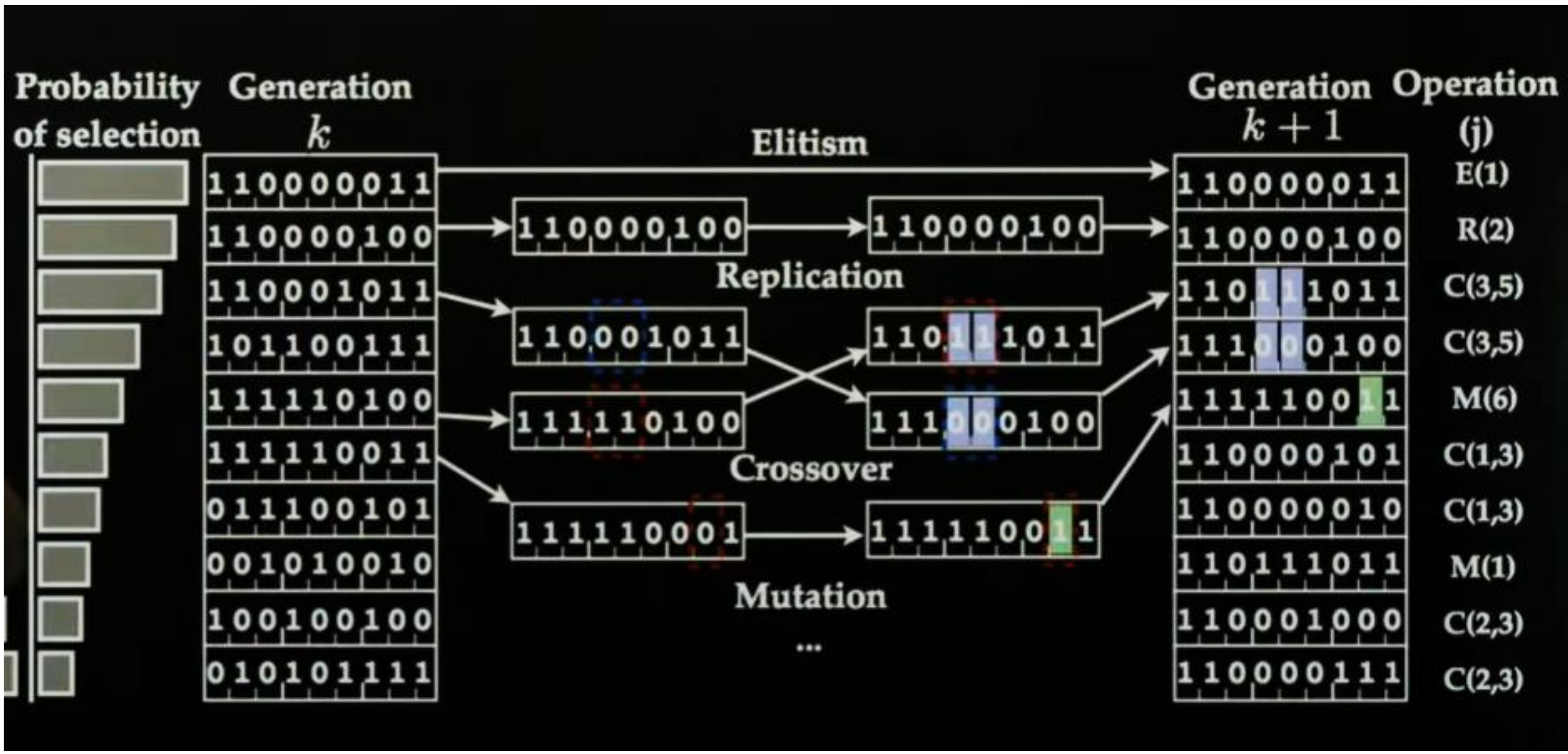
⟹ mutation / crossover / …

01011011011**1**010…
000101**11**01010101…
01**1**101011100**0**100…
…

⟹ calculate the fitness value (evaluate)
selection

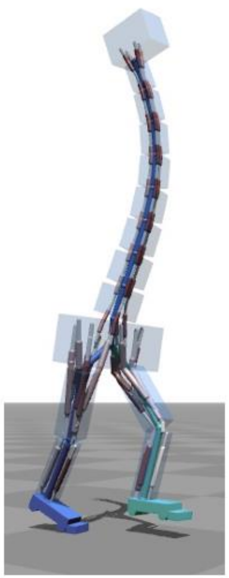
Population-based metaheuristics

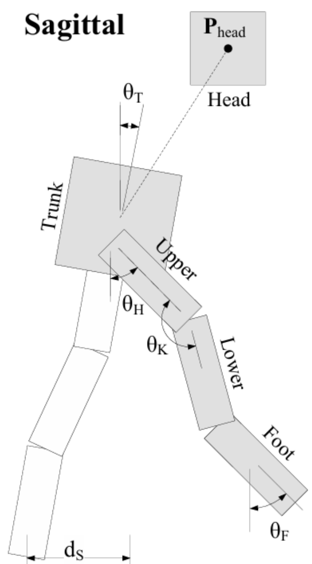
- GA – binary encoding



Population-based metaheuristics

- GA – real-value encoding
  - what if we want to represent the continuous value





1) binning θ to 255 levels

01111111 (127/255) → 10000000 (128/255)

2) gene encoded as real values = [1.2, 3.5 …]

crossover: child = $\alpha x^{(1)} + (1-\alpha)x^{(2)}$
mutation: child = $x + \mathcal{N}(0, \sigma^2)$

Population-based metaheuristics

- Pros and cons
  - GA search a population of points in parallel, not only a single point
    - can be easily parallelized
  - It works well on mixed discrete/continuous problems
  - Simple to understand and set up

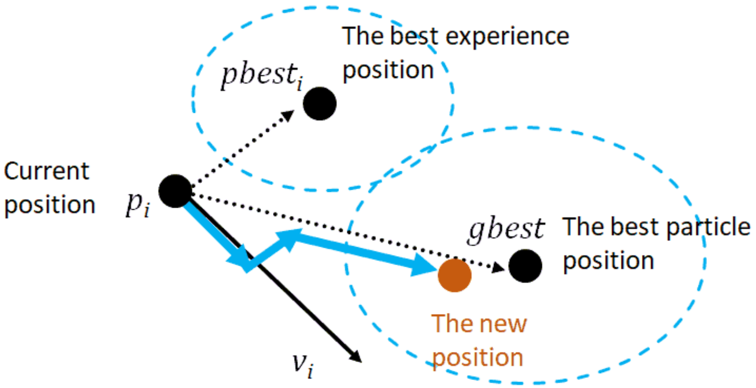
  - convergence behavior is very dependent on tuning parameters
  - cumbersome to take into account constraints
  - no clear termination criteria

Population-based metaheuristics

- Particle Swarm Optimization (PSO)
  - Inspired by social behavior of bird flocking
  - Suppose a group of birds is searching food in an area
    - birds can remember the best place it has found so far
    - birds can see the best location found by other birds

Population-based metaheuristics

- Particle Swarm Optimization (PSO)
- Algorithm
  - position x_i
  - velocity v_i
  - personal best position p_i
  - global best position g



  - velocity and position update rules
  - $x_i(t+1) = x_i(t) + v_i(t+1)$
  - $v_i(t+1) = \underline{wv_i(t)} + \underline{c_1 r_1\big(p_i - x_i(t)\big)} + \underline{c_2 r_2\big(g - x_i(t)\big)}$

Inertia – keeps the particle moving in the same direction

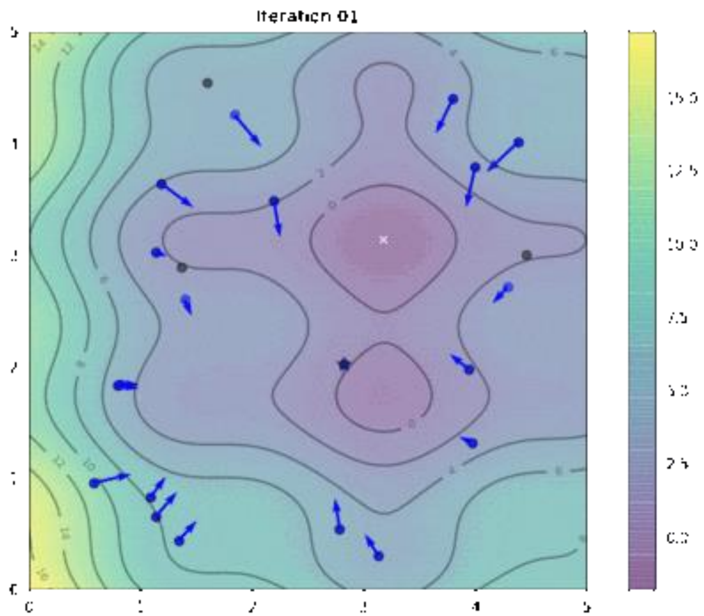
Cognitive – pulls the particle back toward its own best-known position

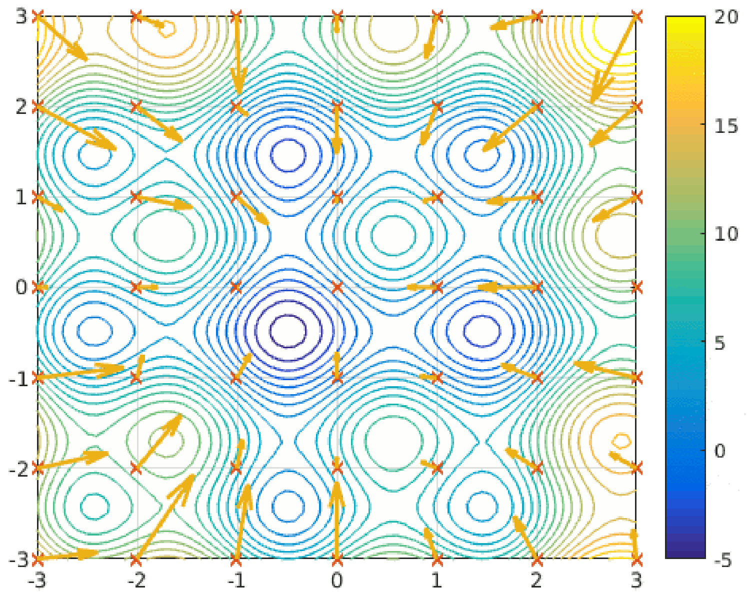
Social – pulls the particle toward the swarm’s best-known position

  - r_1, r_2 : random numbers introducing stochasticity

Population-based metaheuristics

> Particle Swarm Optimization (PSO)





Population-based metaheuristics

- Pros and cons
  - simple and easy to implement
  - works well in high-dimensional, nonlinear, or noisy problems

  - may converge prematurely to a local minimum
  - sensitive to parameter tuning
  - no theoretical convergence guarantees

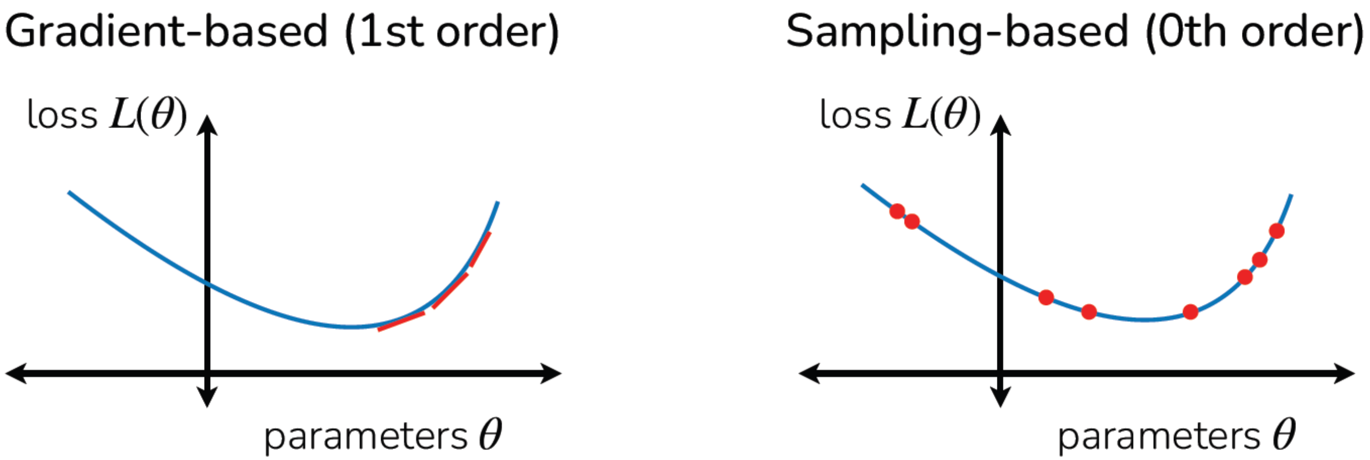
- Applications
  - control system parameter tuning

Population-based metaheuristics

- Nature-inspired metaheuristic optimization
- Collective behavior of animals (Swarm intelligence)
  - Ant Colony Optimization (ACO)
    - discrete domain (e.g. route optimization, TSP)
  - Bacterial Foraging Optimization (BFO)
  - Artificial Bee Colony (ABC)
  - Firefly algorithm (FA)
  - …

Stochastic sampling-based optimization

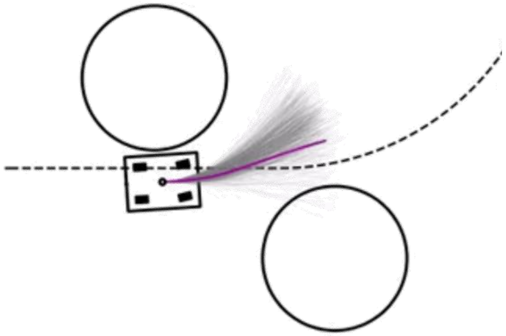
> Sampling-based optimization



- requires no gradient information (gradient-free optimization)
- use randomness to explore the space
- often better at escaping local minima than gradient-based methods
- parallelizable
- scales poorly to high dimensions

Stochastic sampling-based optimization

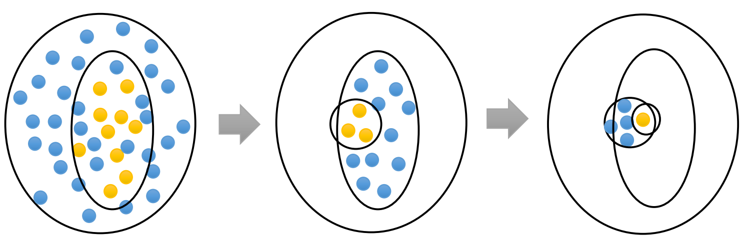
> Sampling-based optimization
  - Random shooting
    - Guess and check
    - sample many action sequences ($A = \{a_t, \dots, a_{t+H}\}$) $A_1, \dots, A_n$ from some distribution (e.g., uniform)
    - choose the best A_i (max return)



    - can we improve the sampling distribution? (prior knowledge)

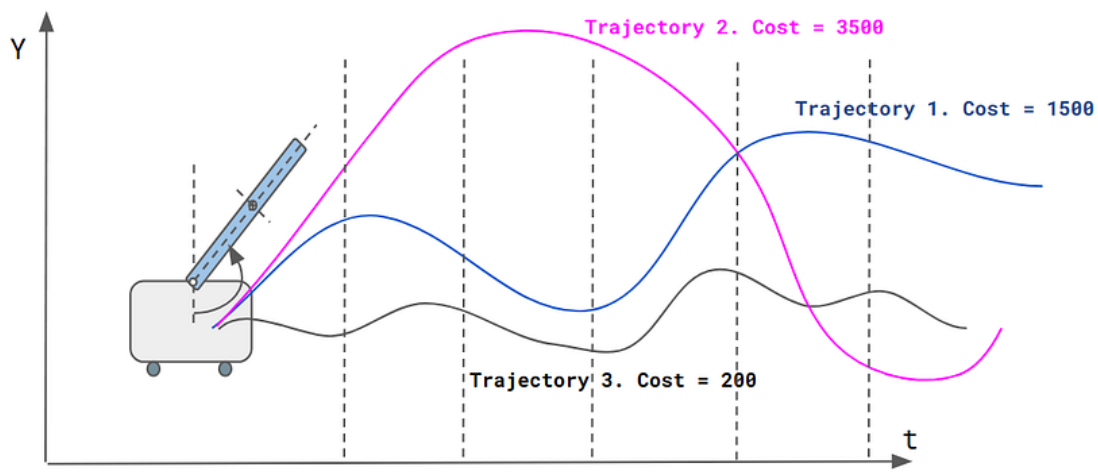
  - Cross-entropy method (CEM)
    - sample many action sequences $A_1, \dots, A_n$ from $p(A)$
    - evaluate and pick elites $A_{i_1}, \dots, A_{i_m}$
    - refit $p(A)$ to the elites $A_{i_1}, \dots, A_{i_m}$

until convergence



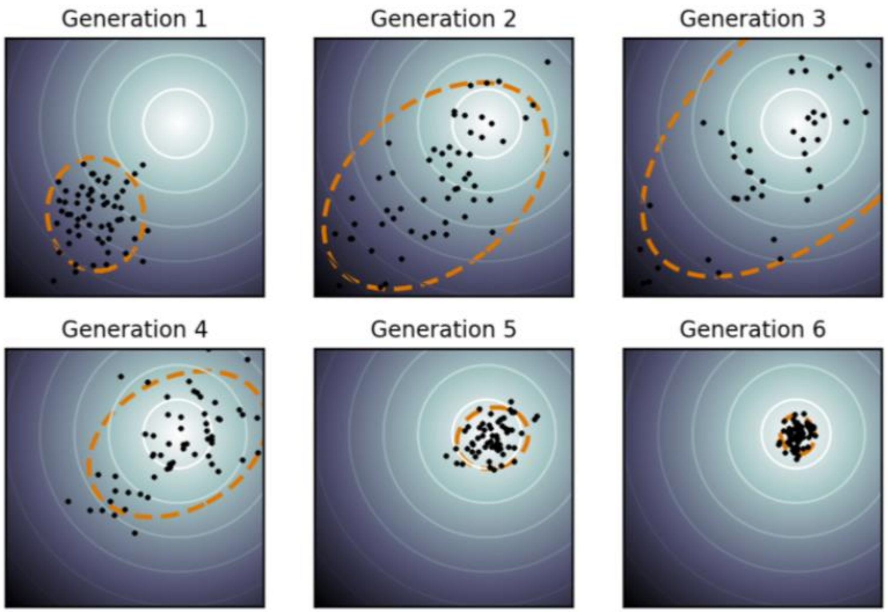
Stochastic sampling-based optimization

> Sampling-based optimization
  - model predictive path integral control (MPPI)
    - sample many action sequences $A_1, \ldots, A_n$ from $p(A) \sim \mathcal{N}(\mu, \Sigma)$
    - evaluate corresponding costs $C_1, \ldots, C_n$
    - compute weights based on costs: $w_i = \frac{\exp\left(-\frac{1}{\lambda} C_i\right)}{\sum_j \exp\left(-\frac{1}{\lambda} C_j\right)}$
    - update $\mu \leftarrow \sum_i w_i A_i$



Stochastic sampling-based optimization

> Sampling-based optimization
  - covariance matrix adaptation evolutionary strategy (CMA-ES)
    - sample many action sequences $A_1, \ldots, A_n$ from $p(A) \sim \mathcal{N}(\mu, \Sigma)$
    - evaluate corresponding costs $C_1, \ldots, C_n$
    - compute weights based on costs: w_i (e.g., exponential, top-K, ...)
    - update $\mu \leftarrow (1-\gamma_\mu)\mu + \gamma_\mu \sum_i w_i A_i, \quad \Sigma \leftarrow (1-\gamma_\Sigma)\Sigma + \gamma_\Sigma \sum_i (A_i - \mu)(A_i - \mu)^\top$



Reference

- https://web.stanford.edu/class/datasci112/lectures/lecture13.pdf
- https://simonblanke.github.io/gradient-free-optimizers-documentation/1.5/
- https://www.deisenroth.cc/teaching/2020-21/ml-seminar/lecture_bayesian_optimization.pdf