CS 581

Advanced Artificial Intelligence

February 19, 2024

Announcements / Reminders

- Please follow the Week 06 To Do List instructions (if you haven't already)
- Written Assignment #02: due on Monday 02/19 at 11:59 PM CST
- Programming Assignment #01: due on Sunday 03/03 at 11:59 PM CST

- Midterm Exam: 02/21/2024
  - Bring REGULAR calculators
  - WE WILL HAVE OUR EXAM IN A DIFFERENT ROOM (WH113)

Plan for Today

- Particle Swarm Optimization
- Gradient Descent
- Review?

Swarm Intelligence

Swarm Intelligence [Wikipedia]

Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence.

SI systems consist typically of a population of simple agents or boids interacting locally with one another and with their environment.

The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents.

Emergence [Wikipedia]

In philosophy, systems theory, science, and art, emergence occurs when a complex entity has properties or behaviors that its parts do not have on their own, and emerge only when they interact in a wider whole.

Emergence plays a central role in theories of integrative levels and of complex systems. For instance, the phenomenon of life as studied in biology is an emergent property of chemistry and quantum physics.

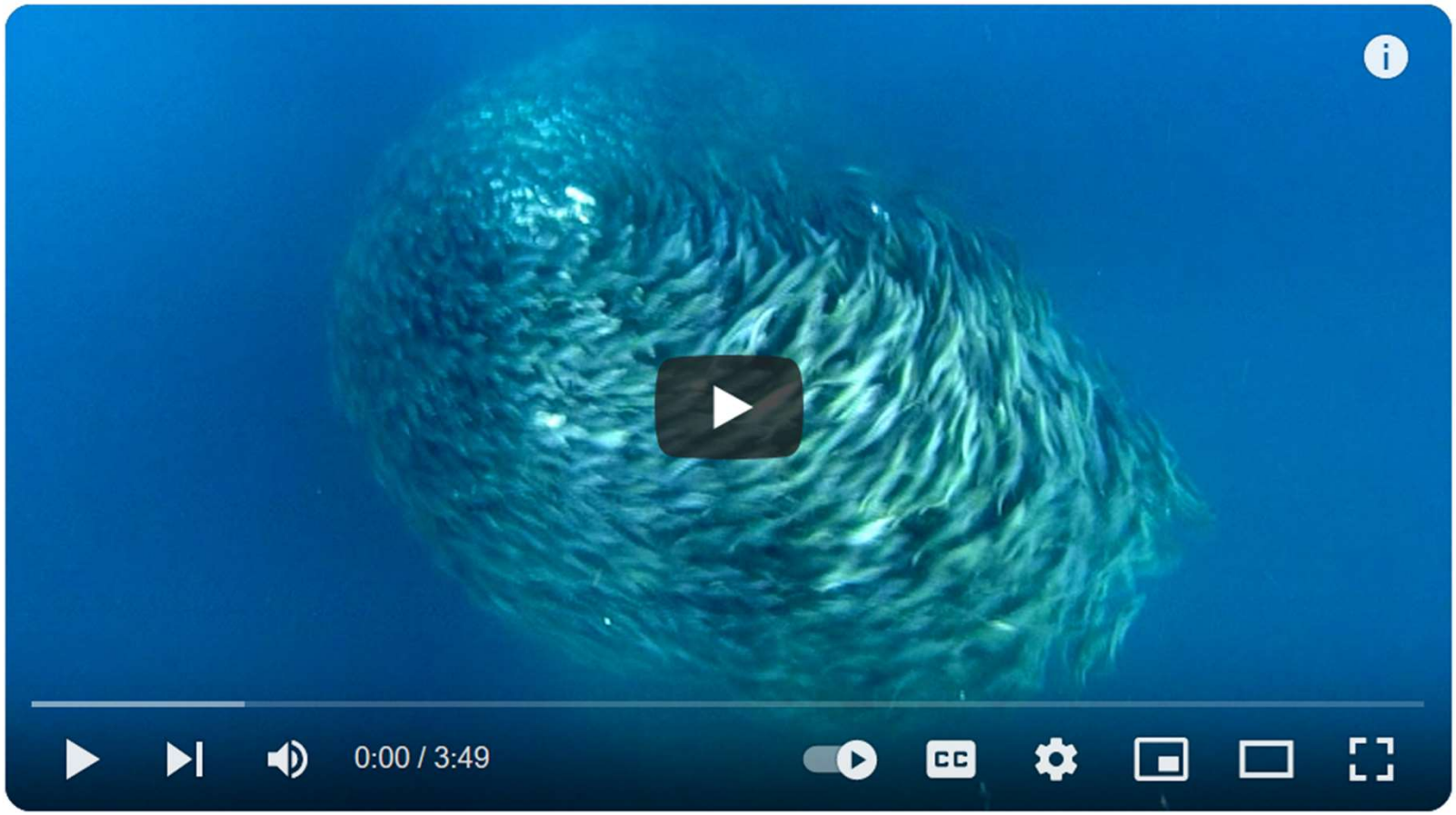
Flocks and Schools

Socio-cognitive Underpinnings

Universal individual behavior principles:

- **Evaluate: tendency to evaluate stimuli (positive/negative, attractive/repulsive) is shared among all kinds of living organisms**
- **Compare: comparing to others is a driver for learning and change**
- **Imitate: effective strategy for learning, though not many living organisms are capable of full imitation**

Flocking and Schooling



Amazing Fish Form Giant Ball to Scare Predators | Blue Planet | BBC Earth

BBC Earth
12.5M subscribers
Subscribe
12K
Share

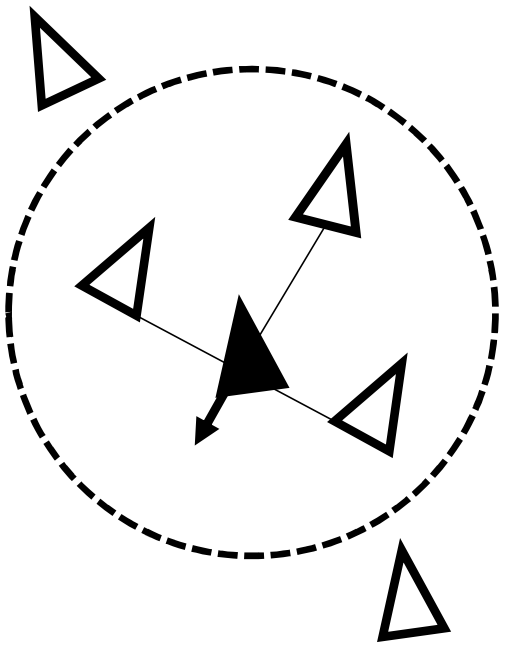
Source: https://www.youtube.com/watch?v=15B8qN9dre4

Flocking and Schooling: Rules

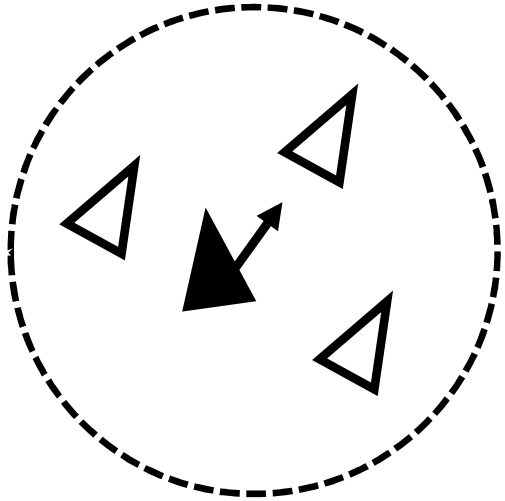
Emergent behavior in flocks and schools can be reduced to simple rules:

- **Separation – an individual should avoid crowding or colliding with its neighbors**
- **Alignment – an individual shoot steer in the average heading of its neighbors**
- **Cohesion – an individual should move toward the average position of its neighbors to maintain the formation of the group**

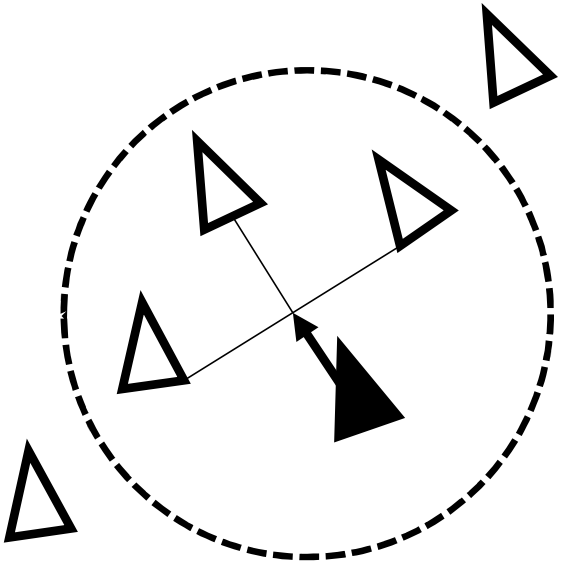
Flocking and Schooling: Rules



Separation



Alignment



Cohesion

Boids

Source: https://www.youtube.com/watch?v=Iv_ZktC865A

Demo: Boids

https://www.harmendeweerd.nl/boids/

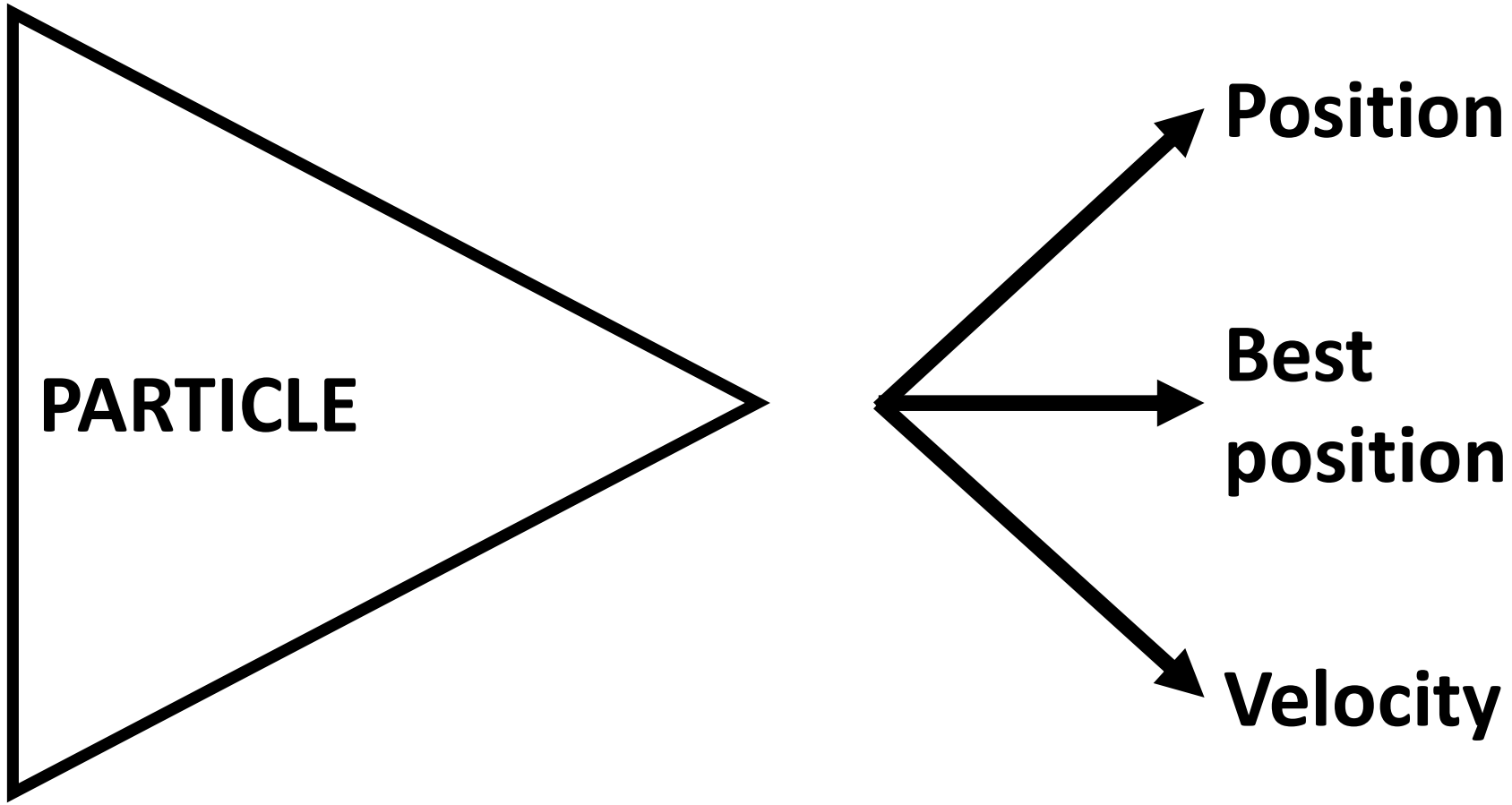
Particle Swarm Optimization (PSO)

Particle Swarm Optimization [WIKI]

In computational science, particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality.

It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formula over the particle's position and velocity.

Particle Properties

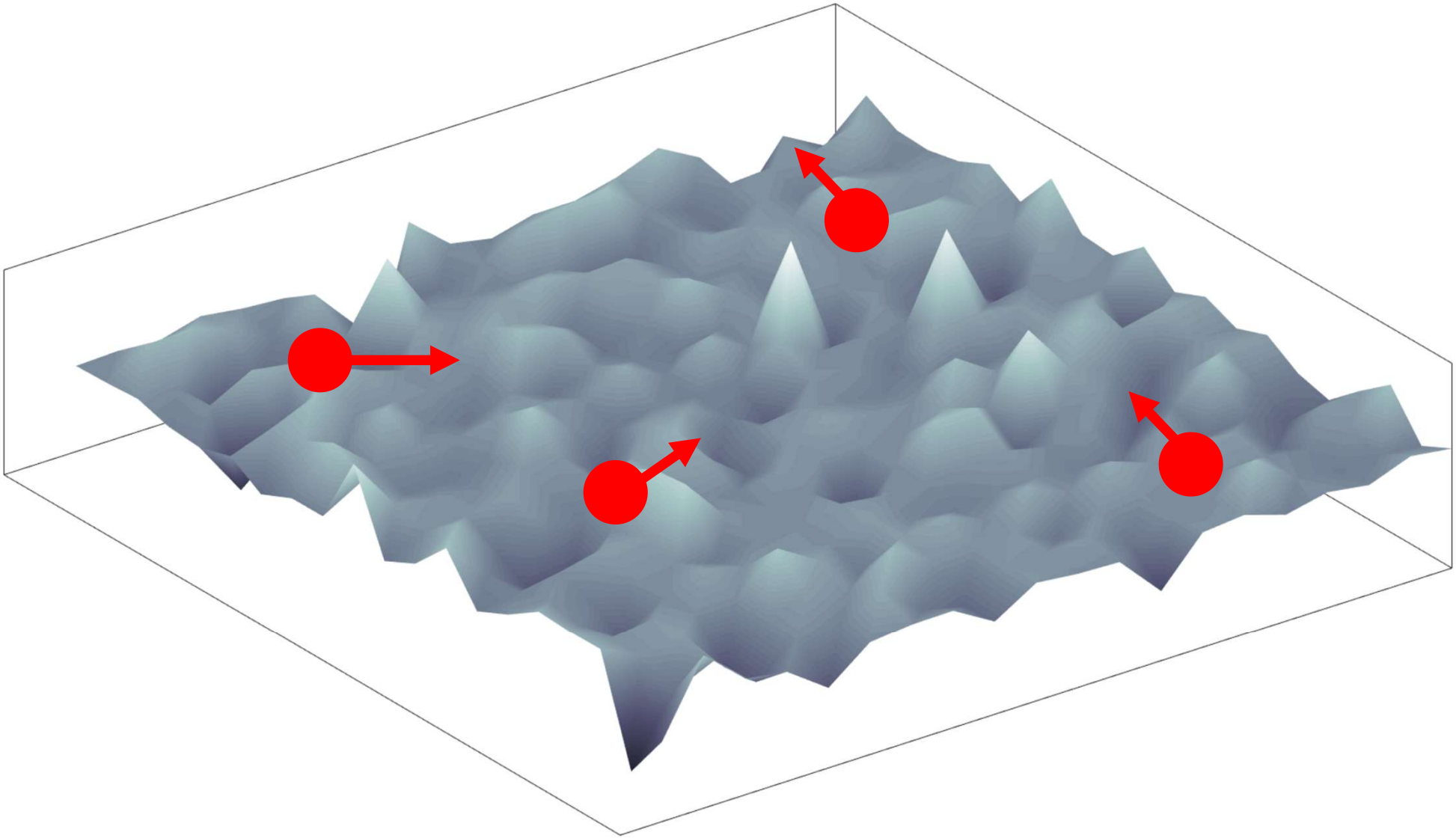


Source: https://www.freepik.com/free-photo/close-up-black-ant_903352.htm Image by onlyyouqj on Freepik

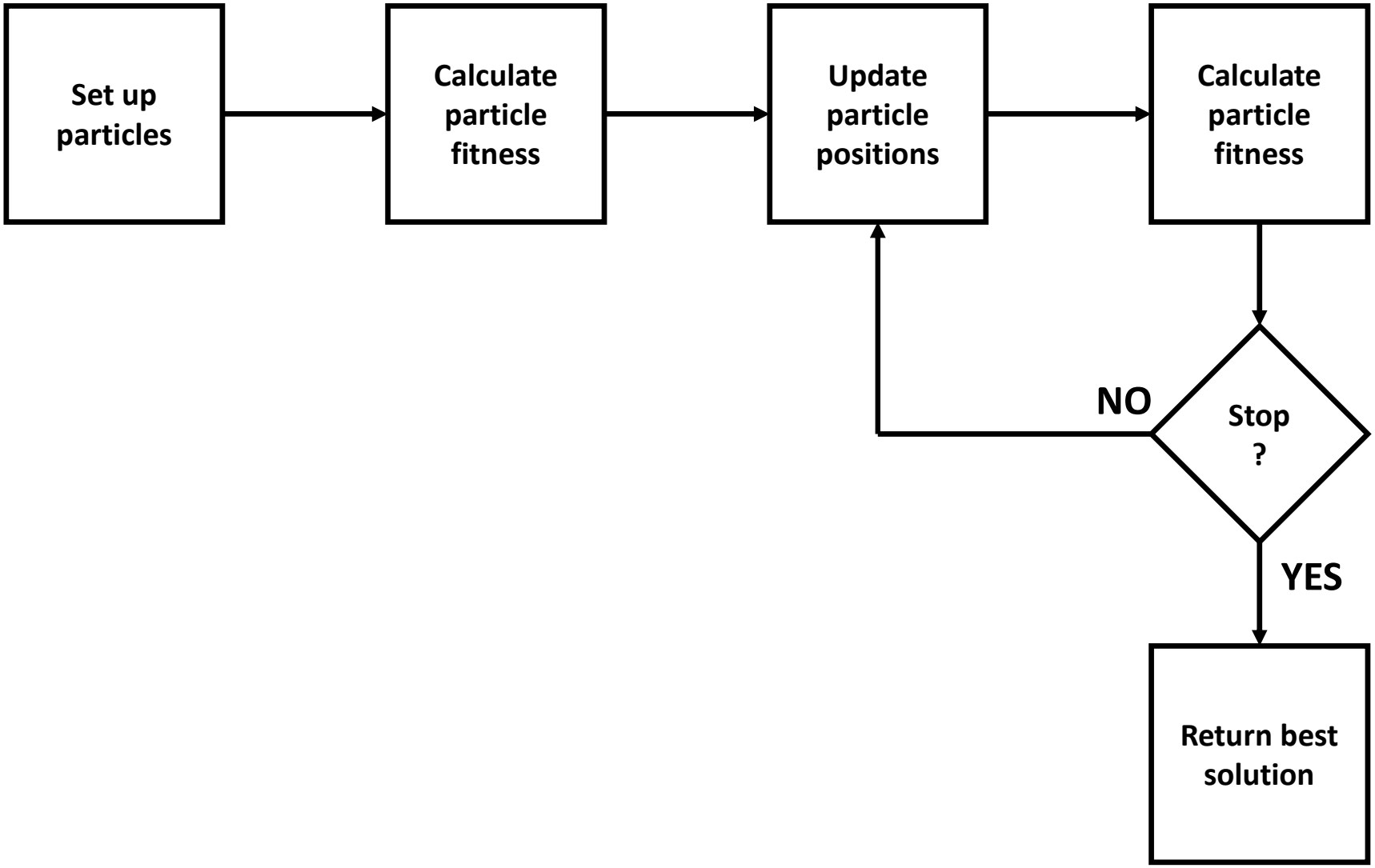
Particle Properties

- Position: the position of the particle in all dimensions
- Best position: the best position found using the fitness function
- Velocity: current velocity of particle's movement

Particles / Environment



Particle Swarm Optimization: Cycle



PSO: Parameters

- **Number of particles** K
- c_1 **significance of personal particle experience (trust in individual knowledge)**
- c_2 **significance of swarm experience (trust in social knowledge)**
- **Inertia weight** w
- V_{max} **velocity cap**
- N_i **neighborhood of particle** i

PSO: Algorithm

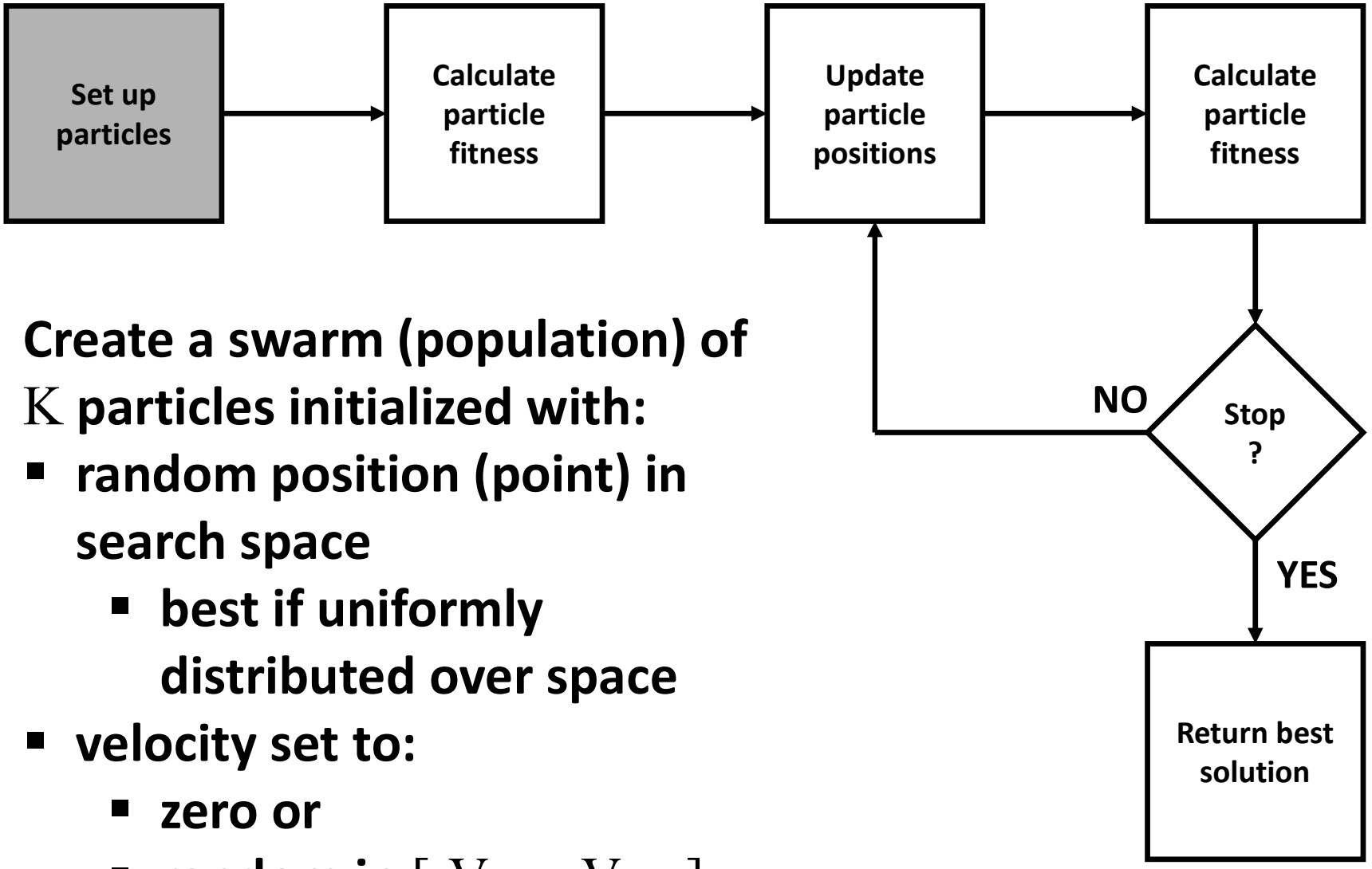
Input: Randomly initialized position and velocity of Particles:

X_i^0 and V_i^0

Output: Position of the approximate global minimum X^*

1: **while** termination condition is not reached do
2: **for** i = 1 to K **do**
3: Evaluate particles using the fitness function f()
4: Update personal best and global best of each particle
5: Update velocity of each particle
6: Update the position of each particle
7: **end for**
8: **end while**

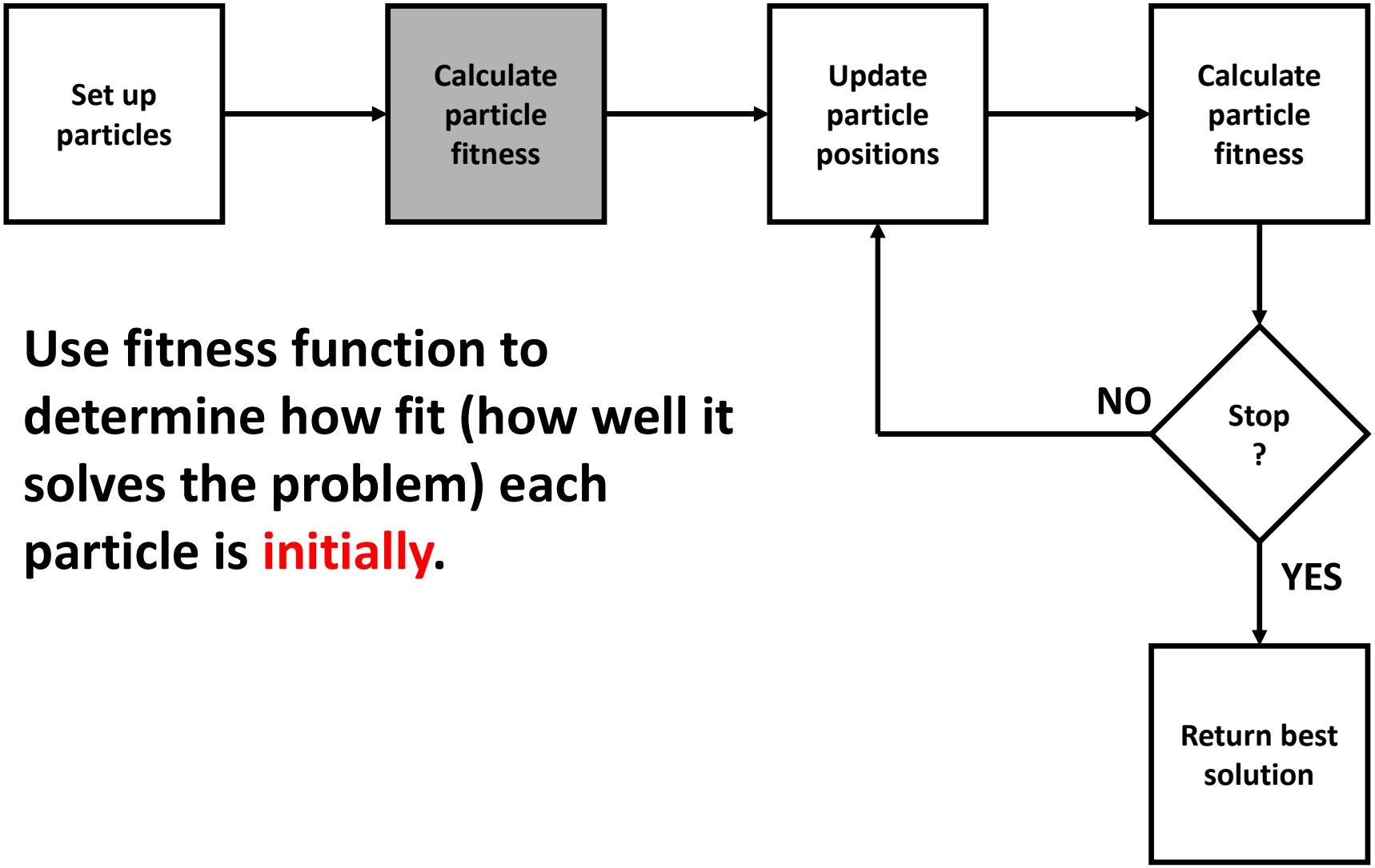
Particle Swarm Optimization: Cycle



Create a swarm (population) of K particles initialized with:

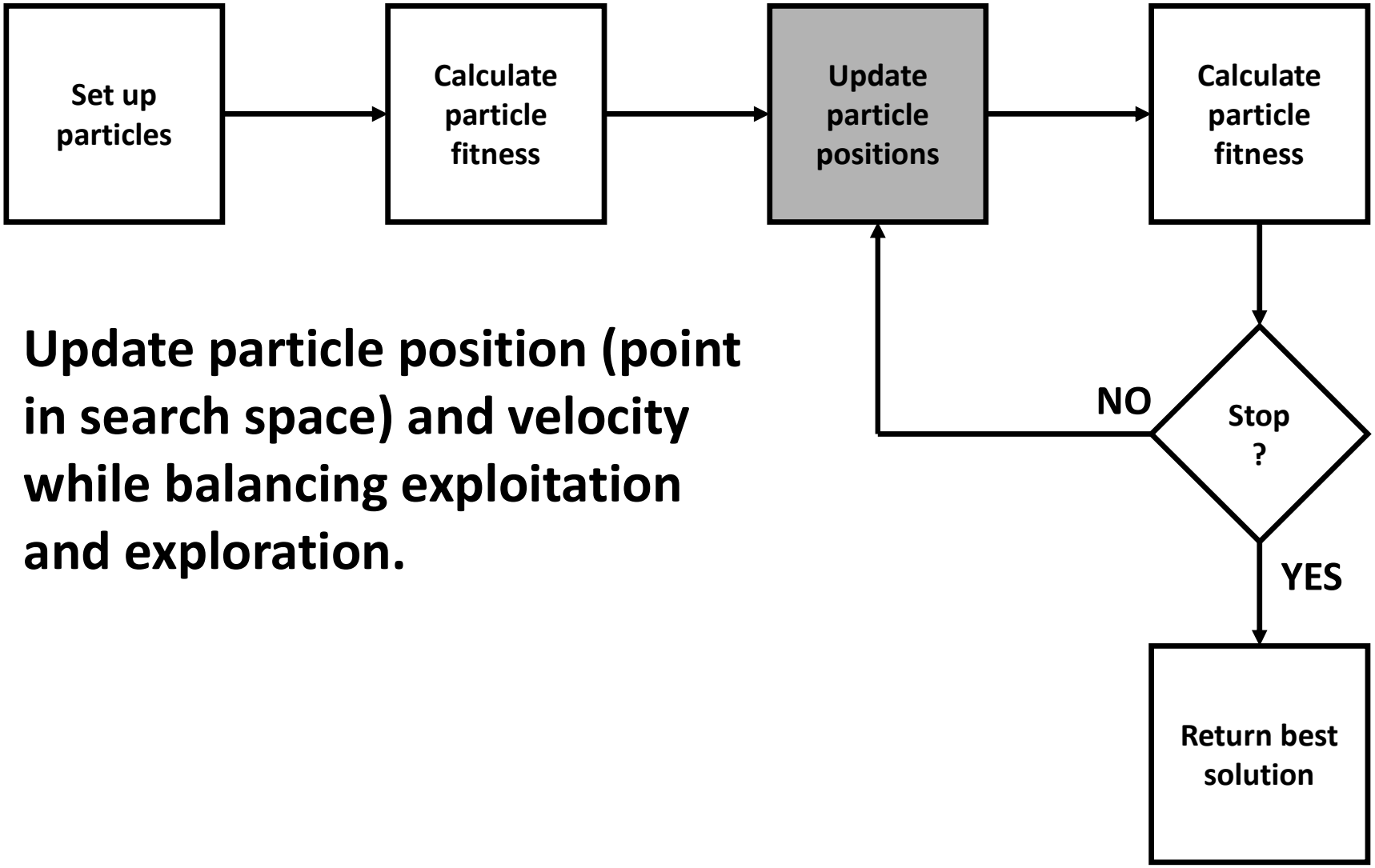
- **random position (point) in search space**
  - **best if uniformly distributed over space**
- **velocity set to:**
  - **zero or**
  - **random in $[-V_{max}, V_{max}]$**

Particle Swarm Optimization: Cycle



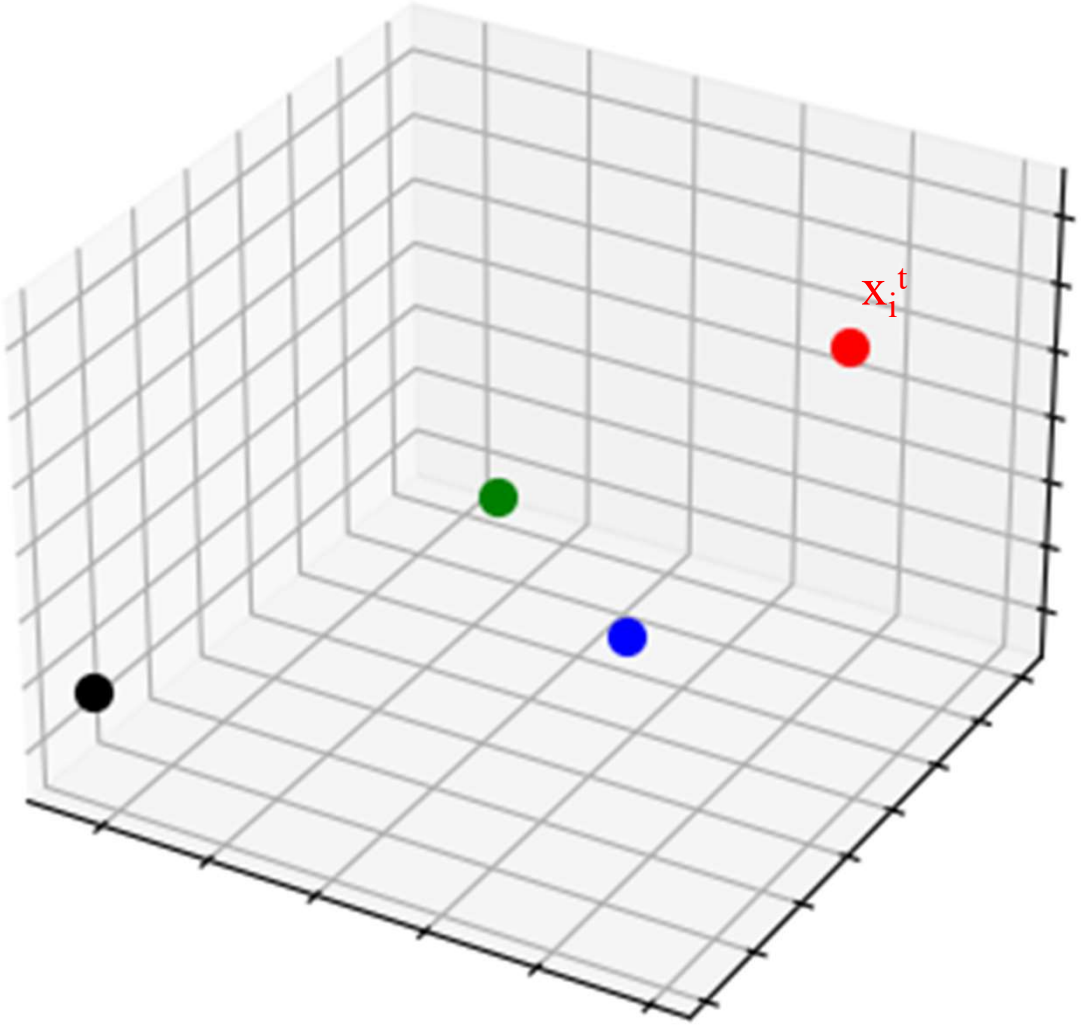
Use fitness function to determine how fit (how well it solves the problem) each particle is initially.

Particle Swarm Optimization: Cycle

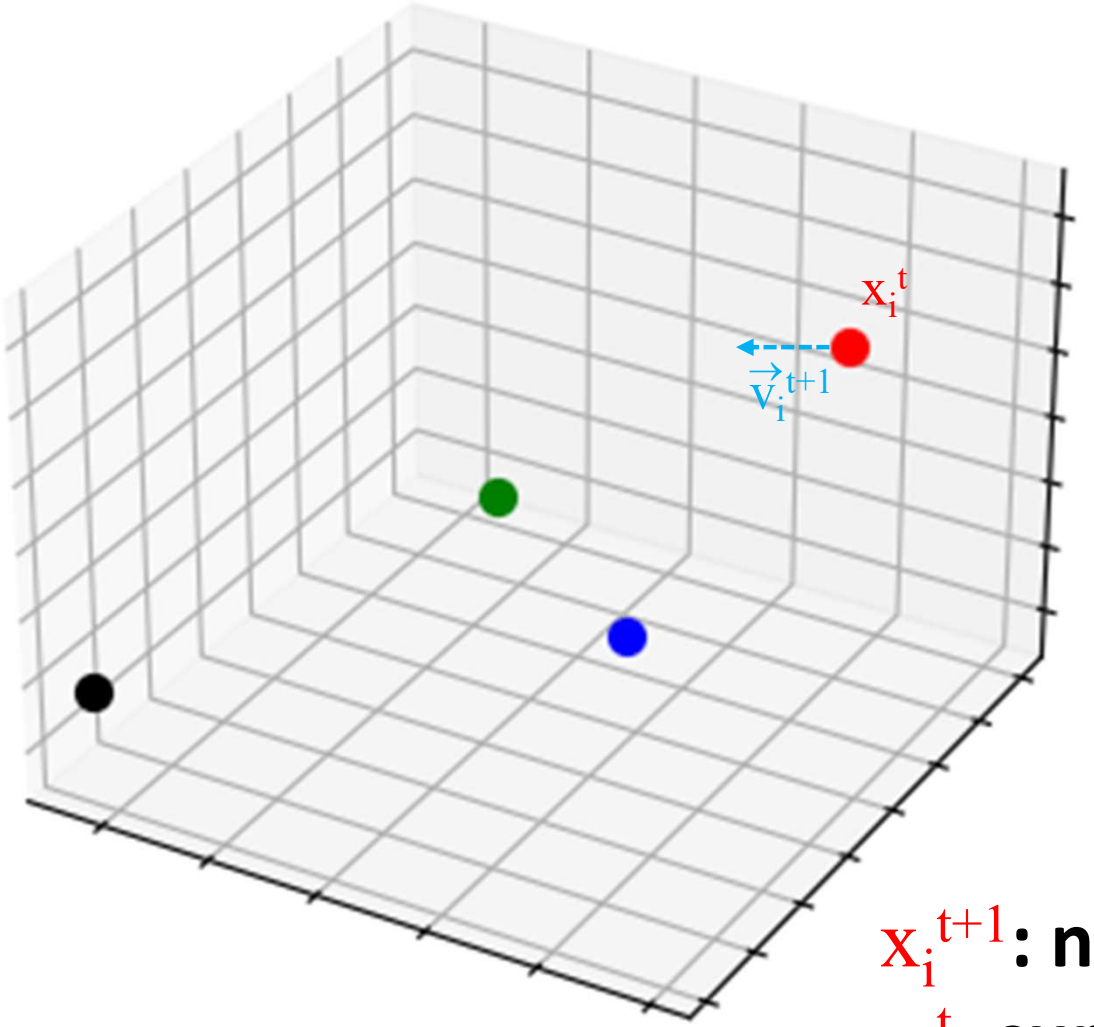


Update particle position (point in search space) and velocity while balancing exploitation and exploration.

Particle Position Update



Particle Position Update



Next position:

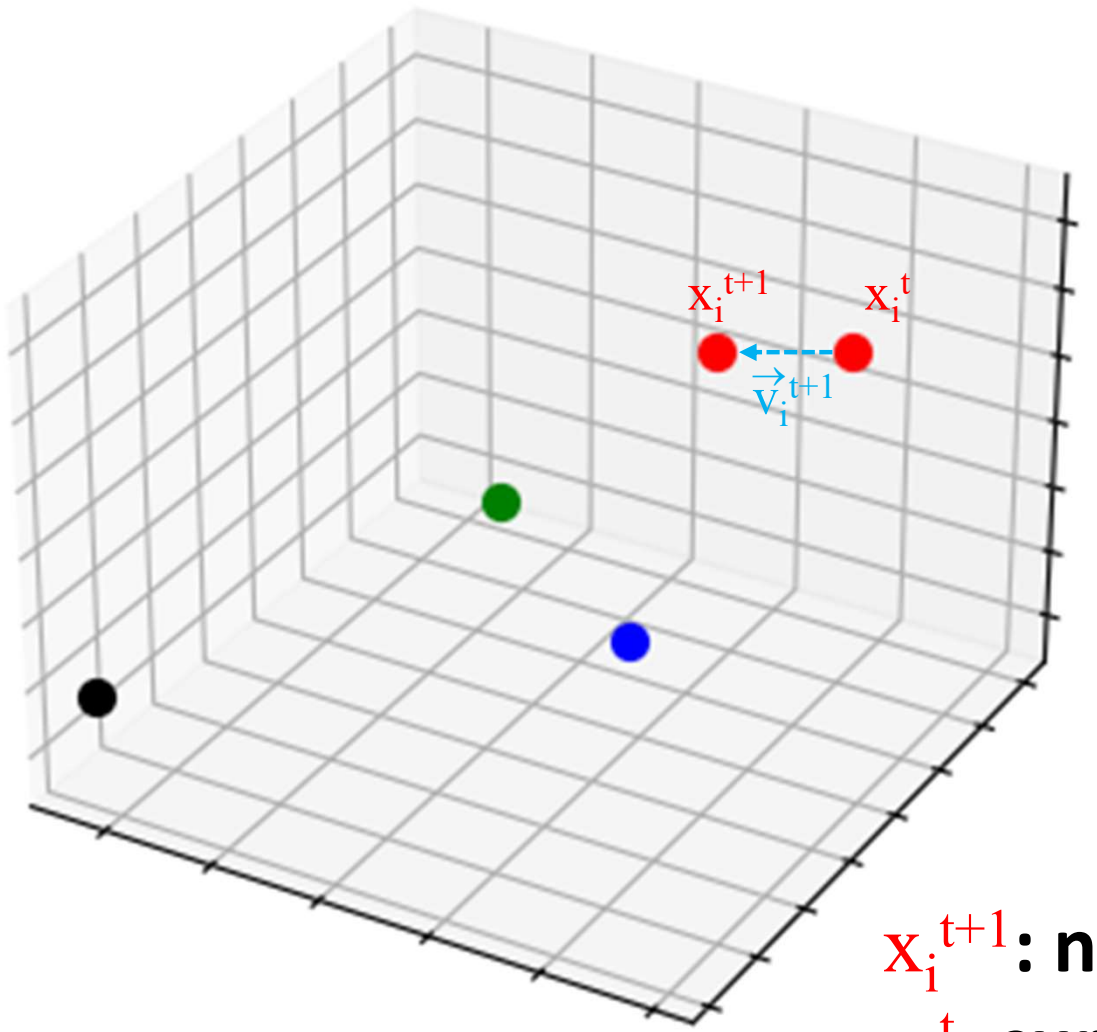
$$x_i^{t+1} = x_i^t + \vec{v}_i^{t+1}$$

x_i^{t+1}: **next position**

x_i^t: **current position**

$\vec{v}_i^{t+1}$: **next velocity**

Particle Position Update



Next position:

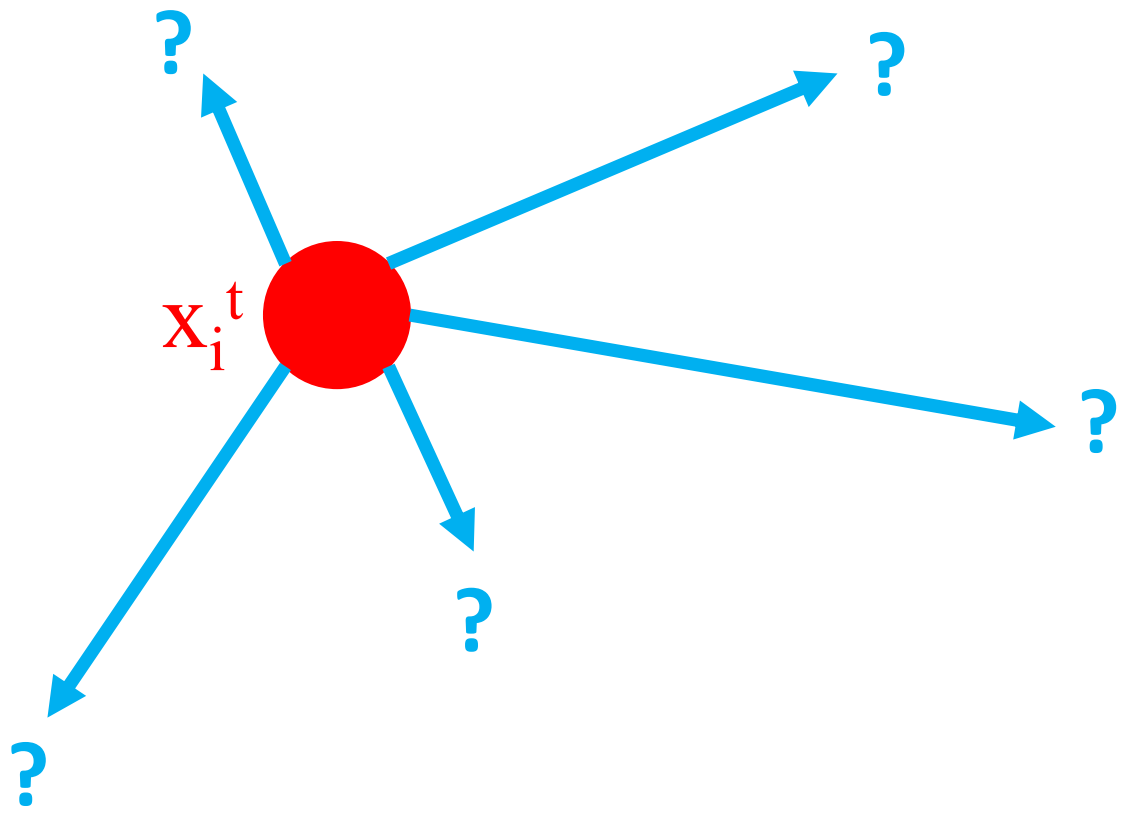
$$x_i^{t+1} = x_i^{t} + \vec{v}_i^{t+1}$$

x_i^{t+1}: **next position**

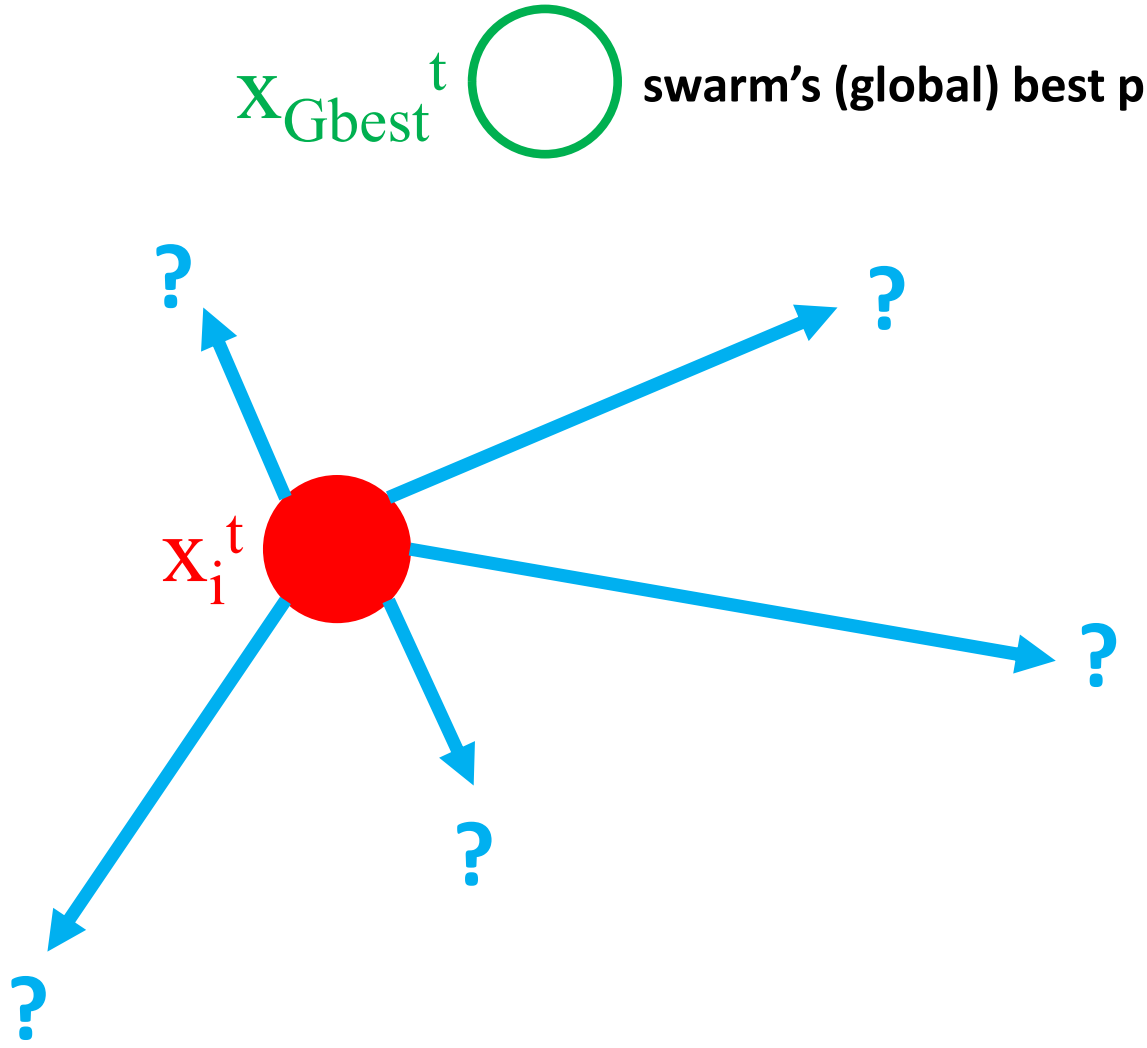
x_i^{t}: **current position**

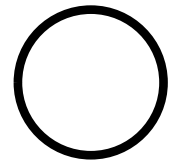
$\vec{v}_i^{t+1}$: **next velocity**

Particle Position: Where Next?



Particle Position: Where Next?

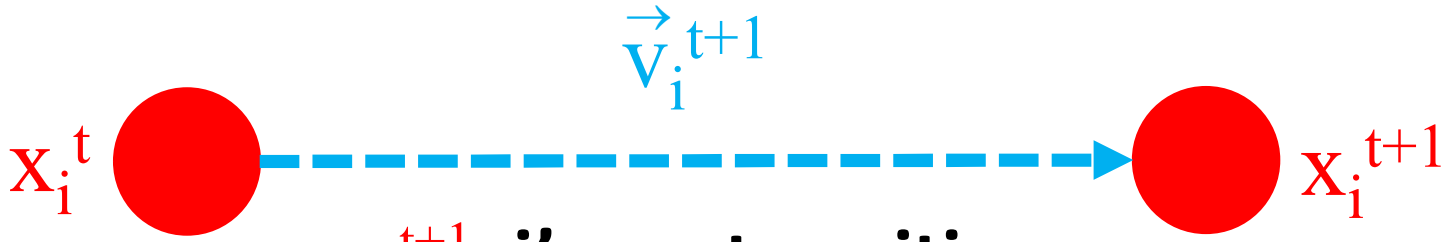




Particle i's best position

x_{ibest}^{t}

Particle Position Update



$x_i^{\,t+1}$: i's next position

$x_i^{\,t}$: i's current position

$\vec{v}_i^{\,t+1}$: i's next velocity

Next position:

$$x_i^{\,t+1} = x_i^{\,t} + \vec{v}_i^{\,t+1}$$

Next velocity:

$$\vec{v}_i^{\,t+1} = w * \vec{v}_i^{\,t} + c_1 * a * (x_{ibest}^{\,t} - x_i^{\,t}) + c_2 * b * (x_{Gbest}^{\,t} - x_i^{\,t})$$

$x_{ibest}^{\,t}$: i's best position

$x_{Gbest}^{\,t}$: swarm's best position

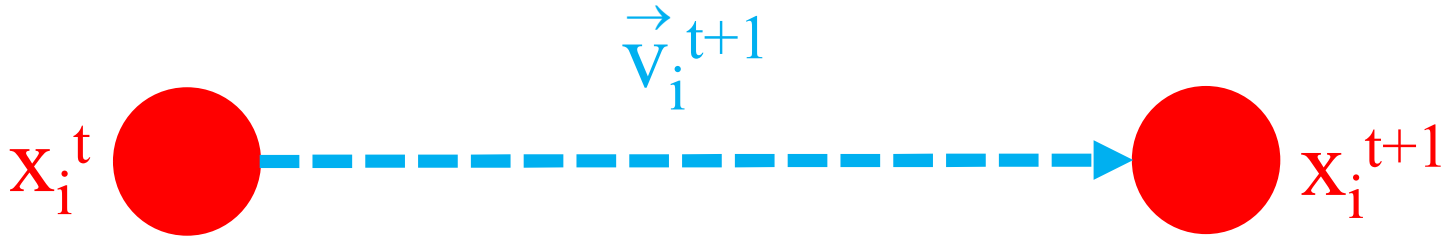
$\vec{v}_i^{\,t}$: i's current velocity

c_1: cognitive constant

c_2: social constant

w: inertia weight

Particle Position Update



Next position:

$$x_i^{\,t+1} = x_i^{\,t} + \vec{v}_i^{\,t+1}$$

a: **random number in [0;1]**

b: **random number in [0;1]**

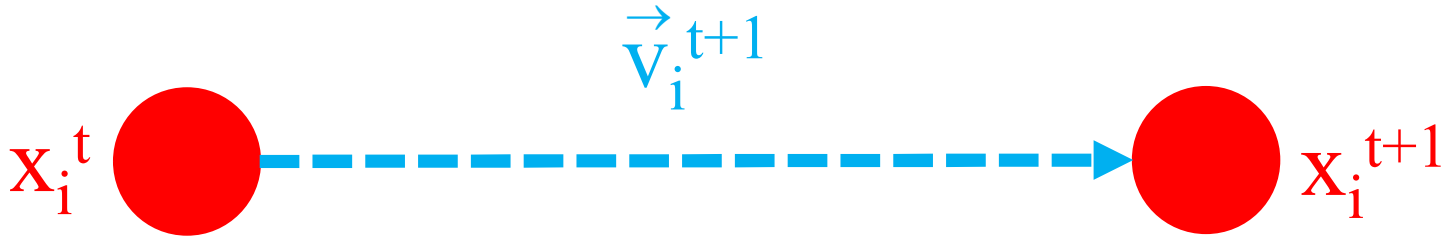
Next velocity:

$$\vec{v}_i^{\,t+1} = w * \vec{v}_i^{\,t} + c_1 * a * (x_{ibest}^{\,t} - x_i^{\,t}) + c_2 * b * (x_{Gbest}^{\,t} - x_i^{\,t})$$

Also common:

$$\vec{v}_i^{\,t+1} = \vec{v}_i^{\,t} + c_1 * a * (x_{ibest}^{\,t} - x_i^{\,t}) + c_2 * b * (x_{Gbest}^{\,t} - x_i^{\,t})$$

Particle Position Update



Next position:

$$x_i^{\,t+1} = x_i^{\,t} + \vec{v}_i^{\,t+1}$$

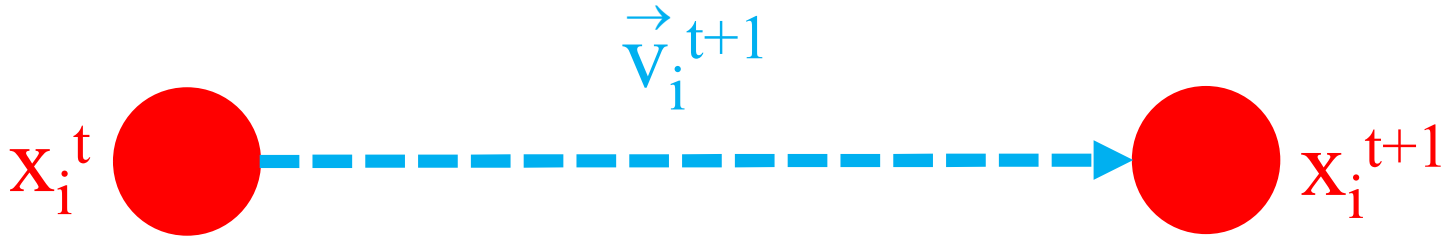
Next velocity:

$$\vec{v}_i^{\,t+1} = w * \vec{v}_i^{\,t} + c_1 * a * (x_{ibest}^{\,t} - x_i^{\,t}) + c_2 * b * (x_{Gbest}^{\,t} - x_i^{\,t})$$

$$\vec{v}_i^{\,t+1} = \text{inertia} + \text{cognitive component} + \text{social component}$$

$$\vec{v}_i^{\,t+1} = \text{inertia} + \text{cognitive influence} + \text{social influence}$$

Particle Position Update



Next position:

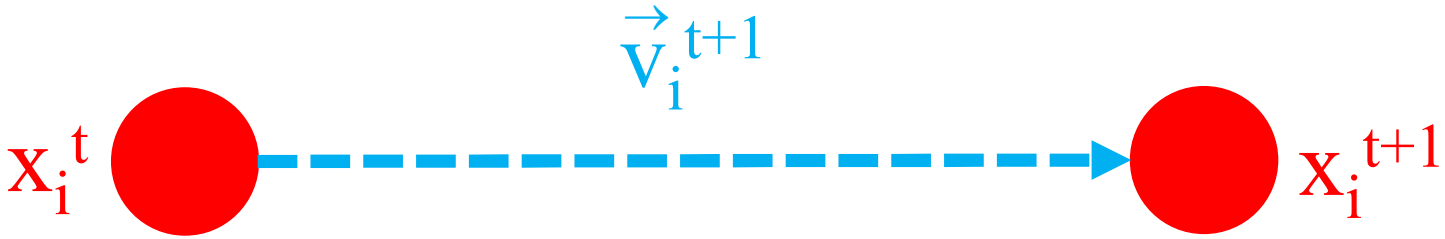
$$x_i^{\,t+1} = x_i^{\,t} + \vec{v}_i^{\,t+1}$$

Next velocity:

$$\vec{v}_i^{\,t+1} = w * \vec{v}_i^{\,t} + c_1 * a * (x_{ibest}^{\,t} - x_i^{\,t}) + c_2 * b * (x_{Gbest}^{\,t} - x_i^{\,t})$$

Vectors

Particle Position Update



Next position:

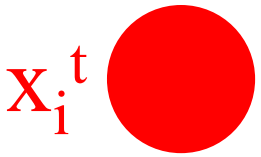
$$x_i^{\,t+1} = x_i^{\,t} + \vec{v}_i^{\,t+1}$$

Next velocity:

$$\vec{v}_i^{\,t+1} = w * \vec{v}_i^{\,t} + c_1 * a * (x_{ibest}^{\,t} - x_i^{\,t}) + c_2 * b * (x_{Gbest}^{\,t} - x_i^{\,t})$$

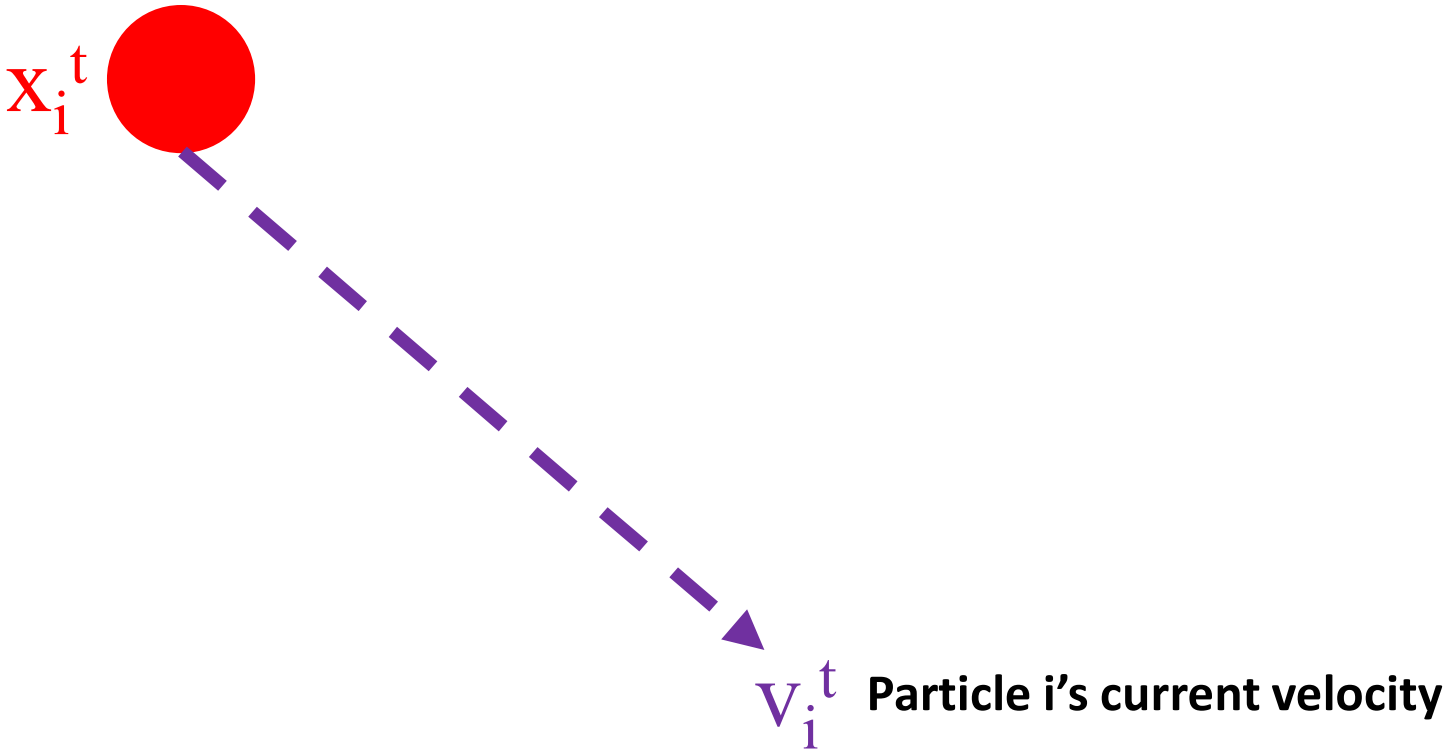
Scalars

Particle Position Update

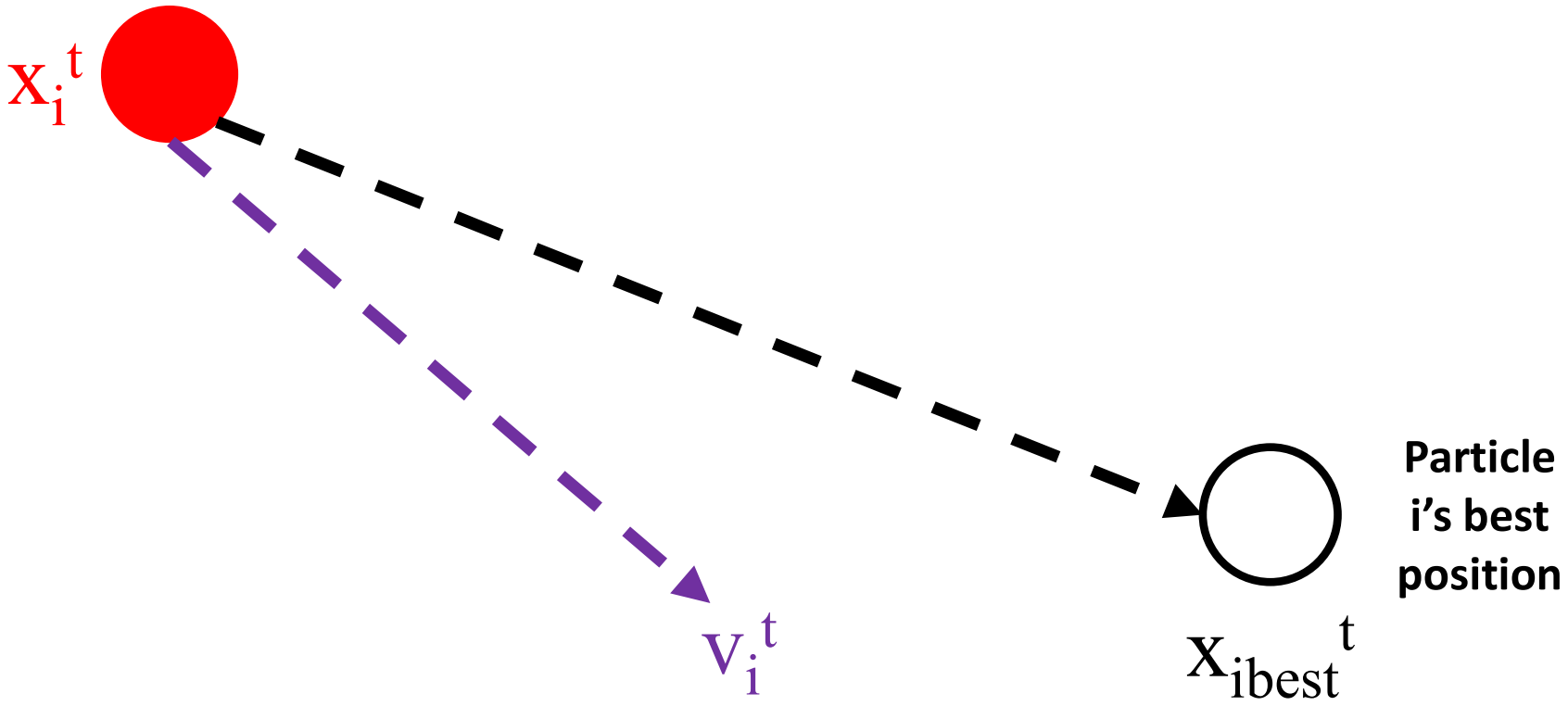


Particle i's current position

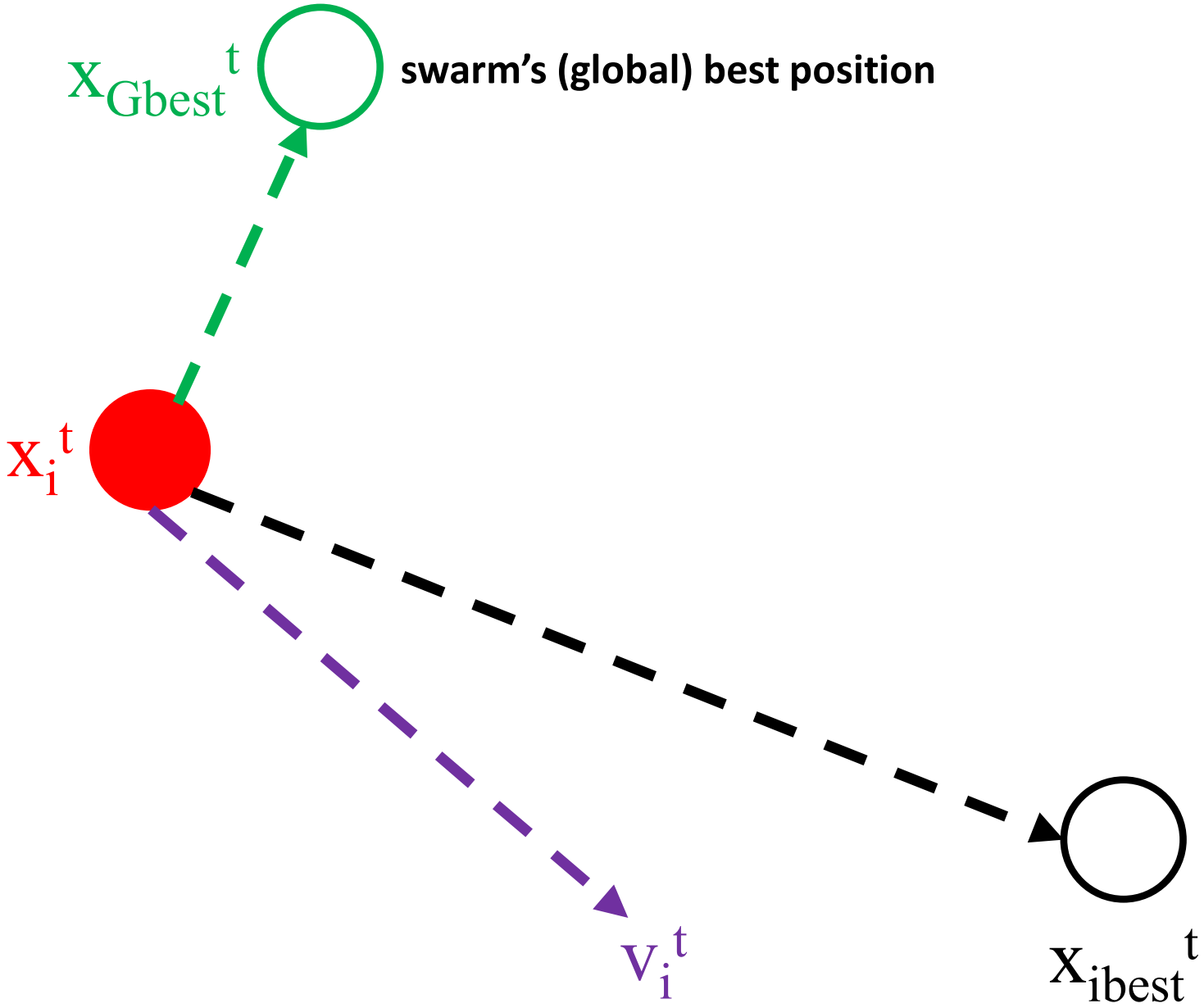
Particle Position Update



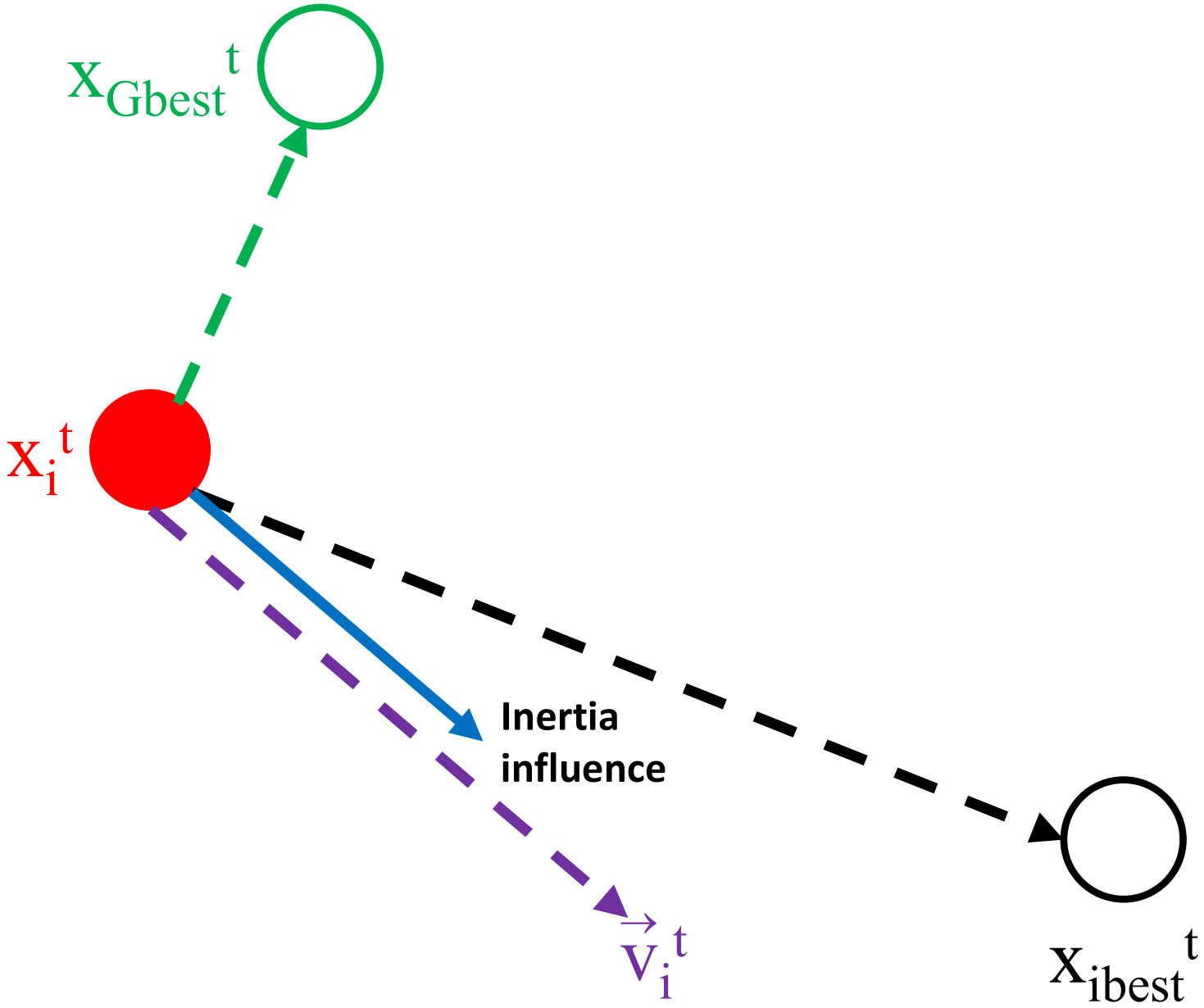
Particle Position Update



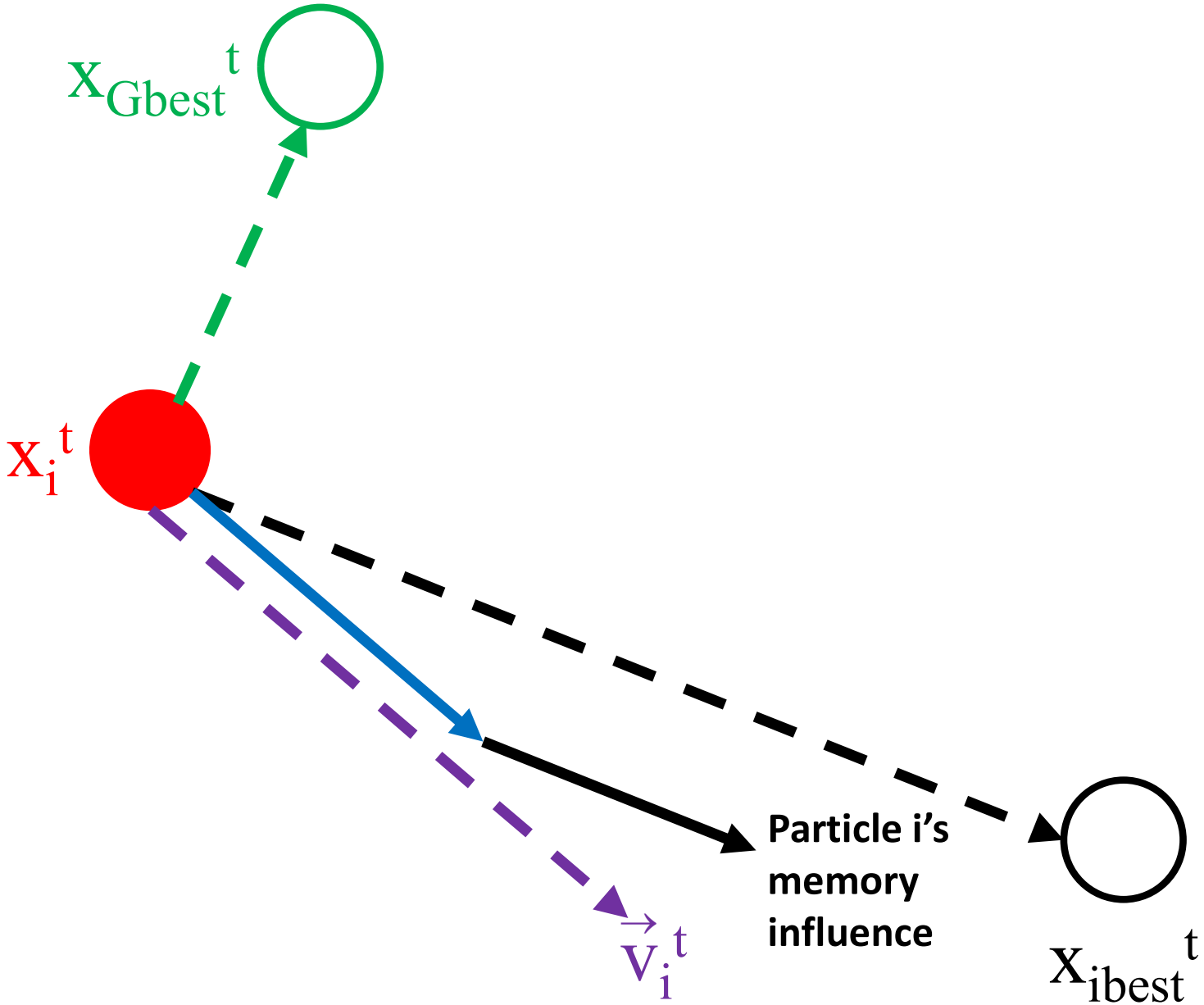
Particle Position Update



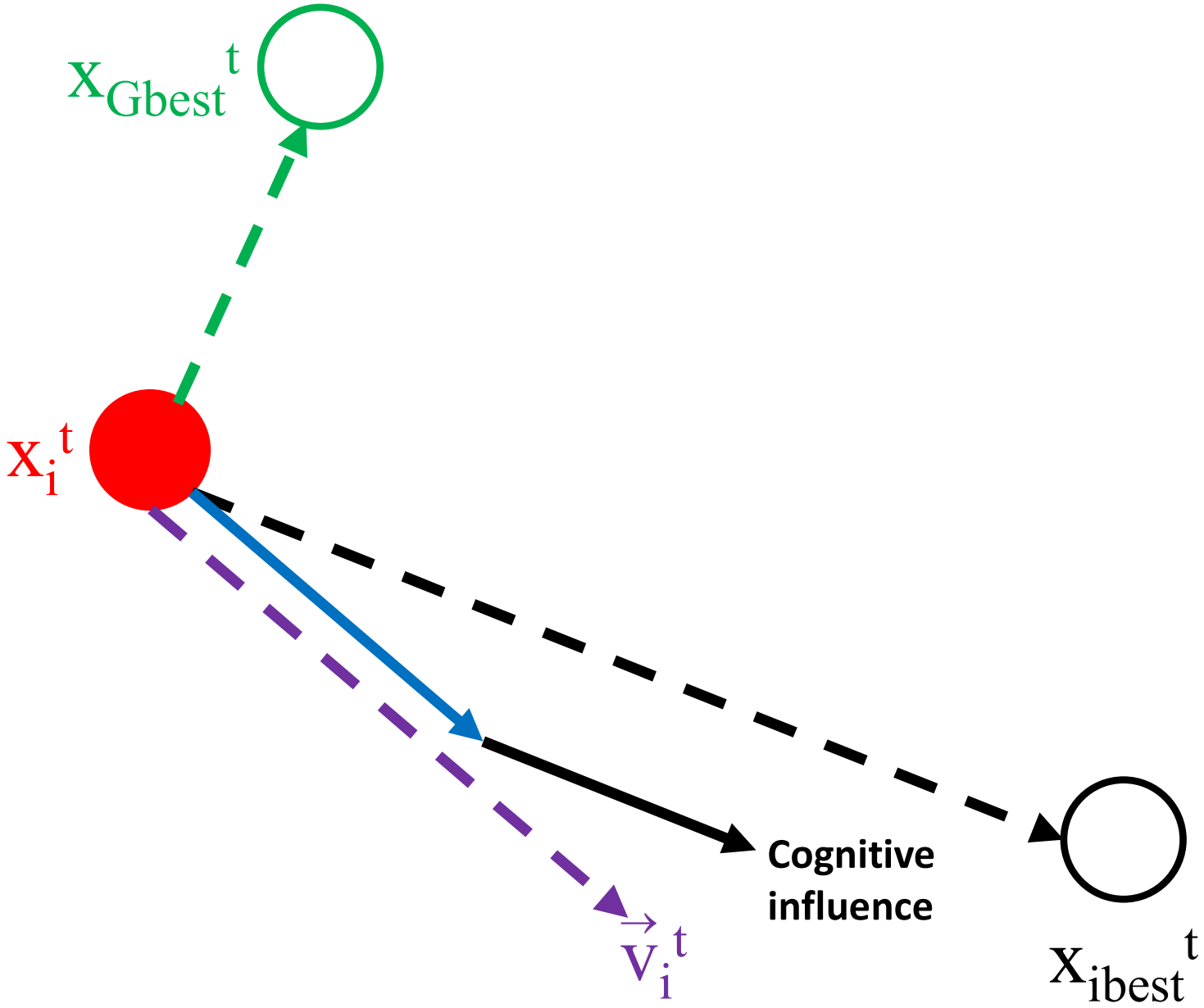
Particle Position Update



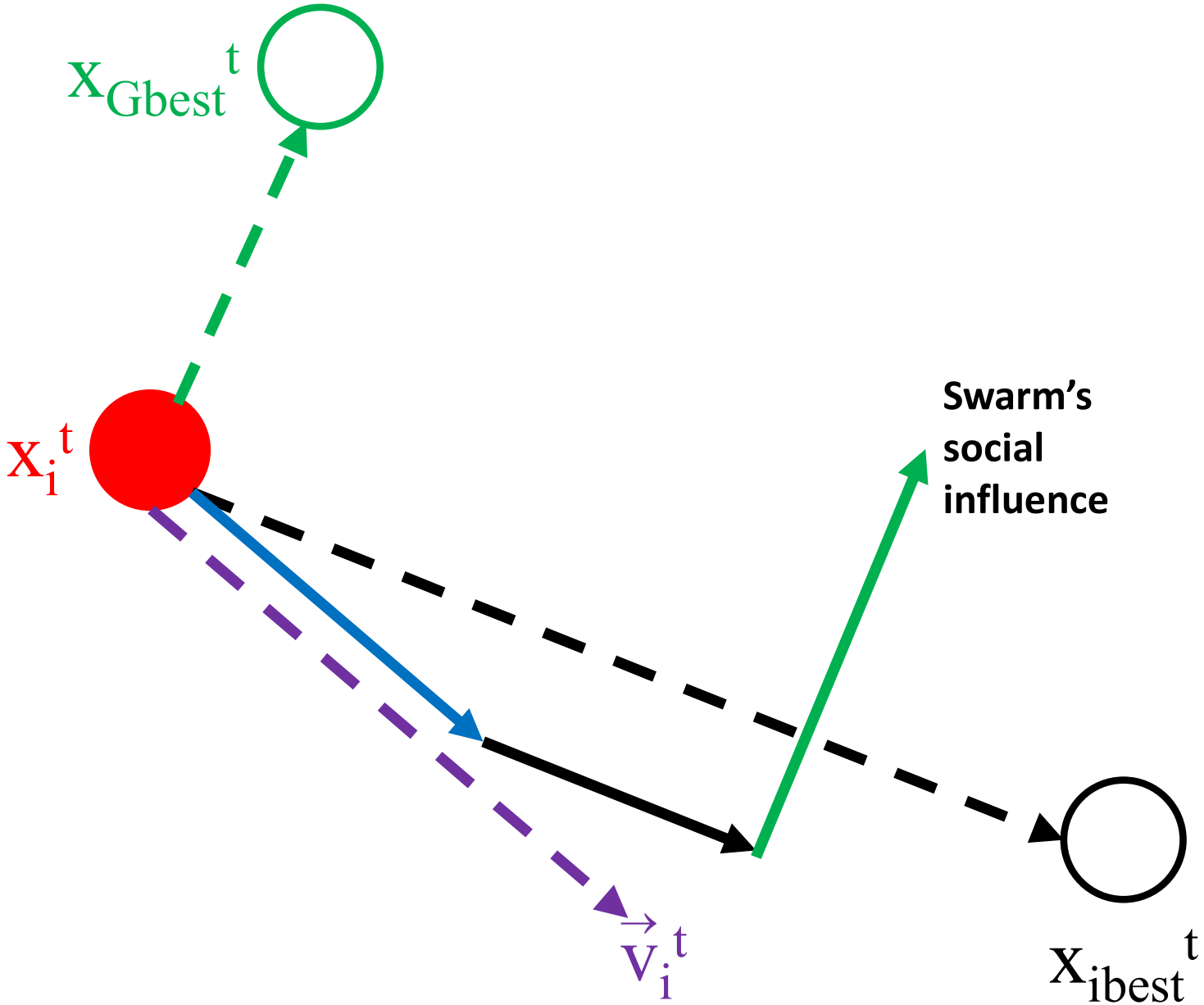
Particle Position Update



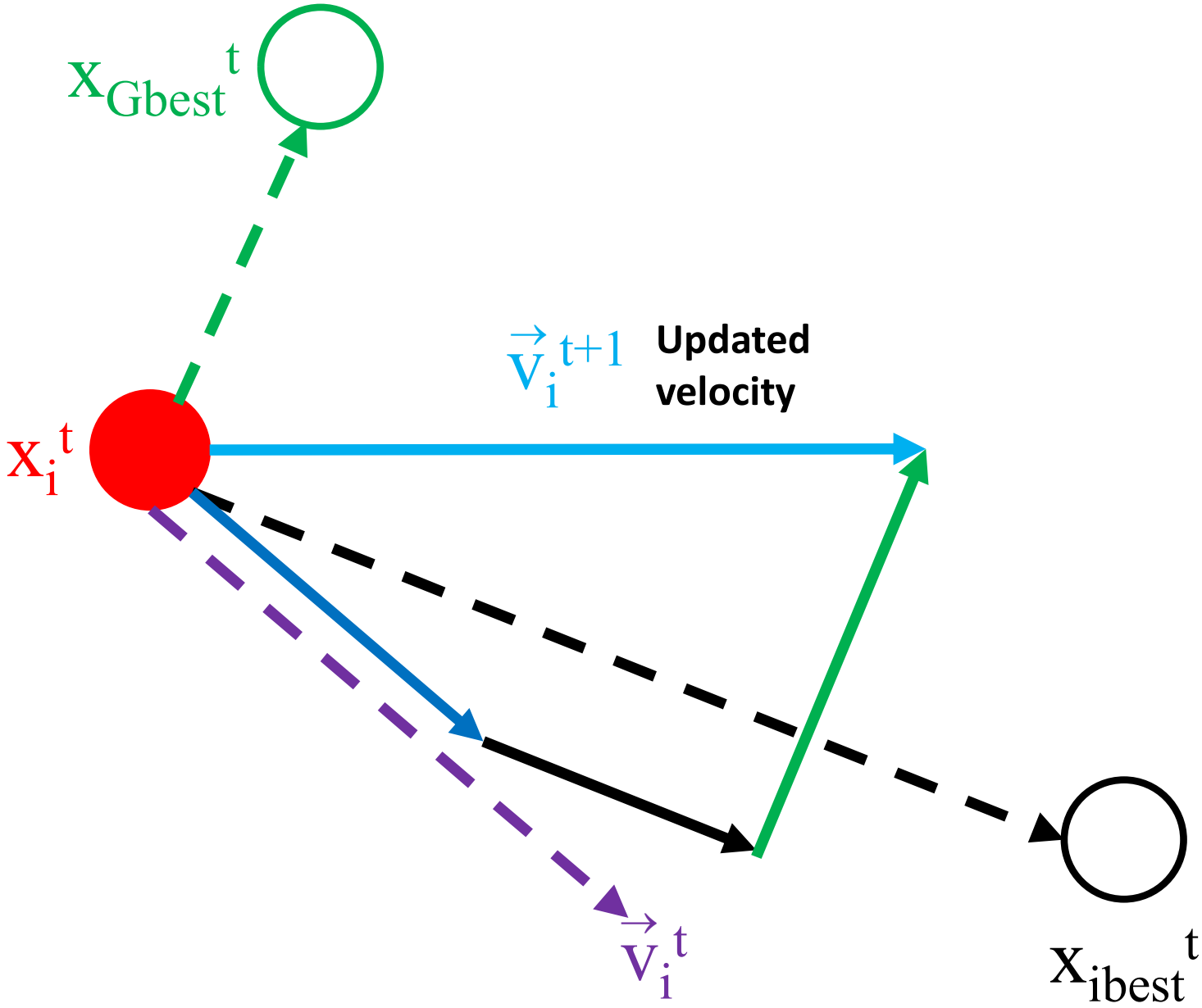
Particle Position Update



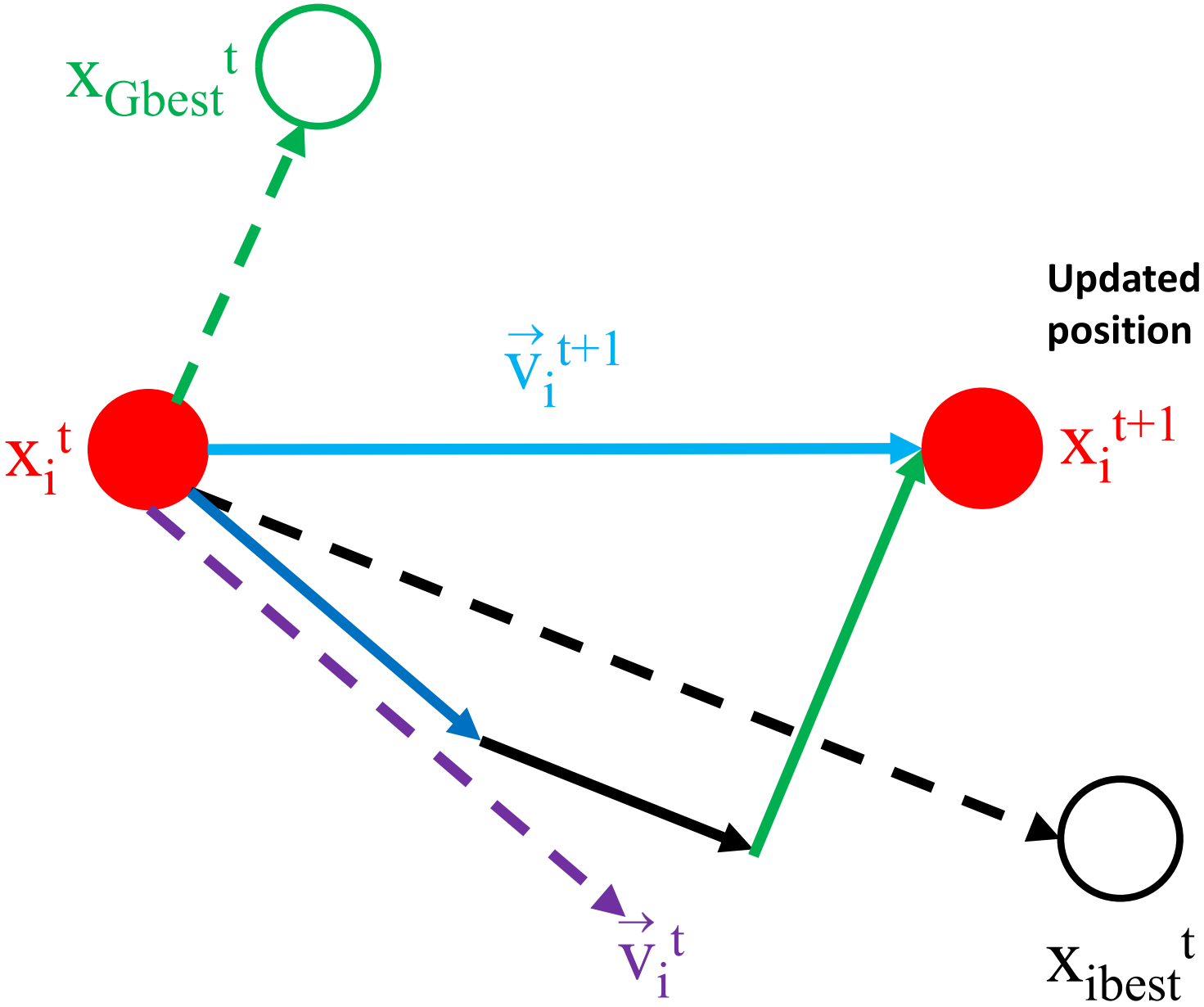
Particle Position Update



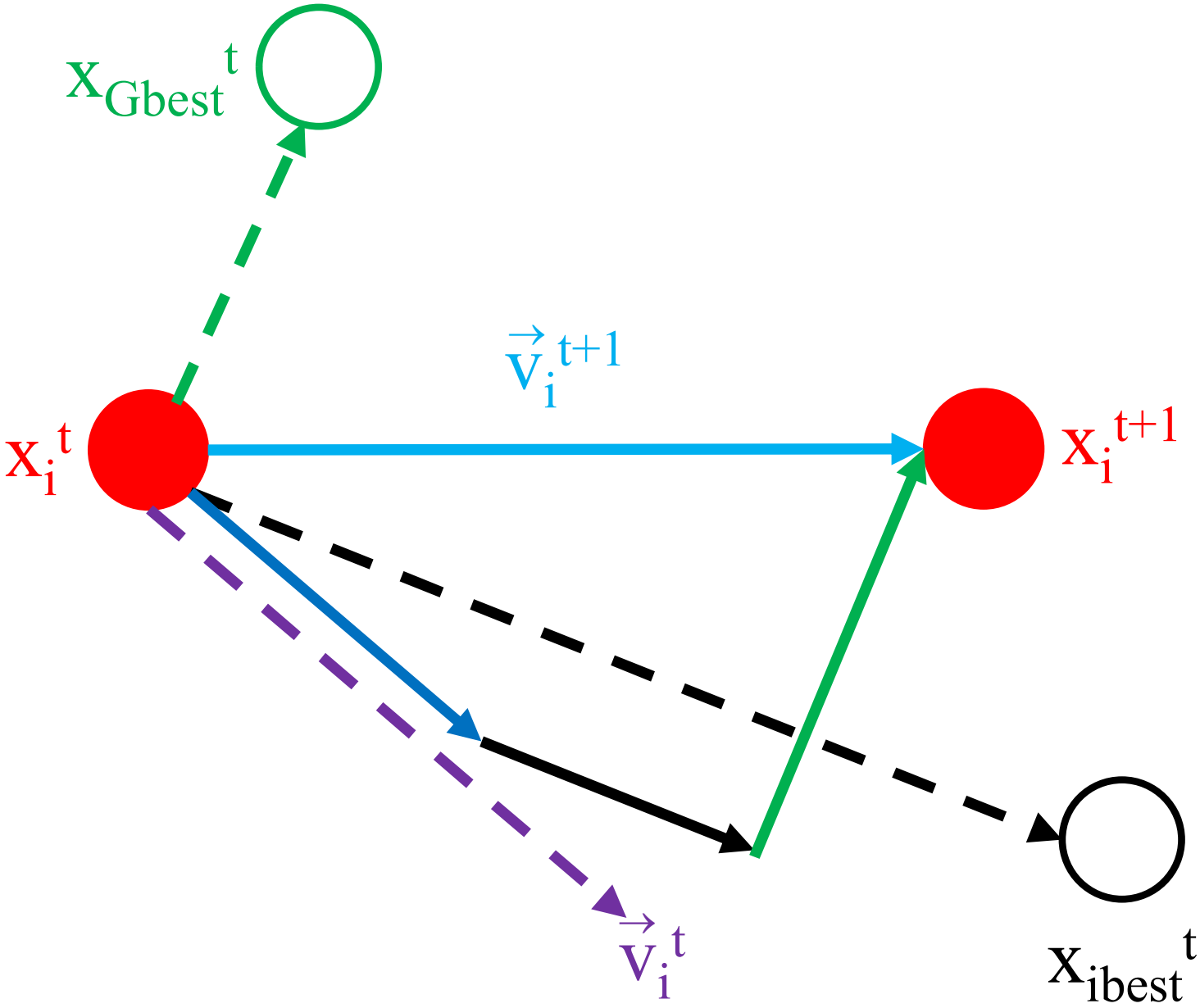
Particle Position Update



Particle Position Update

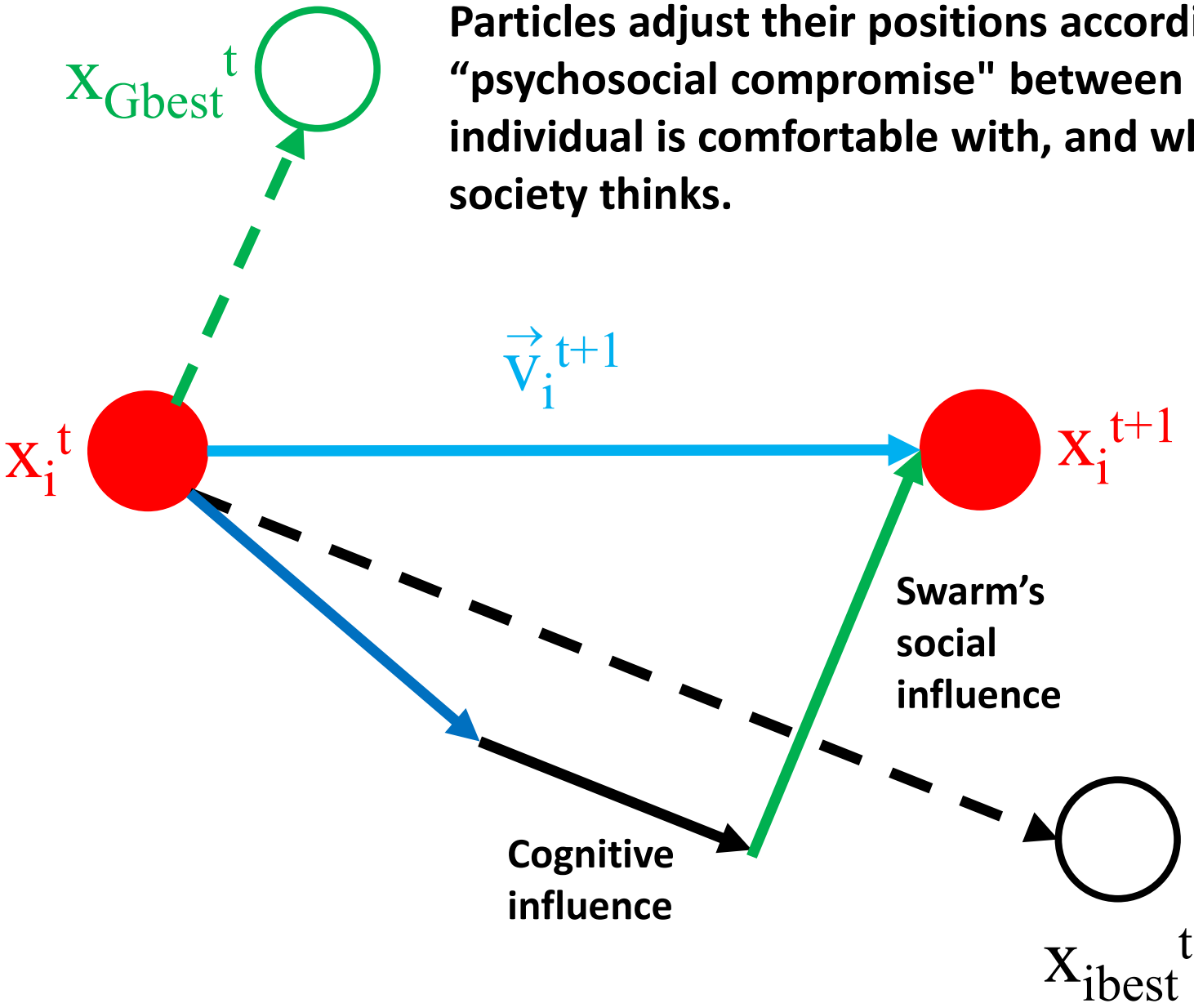


Particle Position Update

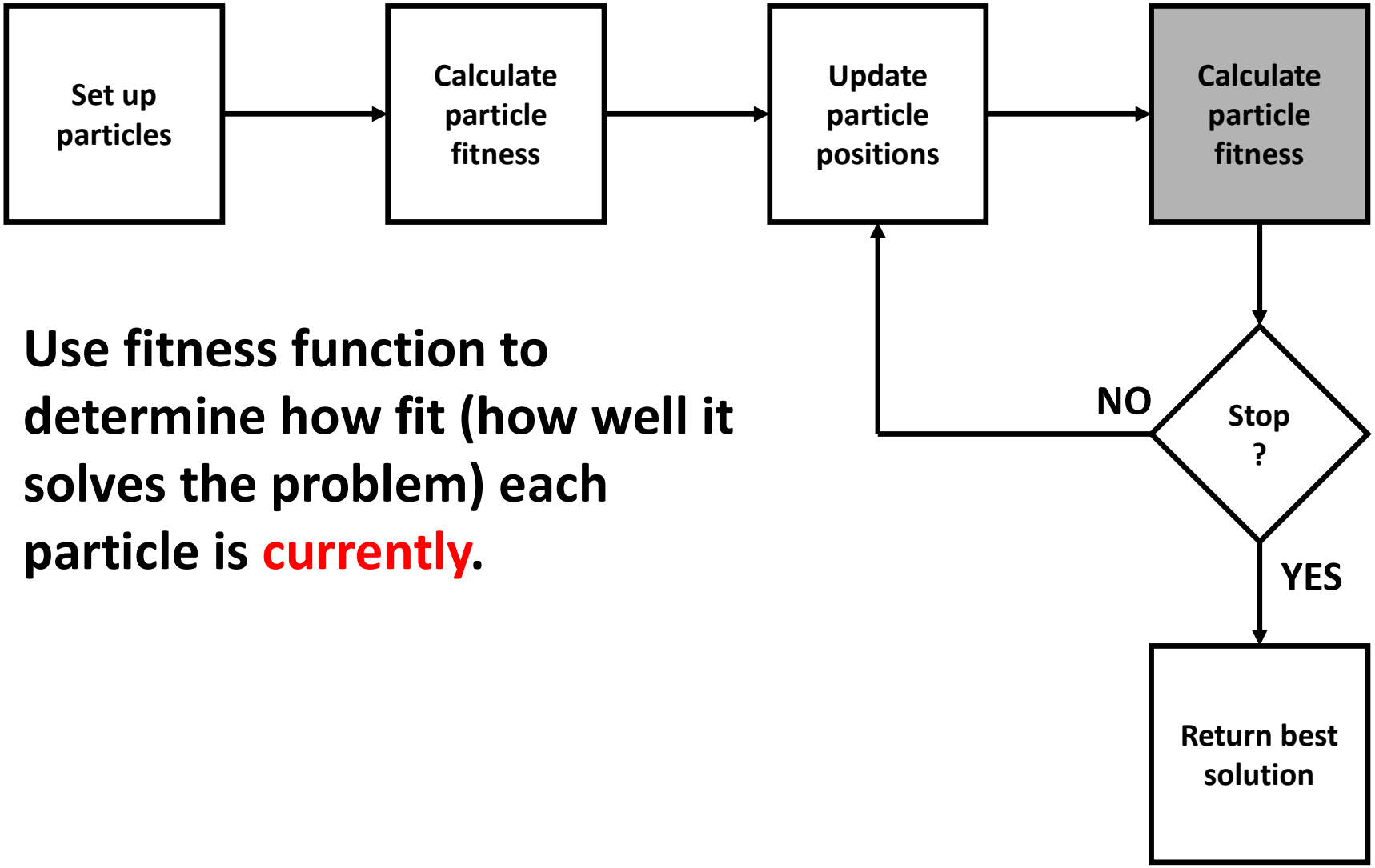


Psychosocial Compromise

Particles adjust their positions according to a "psychosocial compromise" between what an individual is comfortable with, and what society thinks.

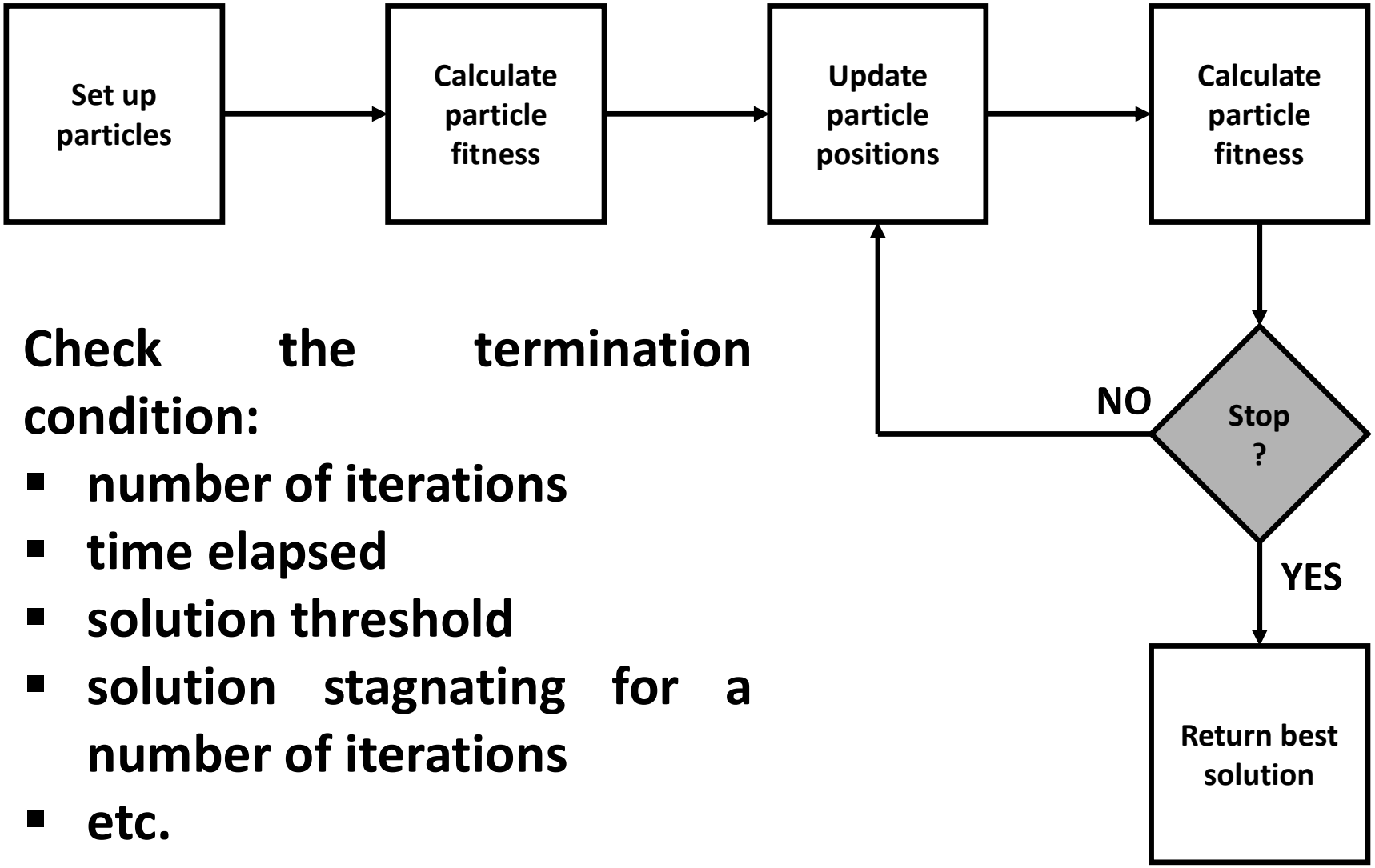


Particle Swarm Optimization: Cycle



Use fitness function to determine how fit (how well it solves the problem) each particle is currently.

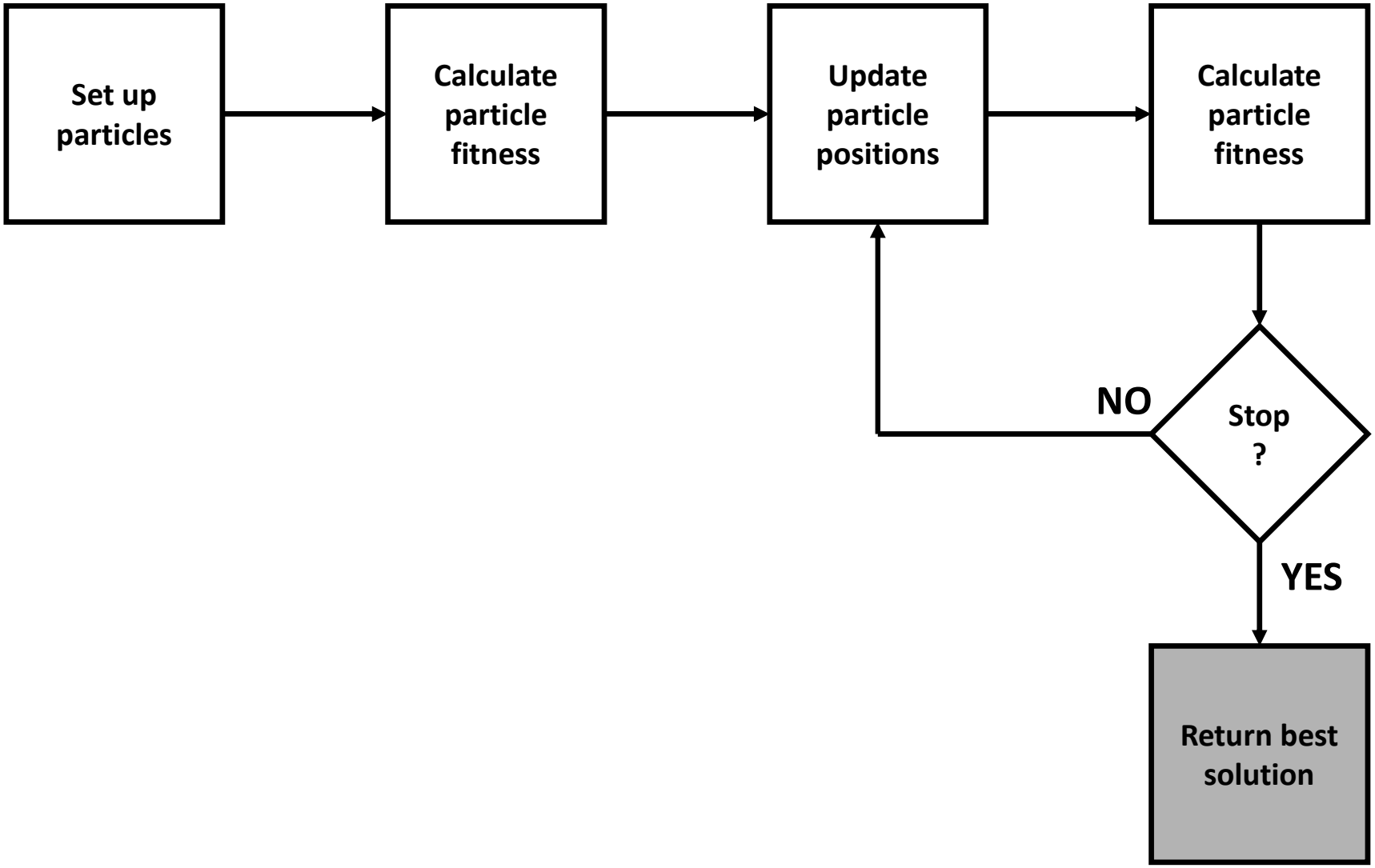
Particle Swarm Optimization: Cycle



Check the termination condition:

- **number of iterations**
- **time elapsed**
- **solution threshold**
- **solution stagnating for a number of iterations**
- **etc.**

Particle Swarm Optimization: Cycle



Particle Swarm Optimization Parameters

PSO: Controlling the Search

- If velocity is too low → algorithm too slow
- If velocity is too high → algorithm too unstable

PSO: Parameters

- **Number of particles** N **usually between 10 and 50**
- c_1 **significance of personal particle experience (trust in individual knowledge)**
- c_2 **significance of swarm experience (trust in social knowledge)**
- **Inertia weight** w
- V_{max} **velocity cap**
- N_i **neighborhood of particle** i
  - **Does NOT have to be global (entire swarm)**

c_1, c_2 Coefficients

- **Typically $c_1 + c_2 = 4$ (empirical values)**

a, b **Coefficients**

- **Can be:**
  - $a^t = f()$
  - $b^t = f()$

Next position:

$$x_i^{t+1} = x_i^t + \vec{v}_i^{t+1}$$

Next velocity:

$$\vec{v}_i^{t+1} = w * \vec{v}_i^t + c_1 * a^t * (x_{ibest}^t - x_i^t) + c_2 * b^t * (x_{Gbest}^t - x_i^t)$$

V_{max} Velocity Cap

To better control particle trajectory and prevent stochastic velocity change to have uncontrolled effect (divergence), PSO parameter V_{max} can be used to dampen that effect:

if $v_{id} > V_{max}$**, then** $v_{id} = V_{max}$

else if $v_{id} < -V_{max}$**, then** $v_{id} = -V_{max}$

Where:

v_{id}: **particle** i **velocity in dimension** d **(scalar)**

w Inertia Weight

- Large inertia weight facilitates global exploration, small on facilitates local exploration
- w must be selected carefully and/or decreased (adaptive) over the run
- Inertia weight similar to temperature in simulated annealing

Exploration vs. Exploitation

The number of iterations influences multiple aspects of the search:

- **Exploration: particles need time to explore the search space and, ultimately, find better solutions**
- **Exploitation: particles should converge on a good solution after some exploration**

Diversification vs. Intensification

The number of iterations influences multiple aspects of the search:

- **Diversification: particles need time to explore the search space and, ultimately, find better solutions**
- **Intensification: particles should converge on a good solution after some exploration**

Diversification vs. Intensification

Next position:

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

Next velocity:

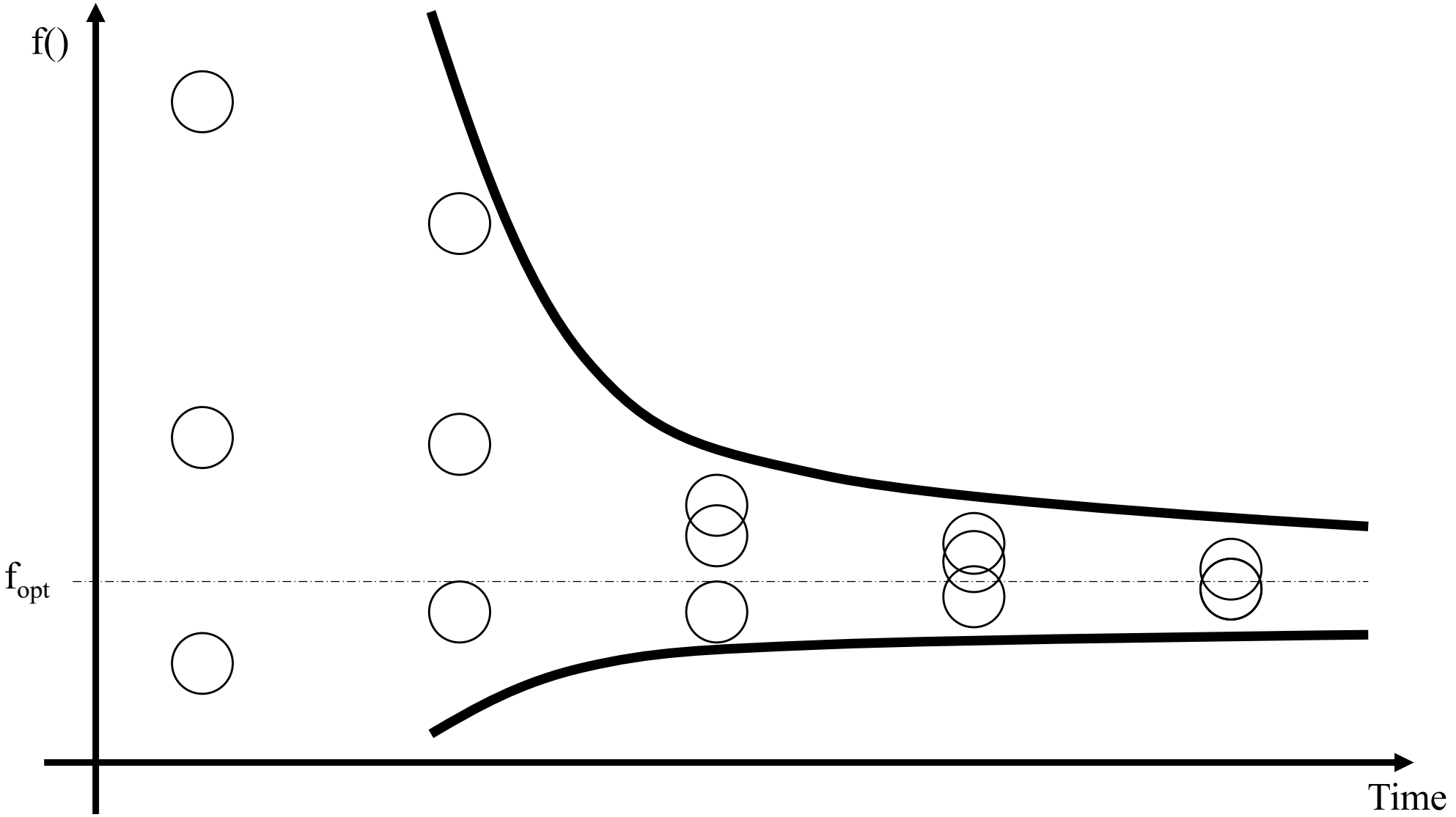
$$\vec{v}_i^{t+1} = w * \vec{v}_i^t + c_1 * a * (x_{ibest}^t - x_i^t) + c_2 * b * (x_{Gbest}^t - x_i^t)$$

DIVERSIFICATION **INTENSIFICATION**

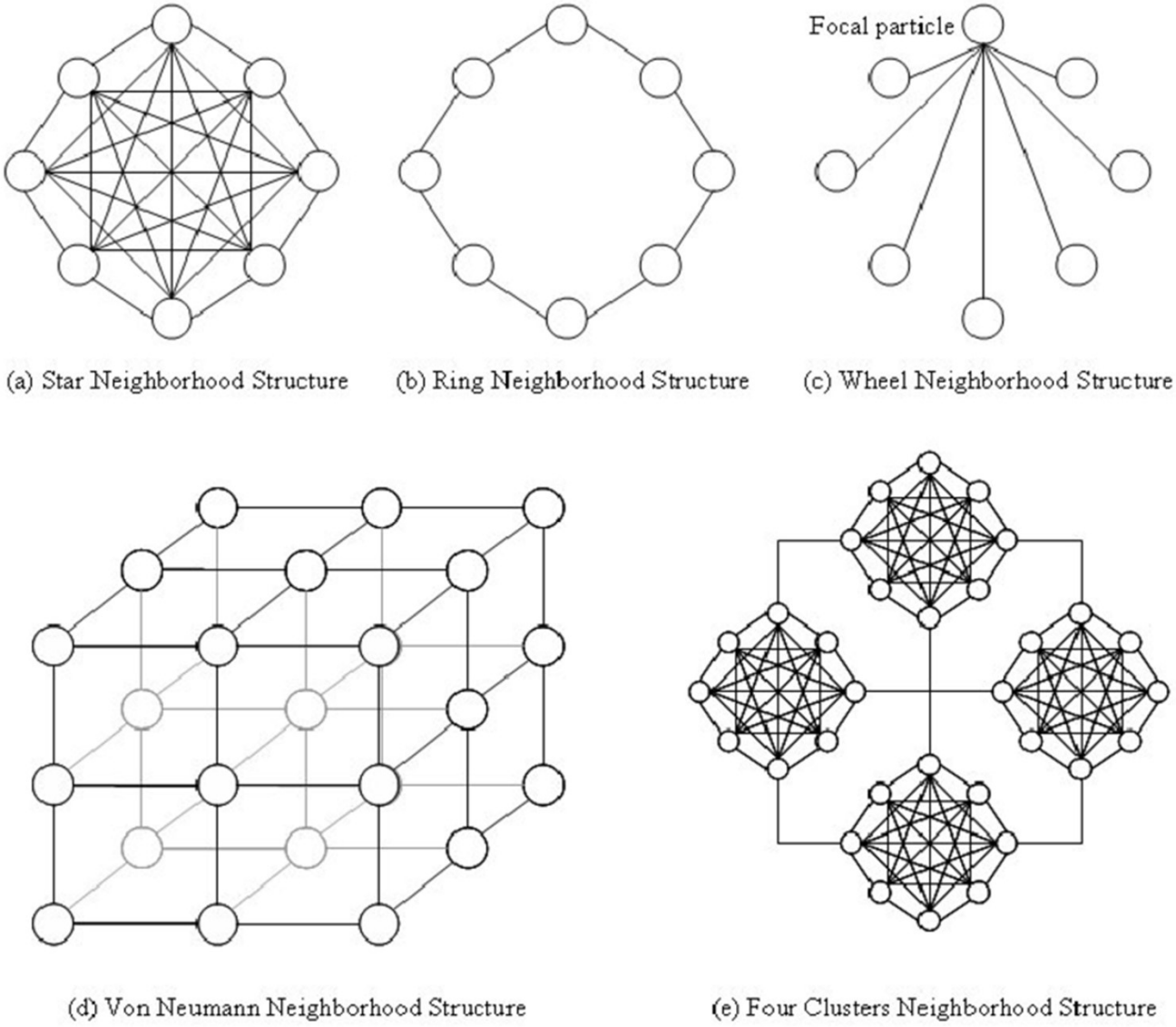
Intensification: explores the previous solutions, finds the best solution of a given region

Diversification: searches new solutions, finds the regions with potentially the best solutions

Exploration vs. Exploitation

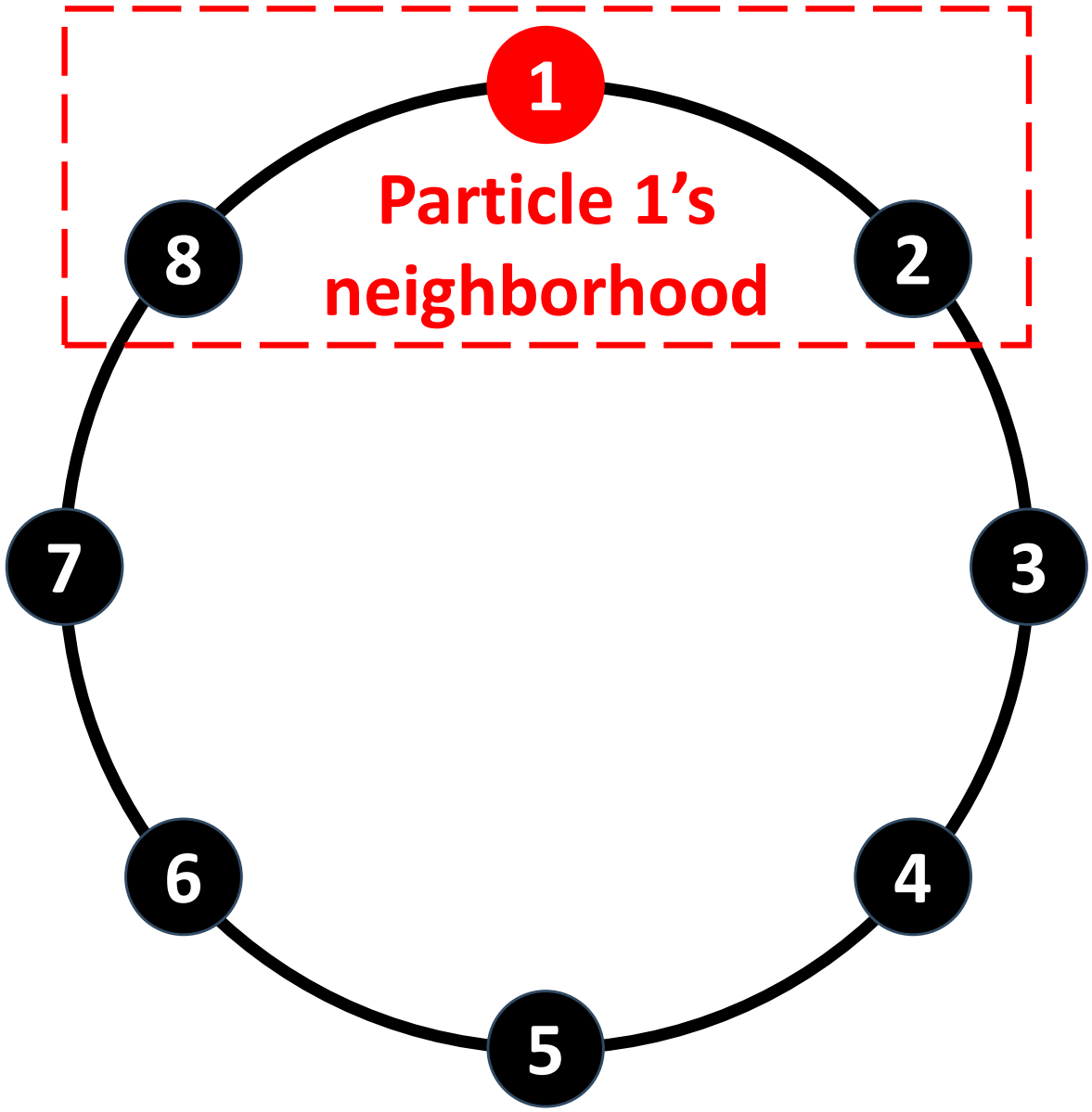


Neighborhoods



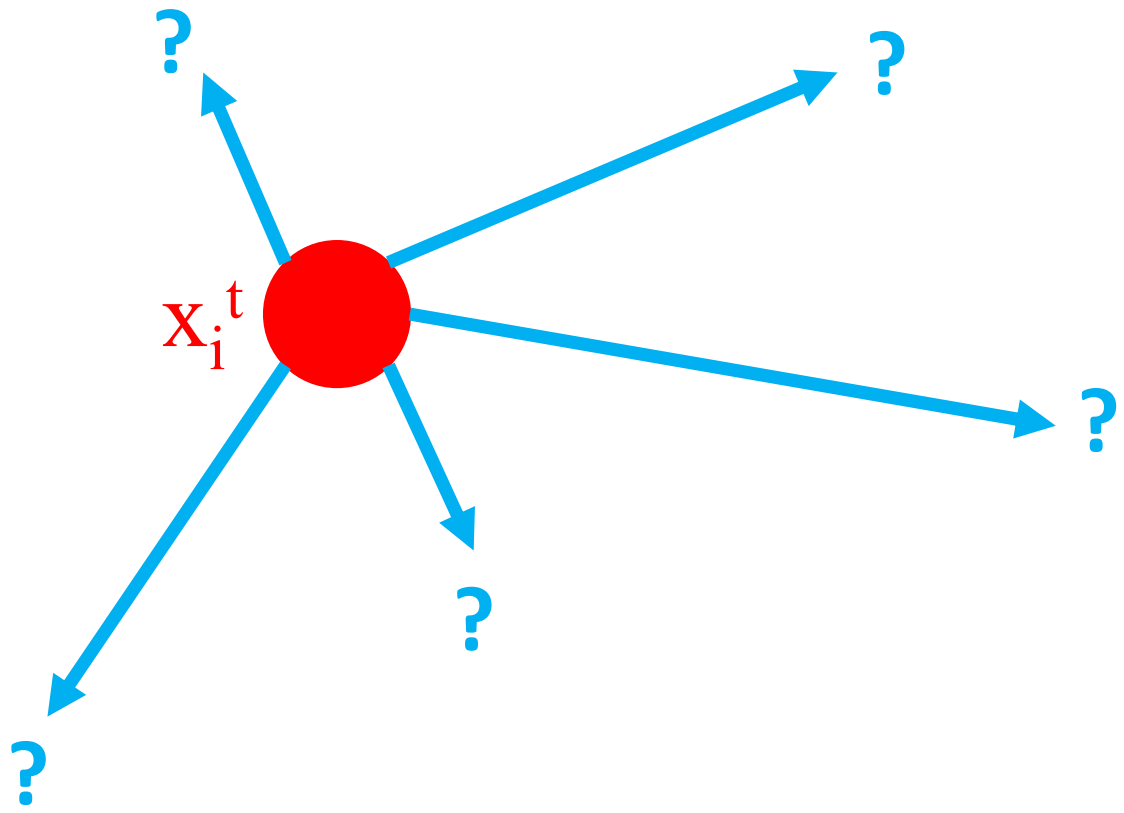
Source: Angel Munoz-Zavala, et al. "Robust PSO-Based Constrained Optimization by Perturbing the Particle's Memory"

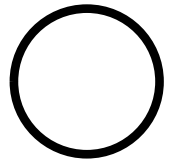
Ring / Circular Neighborhood



Particle Position: Where Next?

neighborhood's (local) best position

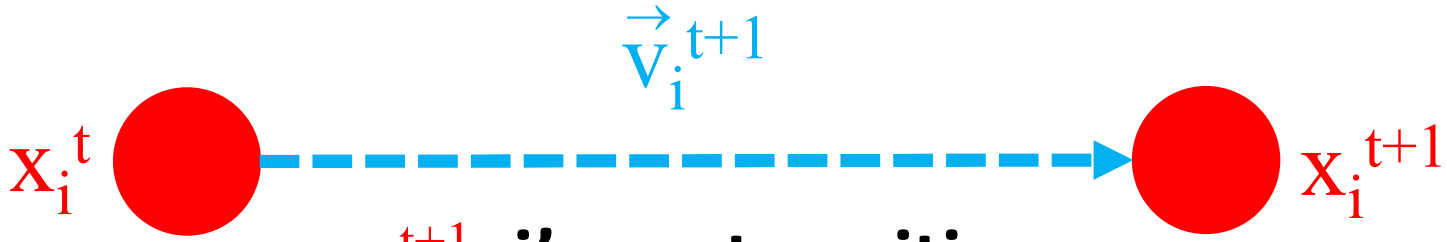




Particle i's best position

x_{ibest}^{t}

Particle Position Update



$x_i^{\,t+1}$: i's next position

$x_i^{\,t}$: i's current position

$\vec{v}_i^{\,t+1}$: i's next velocity

Next position:

$$x_i^{\,t+1} = x_i^{\,t} + \vec{v}_i^{\,t+1}$$

Next velocity:

$$\vec{v}_i^{\,t+1} = w * \vec{v}_i^{\,t} + c_1 * a * (x_{ibest}^{\,t} - x_i^{\,t}) + c_2 * b * (x_{Nbest}^{\,t} - x_i^{\,t})$$

$x_{ibest}^{\,t}$: i's best position

$x_{Nbest}^{\,t}$: neighborhood's best

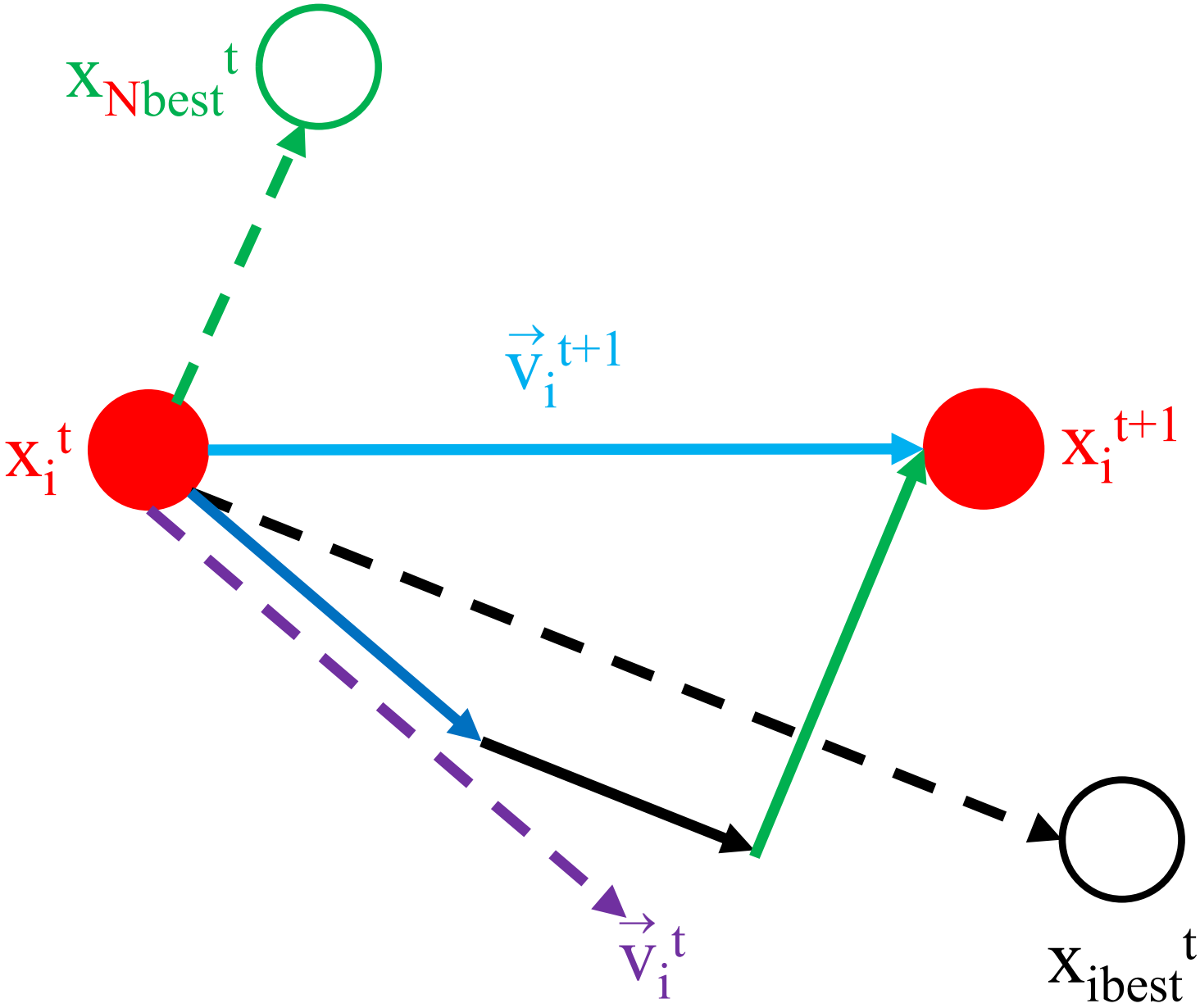
$\vec{v}_i^{\,t}$: i's current velocity

c_1: cognitive constant

c_2: social constant

w: inertia weight

Particle Position Update



PSO: Characteristics

- PSO has a memory:
  - "where" the best solution was (as opposed to “what”)
- Quality: population responds to quality factors (personal and global best)
- Diverse response: responses allocated between personal and global best
- Stability: population changes state only when global best changes
- Adaptability: population does change state when global best changes

PSO: Advantages / Disadvantages

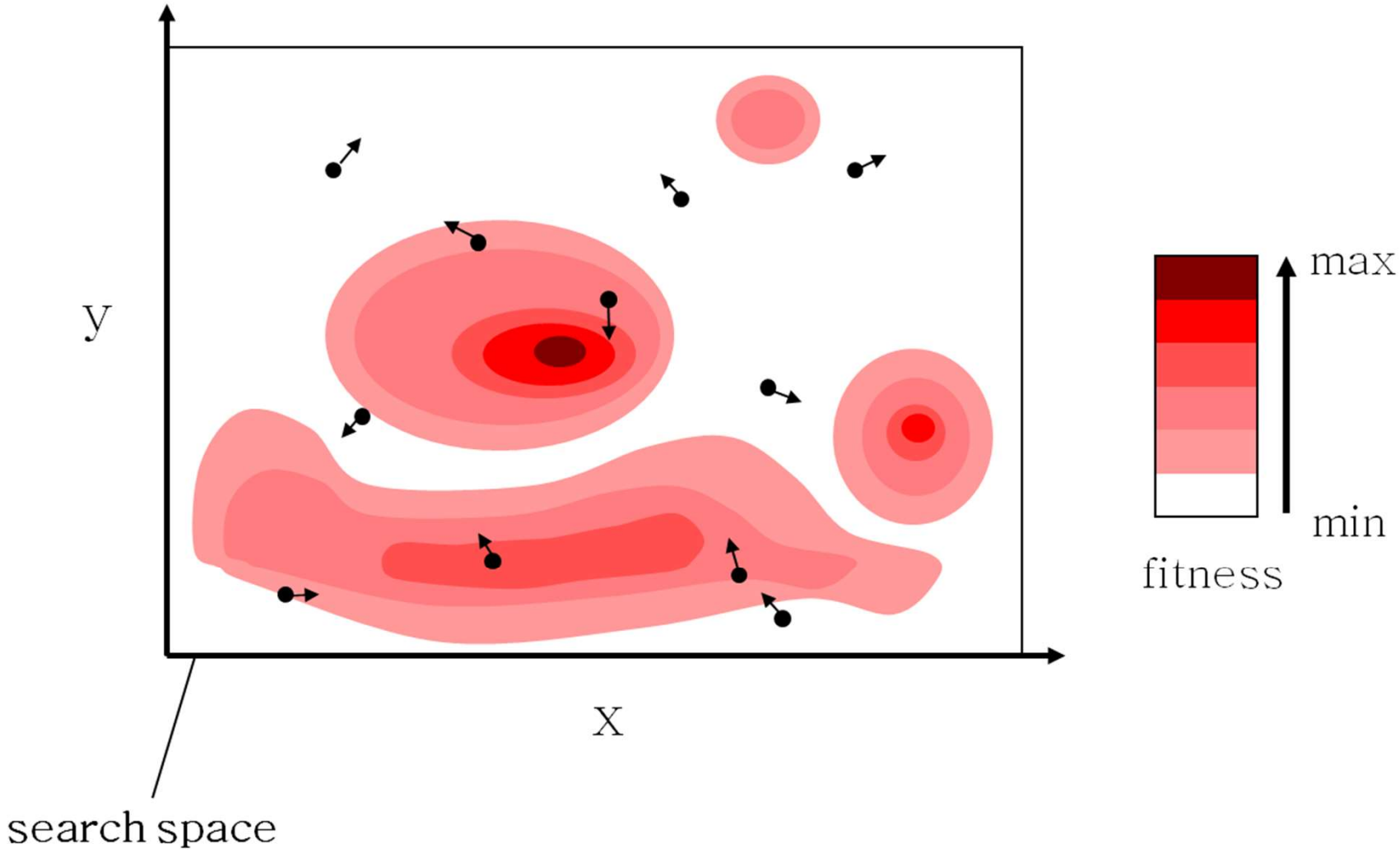
Advantages

- **Insensitive to scaling of design variables**
- **Simple implementation**
- **Easily parallelized for concurrent processing**
- **Derivative free**
- **Very few algorithm parameters**
- **Very efficient global search algorithm**

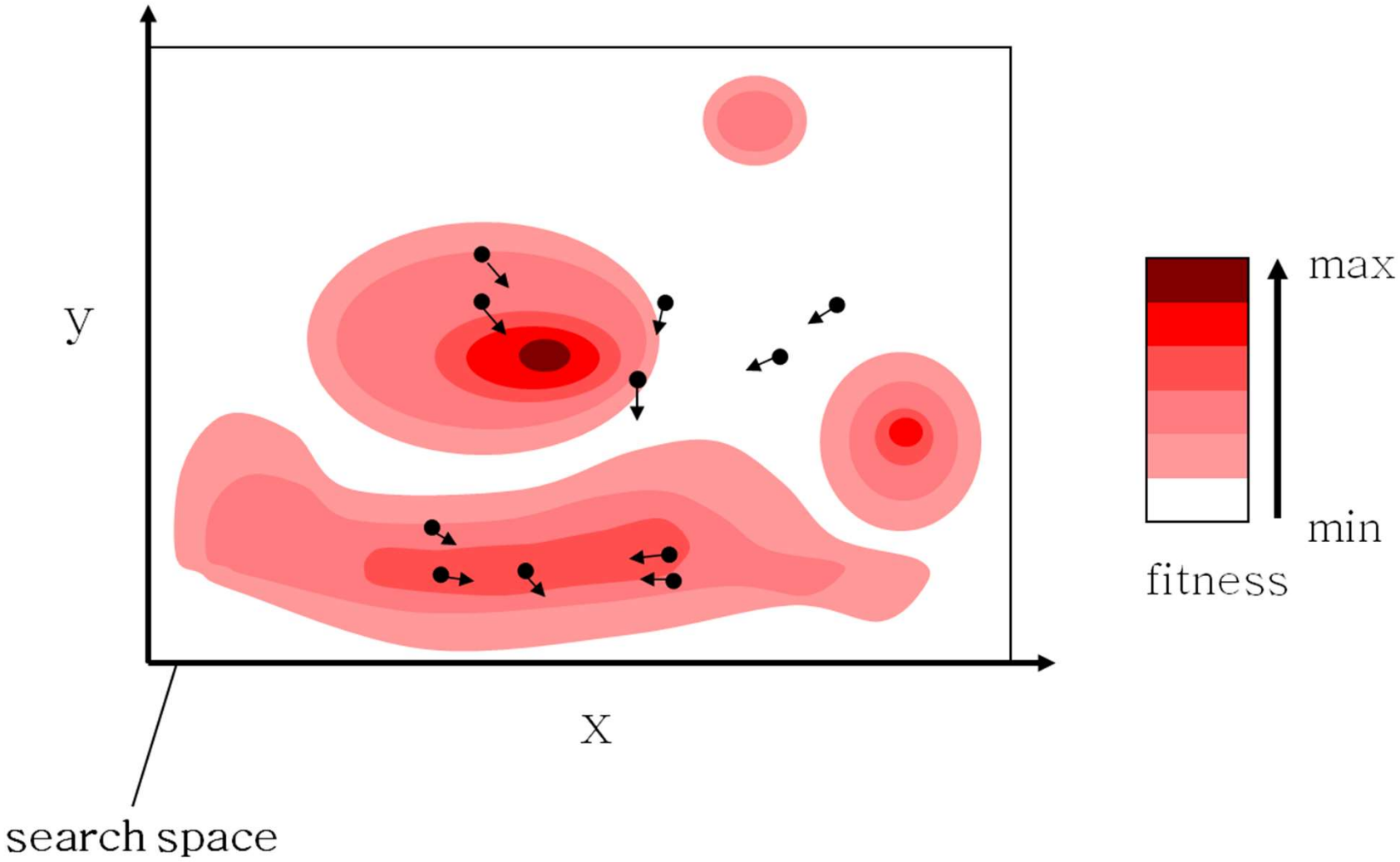
Disadvantages

- **Tendency to a fast and premature convergence in mid optimum points**
- **Slow convergence in refined search**

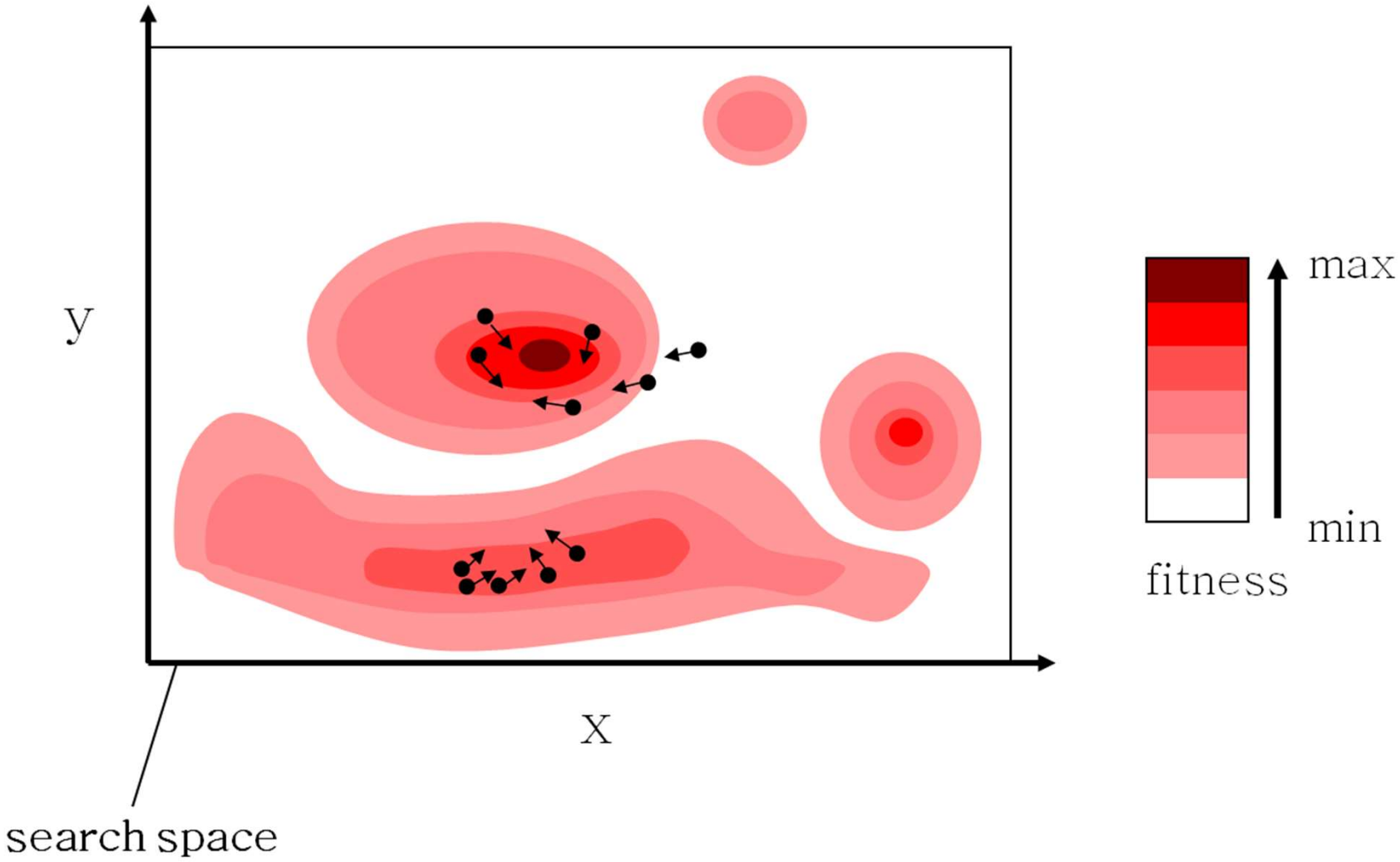
PSO: Visualization



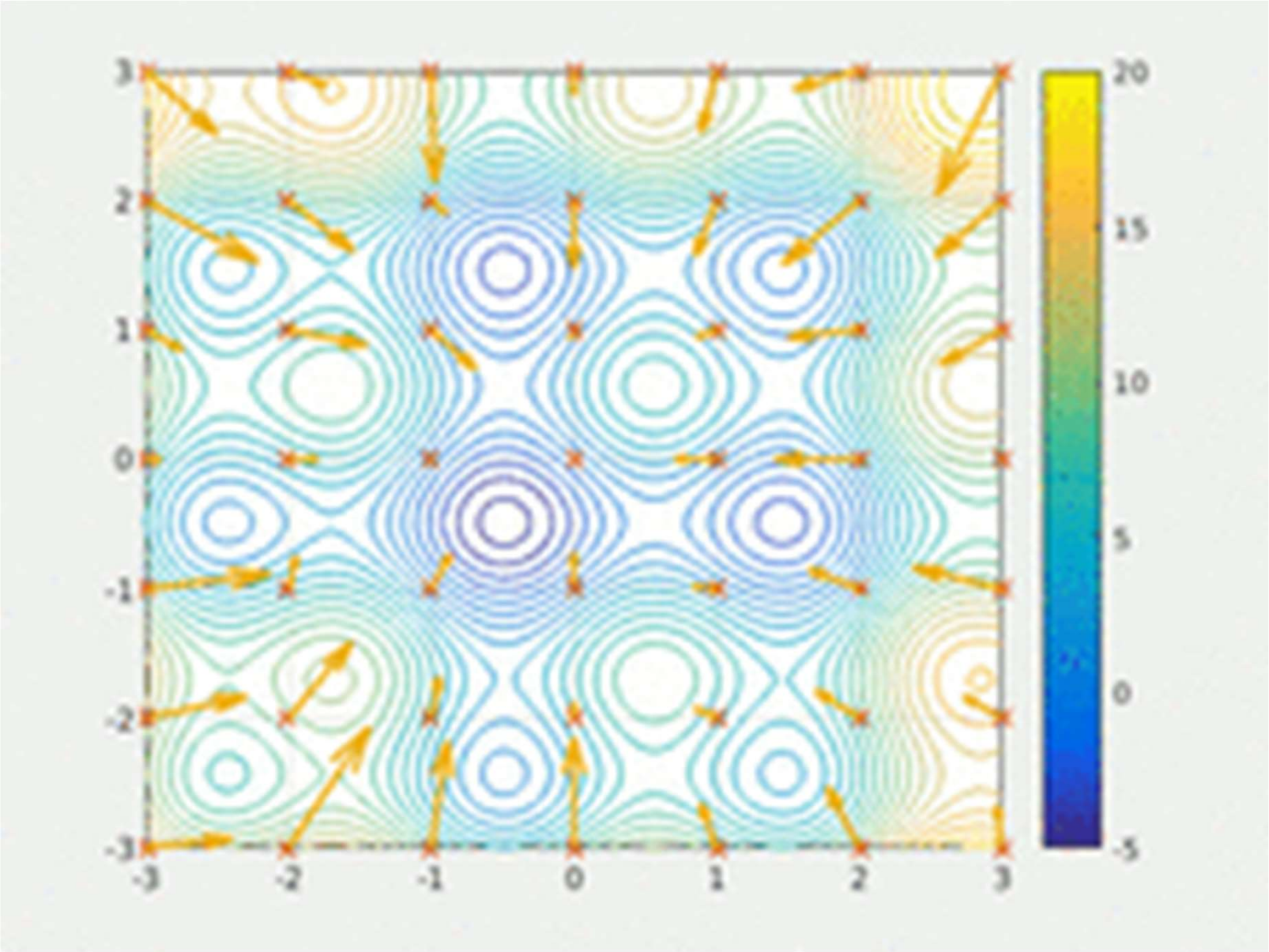
PSO: Visualization



PSO: Visualization

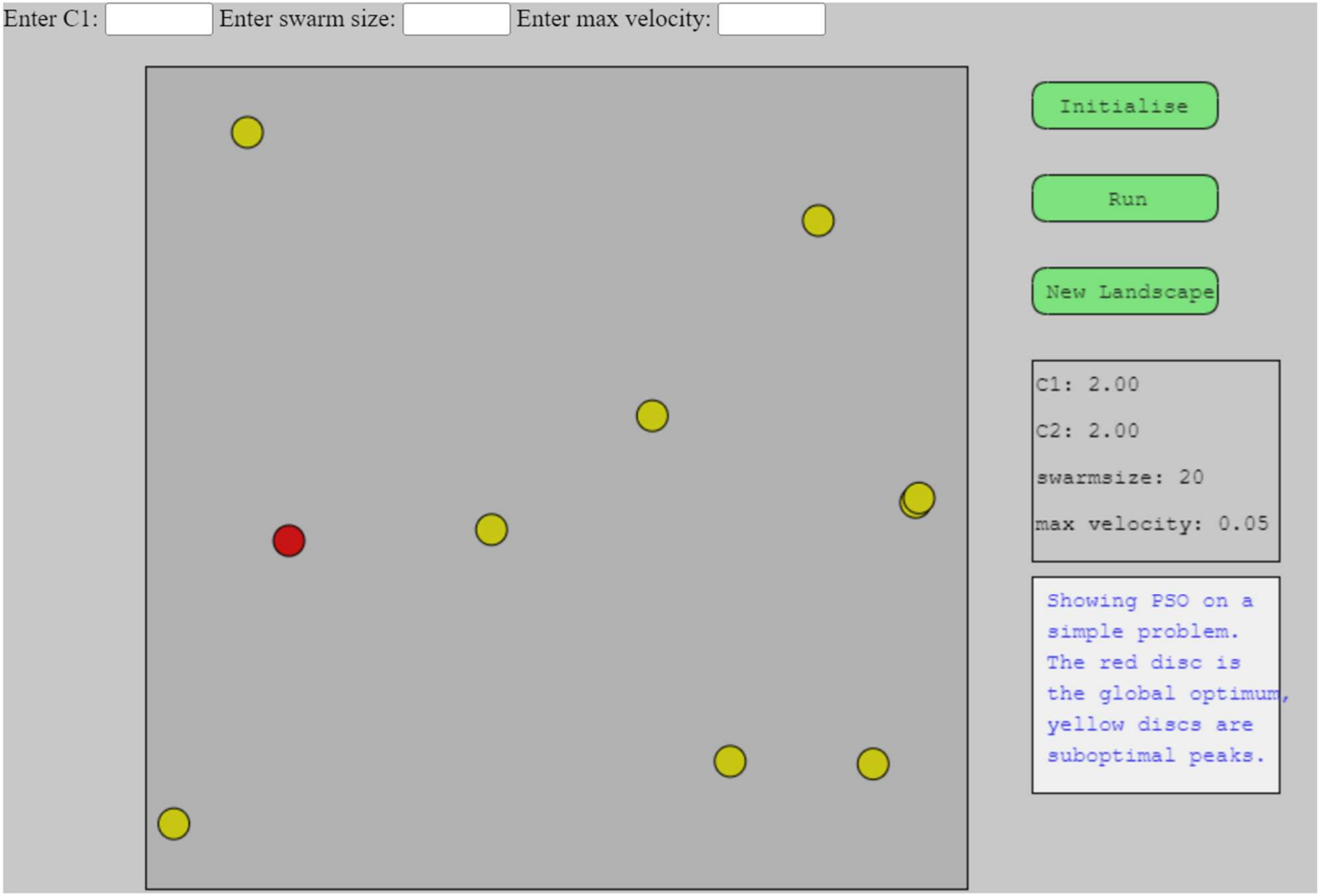


PSO: Visualization



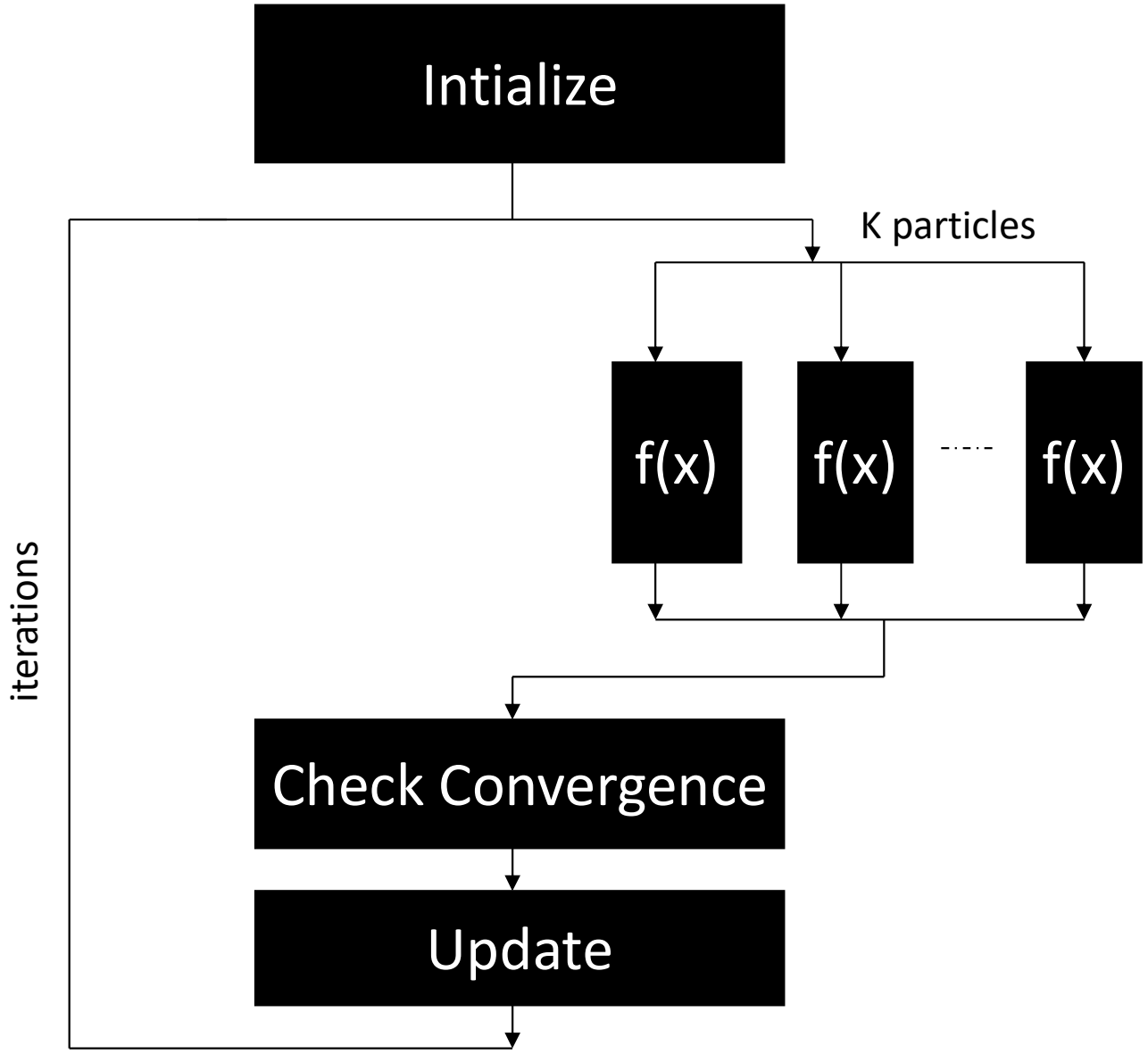
Source: https://en.wikipedia.org/wiki/Particle_swarm_optimization

PSO: Interactive Demo



Source: http://www.macs.hw.ac.uk/~dwcorne/mypages/apps/pso.html

Parallel PSO



PSO Variants / Modifications

- 2-D Otsu PSO
- Active Target PSO
- Adaptive PSO
- Adaptive Mutation PSO
- Adaptive PSO Guided by Acceleration Information
- Attractive Repulsive Particle Swarm Optimization
- Binary PSO
- Cooperative Multiple PSO
- Dynamic and Adjustable PSO
- Extended Particle Swarms

Heuristics and Metaheuristics

- **Heuristics:**
  - **how to choose the next neighbor?**
  - **use local information (state and its neighborhood)**
  - **direct the search towards a local min/maximum**

- **Metaheuristics:**
  - **how to escape local minima?**
  - **direct the search towards a global min/maximum**
  - **typically include some memory or learning**

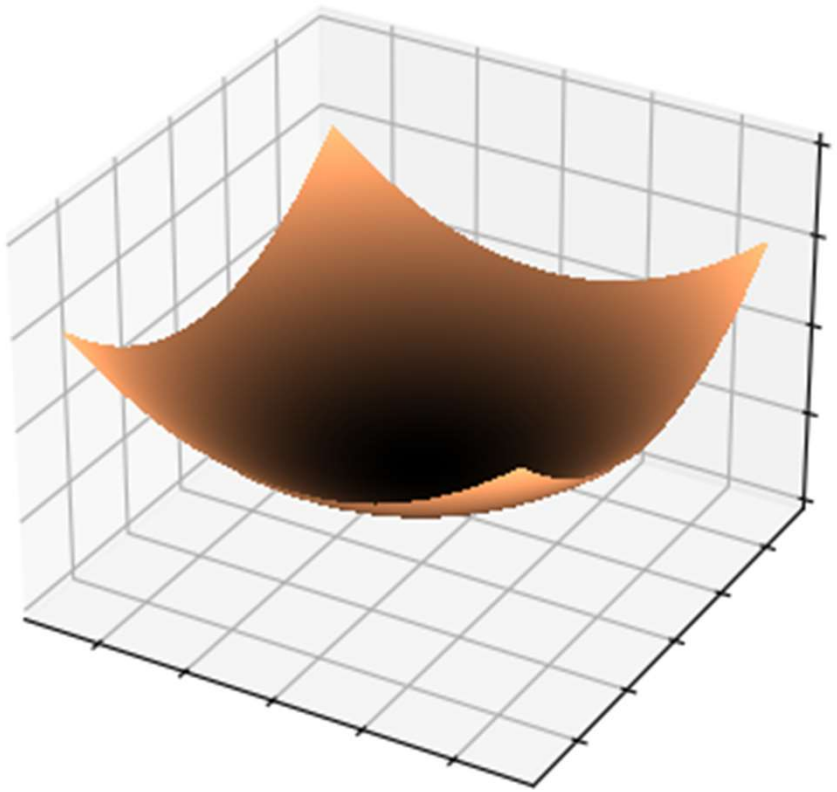
Other Swarm Algorithms

- Bat Algorithm
- Artificial Fish Swarm
- Cuckoo Search Algorithm
- Harmony Search Algorithm
- Flower Pollination Algorithm
- Firefly Algorithm
- Artificial Bee Colony Algorithm
- Wolf-based Search Algorithm
- Artificial Immune System

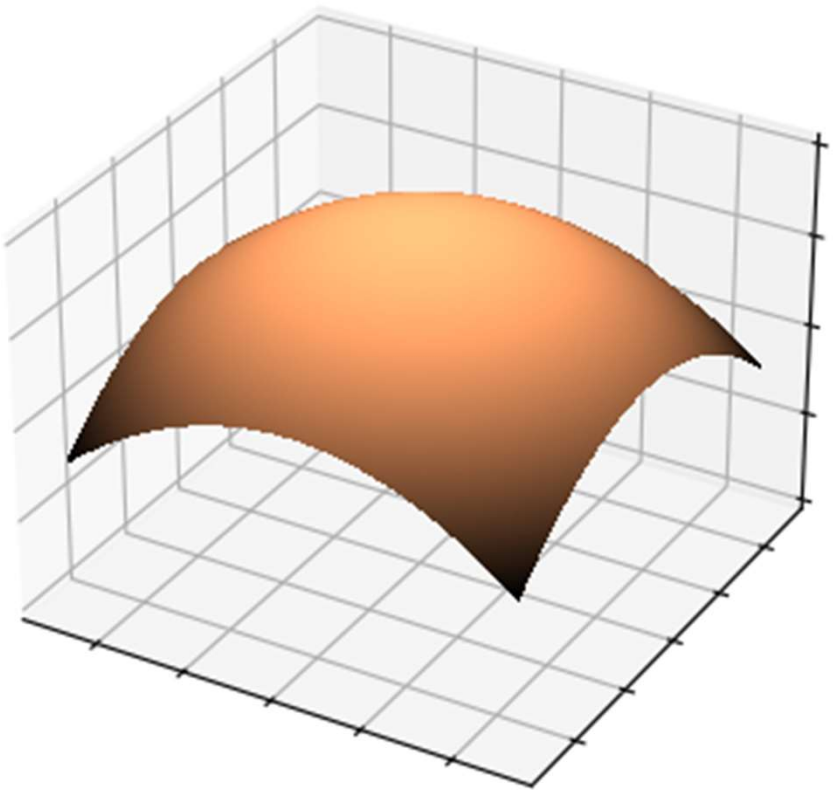
Gradient Descent

Convex vs. Concave Functions

Convex Function

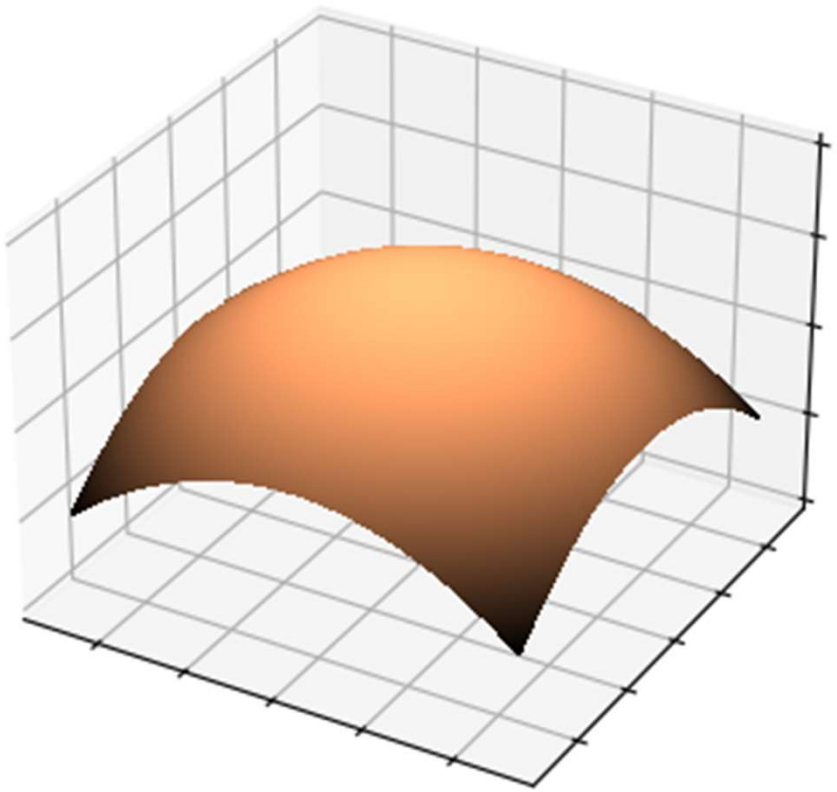


Concave Function

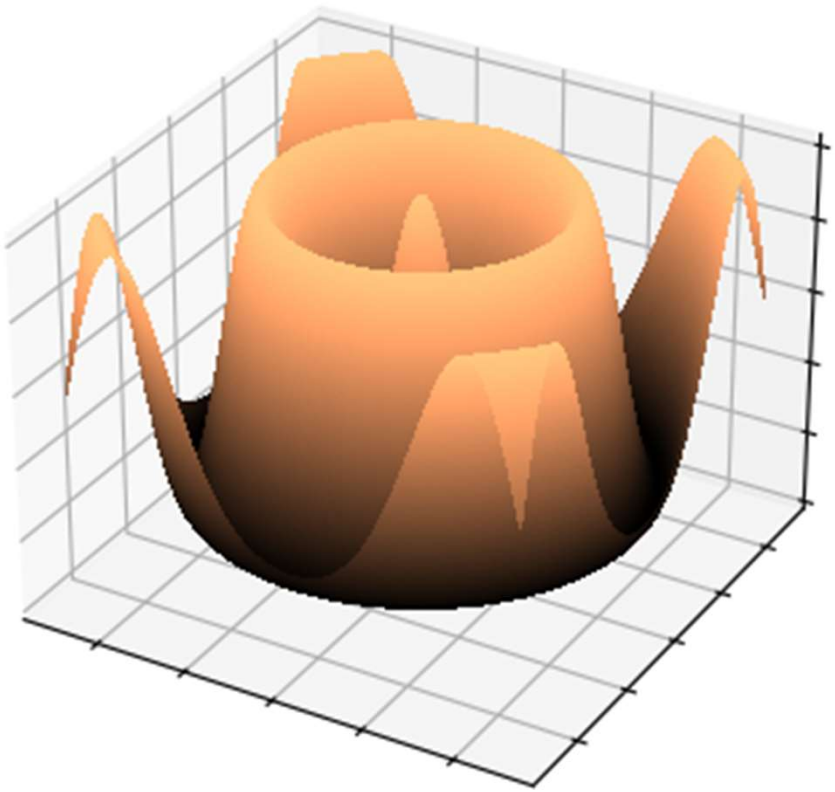


Concave vs. Non-Convex Functions

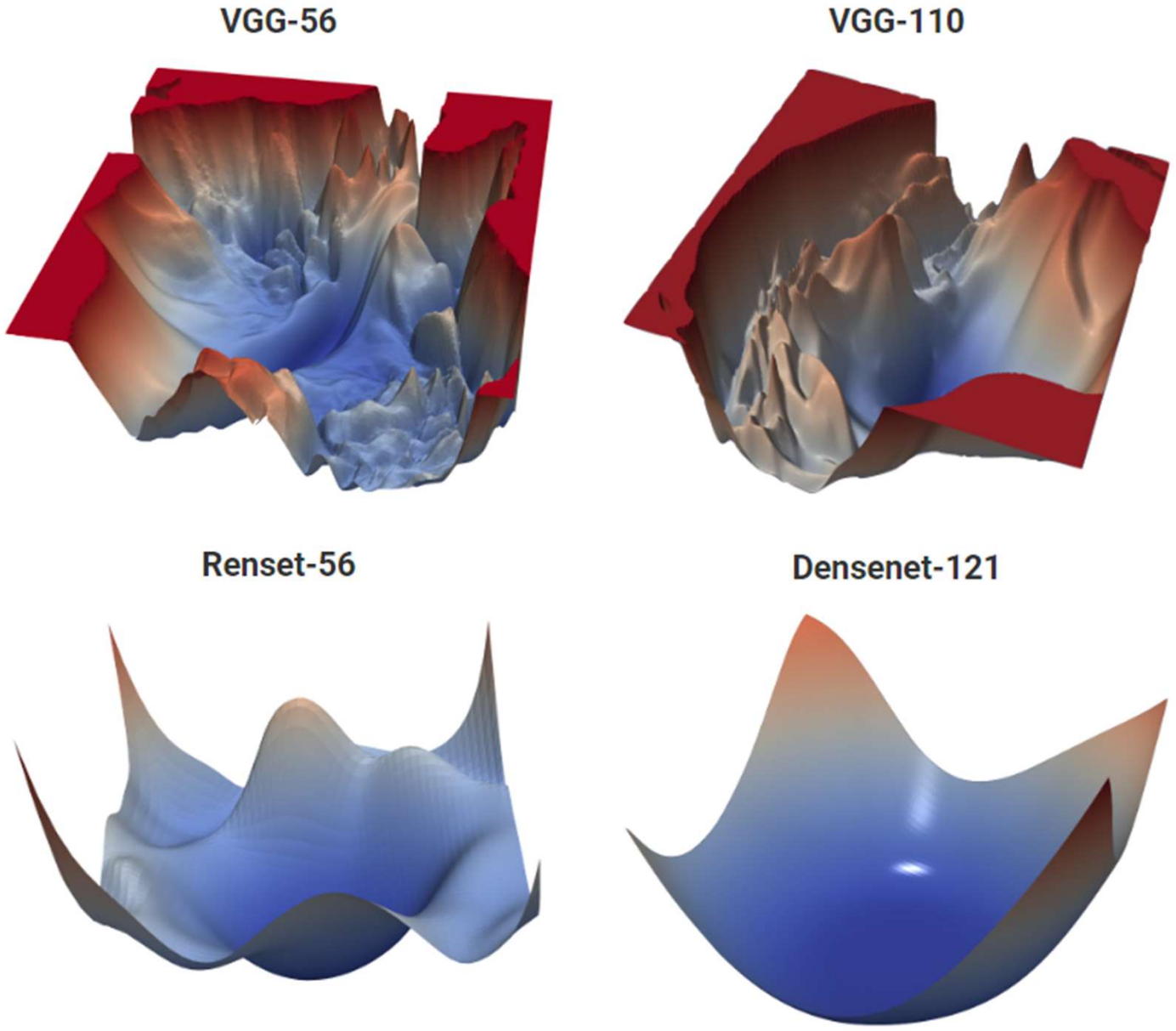
Concave Function



Non-Convex Function



Neural Network Loss Landscape



Source: https://www.cs.umd.edu/~tomg/projects/landscapes/

Evaluation / Objective Function

Say we have the following objective function:

$$f(\mathbf{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3)$$

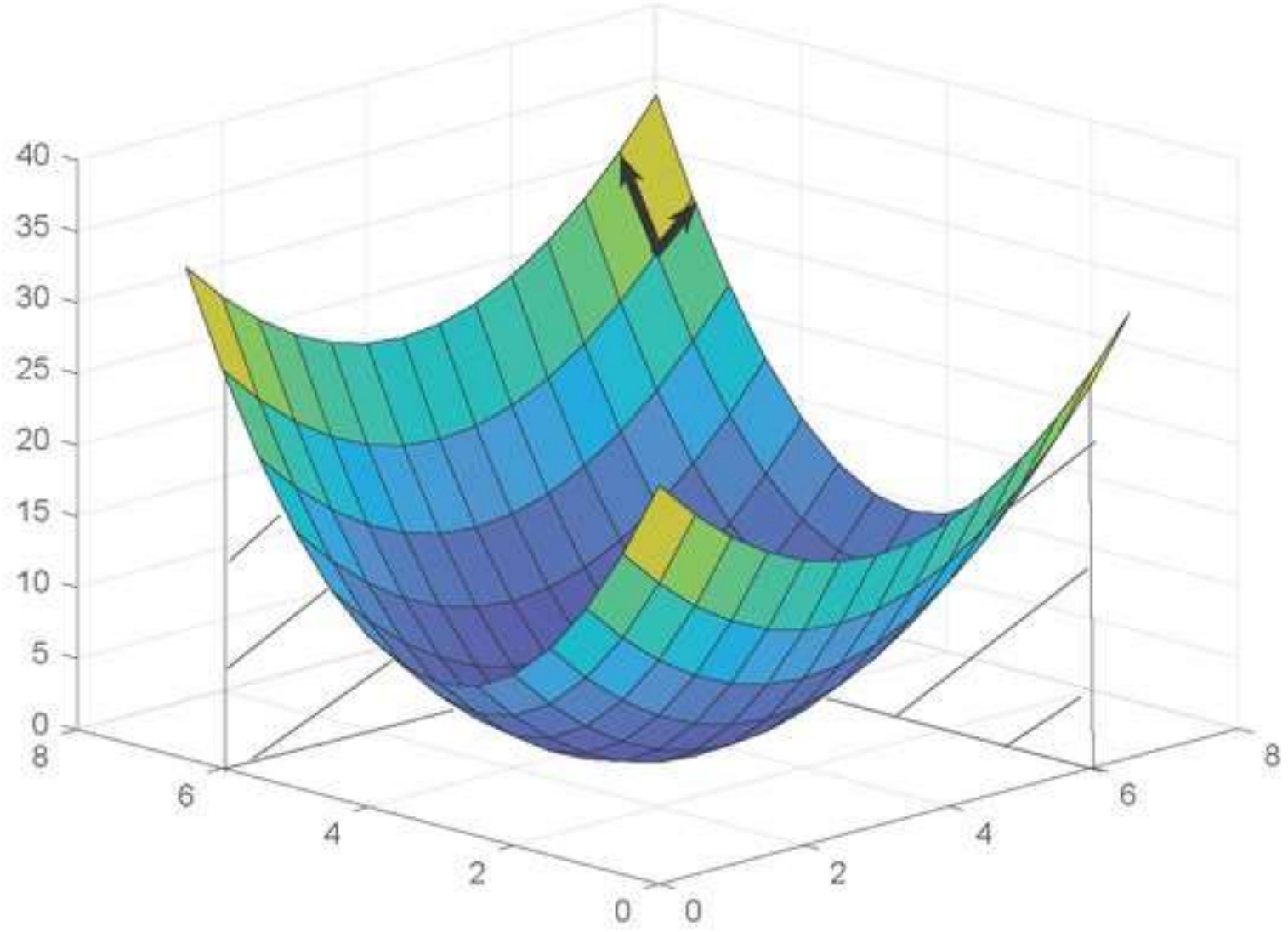
Gradients and Gradient Descent

The **gradient** of a function of many variables is a **vector pointing in the direction of the greatest increase** in a function.

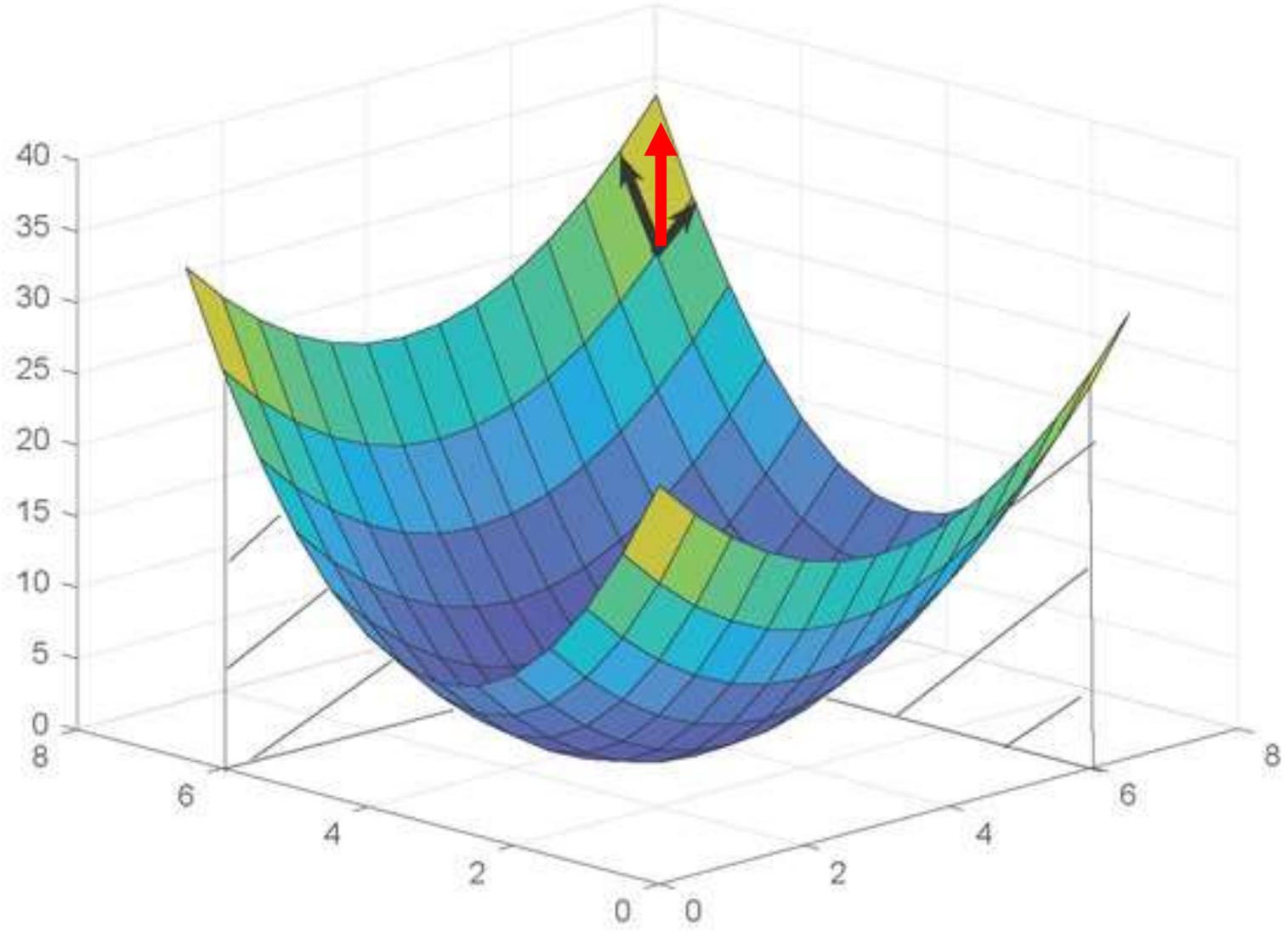
$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

Gradient Descent: Find the gradient of the function at the current point and move in the **opposite** direction.

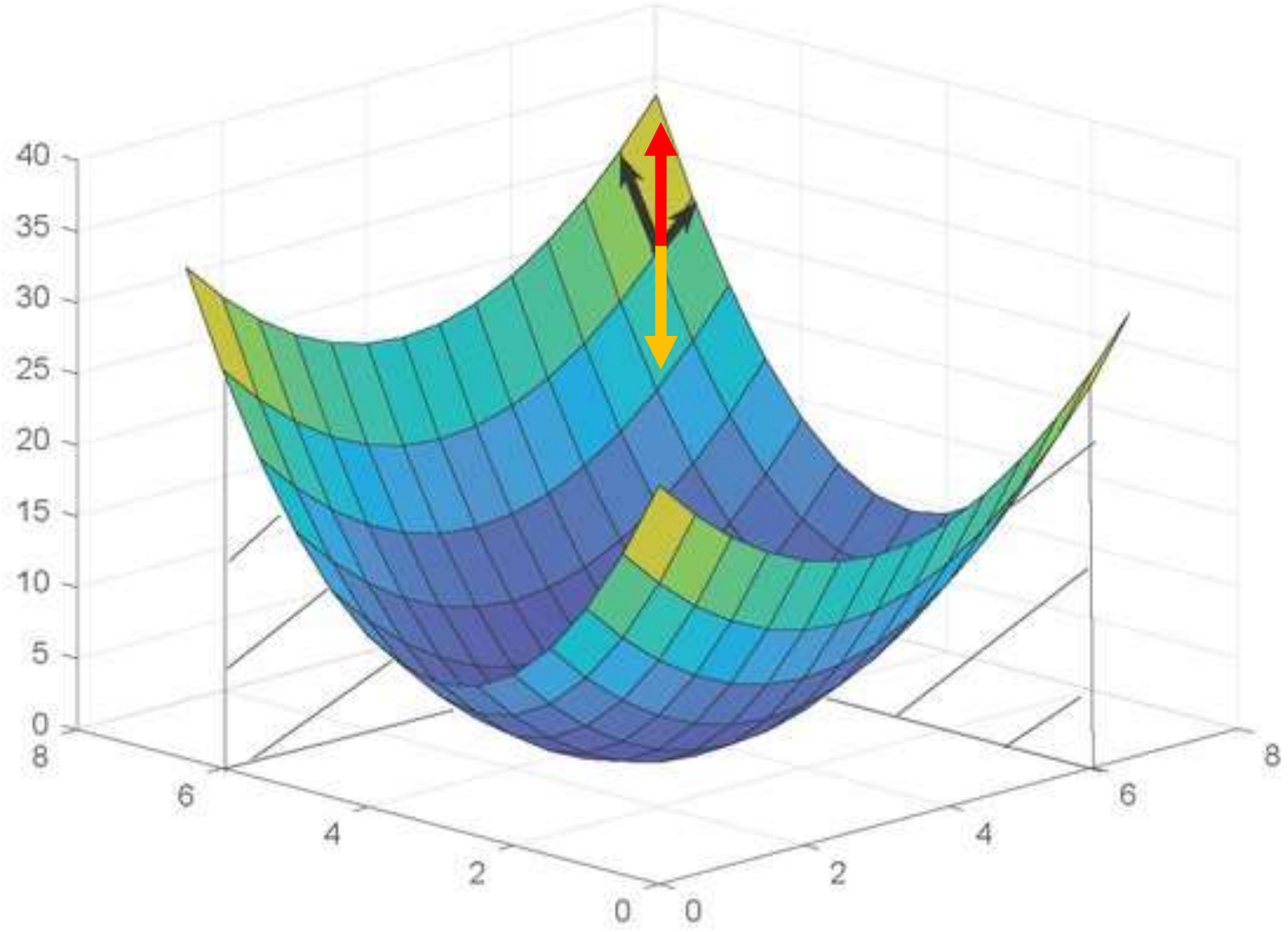
Gradient=Steepest Ascent Direction



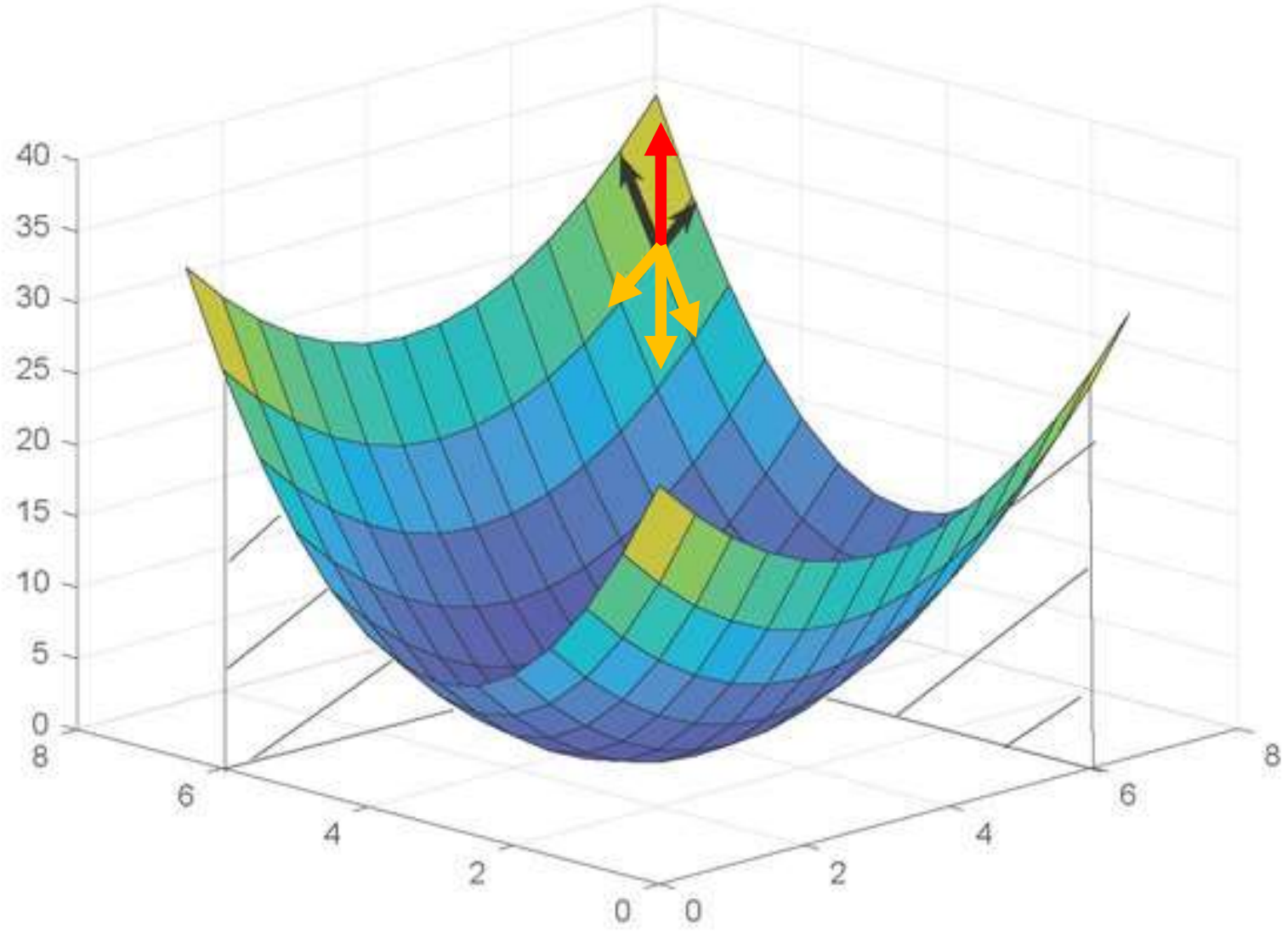
Gradient=Steepest Ascent Direction



-Gradient=Steepest Descent Direction



-Gradient=Steepest Descent Direction



Gradients and Gradient Descent

Gradient Descent Algorithm:

- **Pick an initial point x_0**
- **Iterate until convergence**

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

where γ_t is the t^{th} step size (sometimes called learning rate)

Empirical Gradient

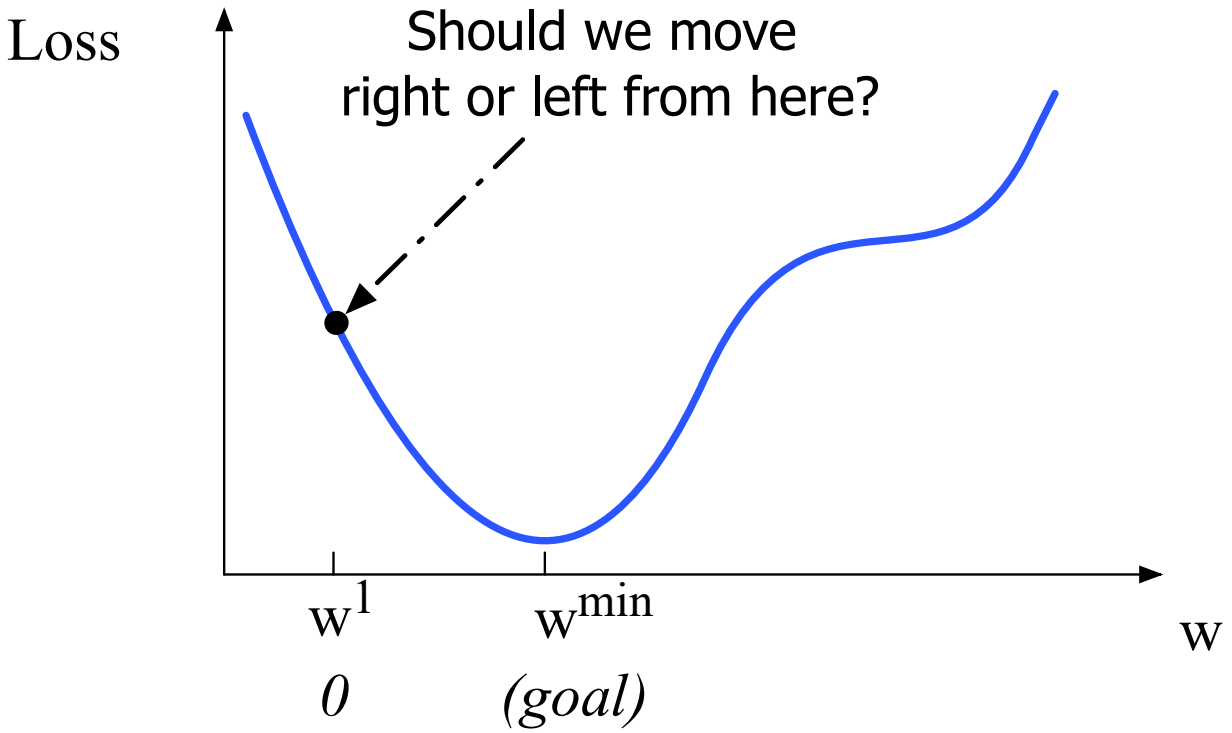
In other cases, the objective / evaluation function might not be available in a differentiable form at all.

In those cases, a so-called empirical gradient can be determined by evaluating the response to small increments and decrements in each coordinate.

Minimizing Functions

Q: Given current **w**, should we make it bigger or smaller?

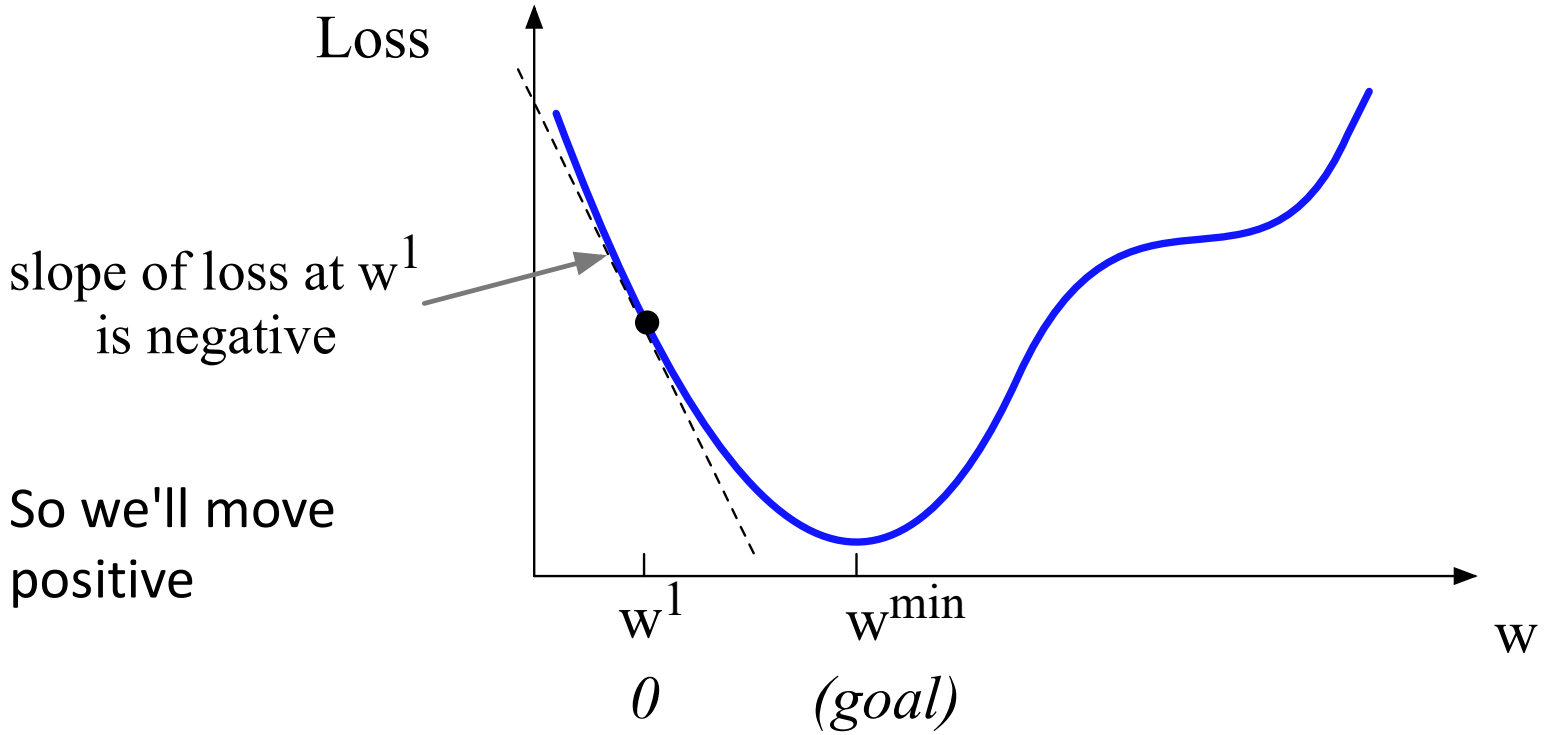
A: Move **w** in the reverse direction from the slope of the function



Minimizing Functions

Q: Given current **w**, should we make it bigger or smaller?

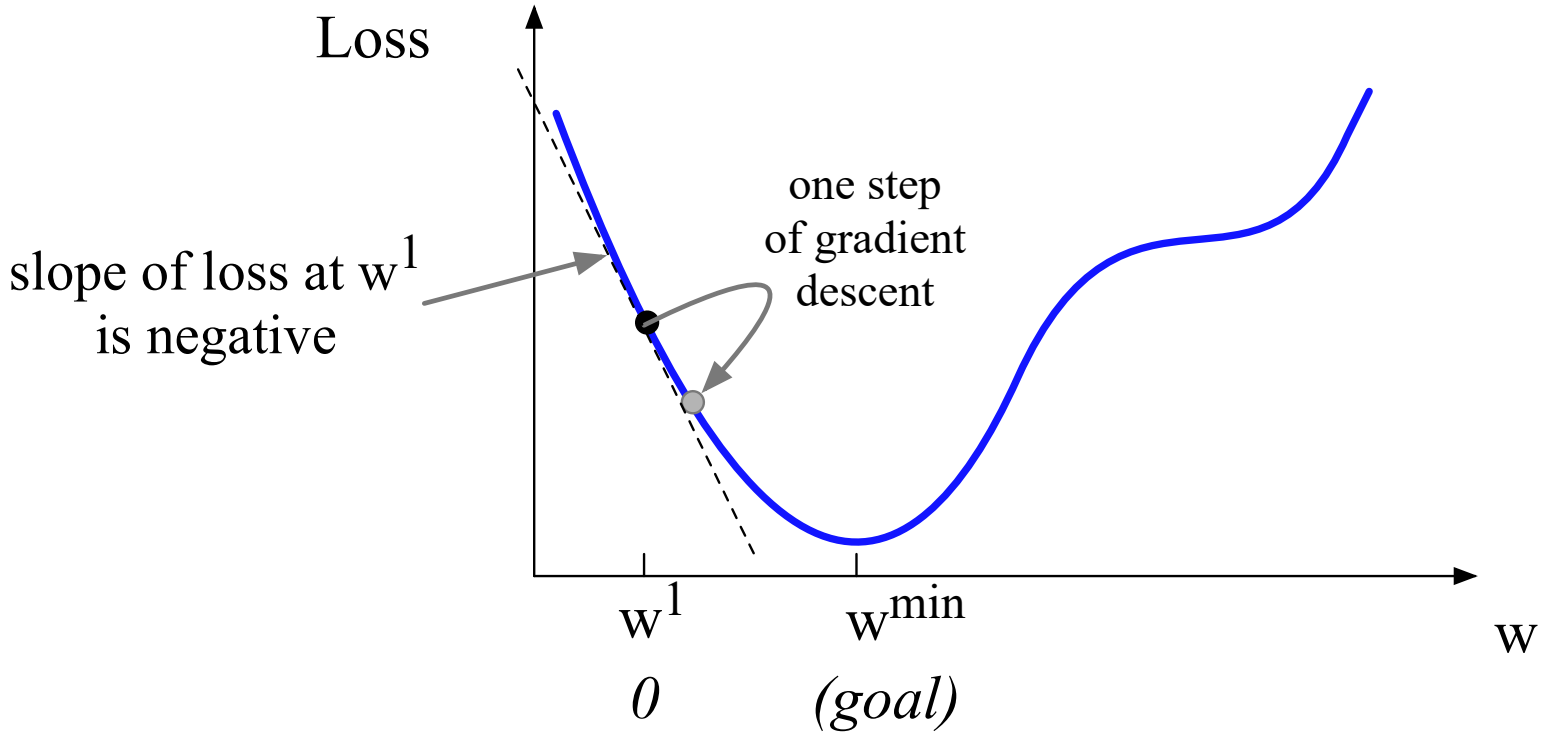
A: Move **w** in the reverse direction from the slope of the function



Minimizing Functions

Q: Given current **w**, should we make it bigger or smaller?

A: Move **w** in the reverse direction from the slope of the function

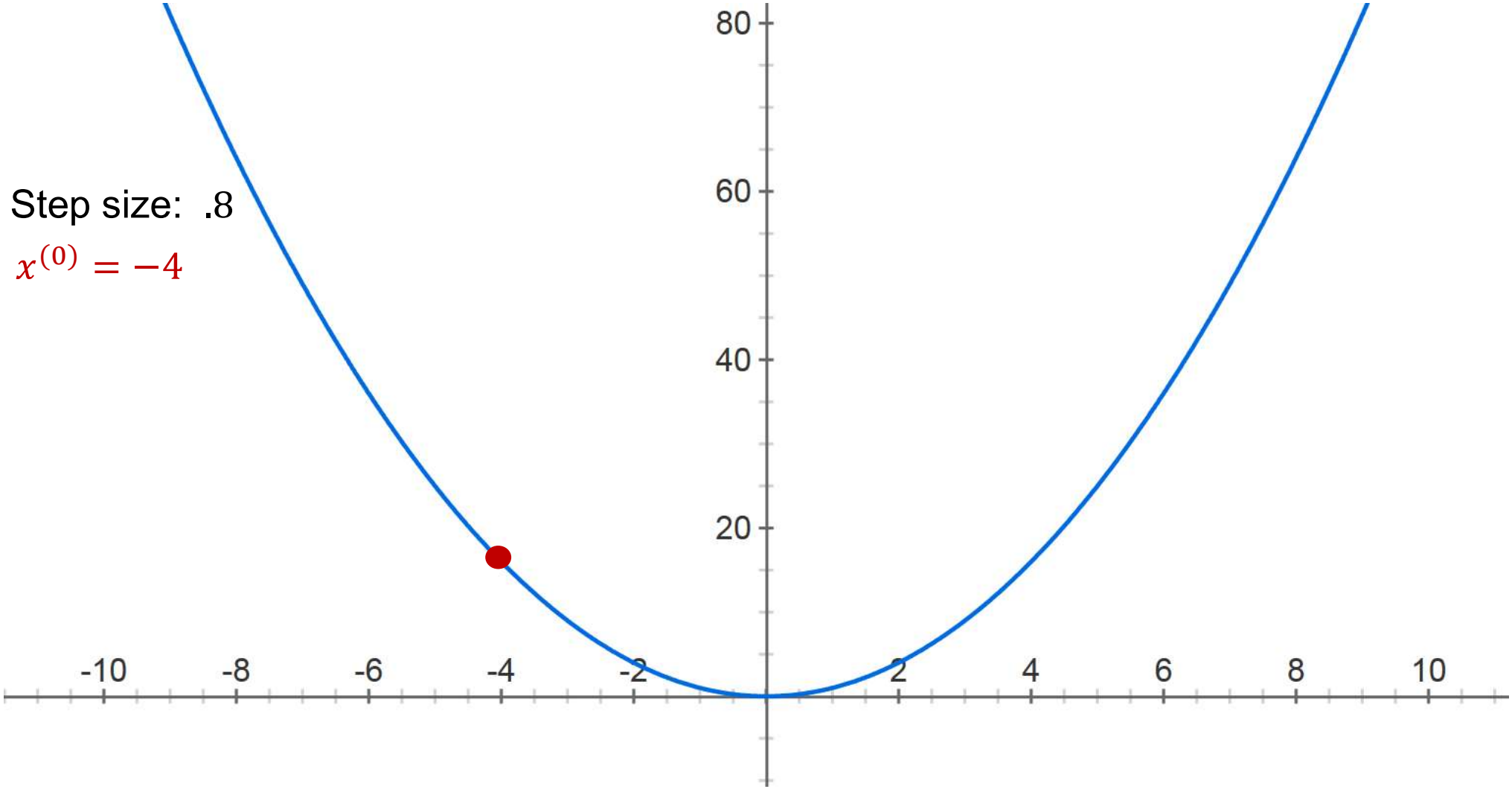


Minimizing Functions

$f(x) = x^2$

Step size: .8

$x^{(0)} = -4$



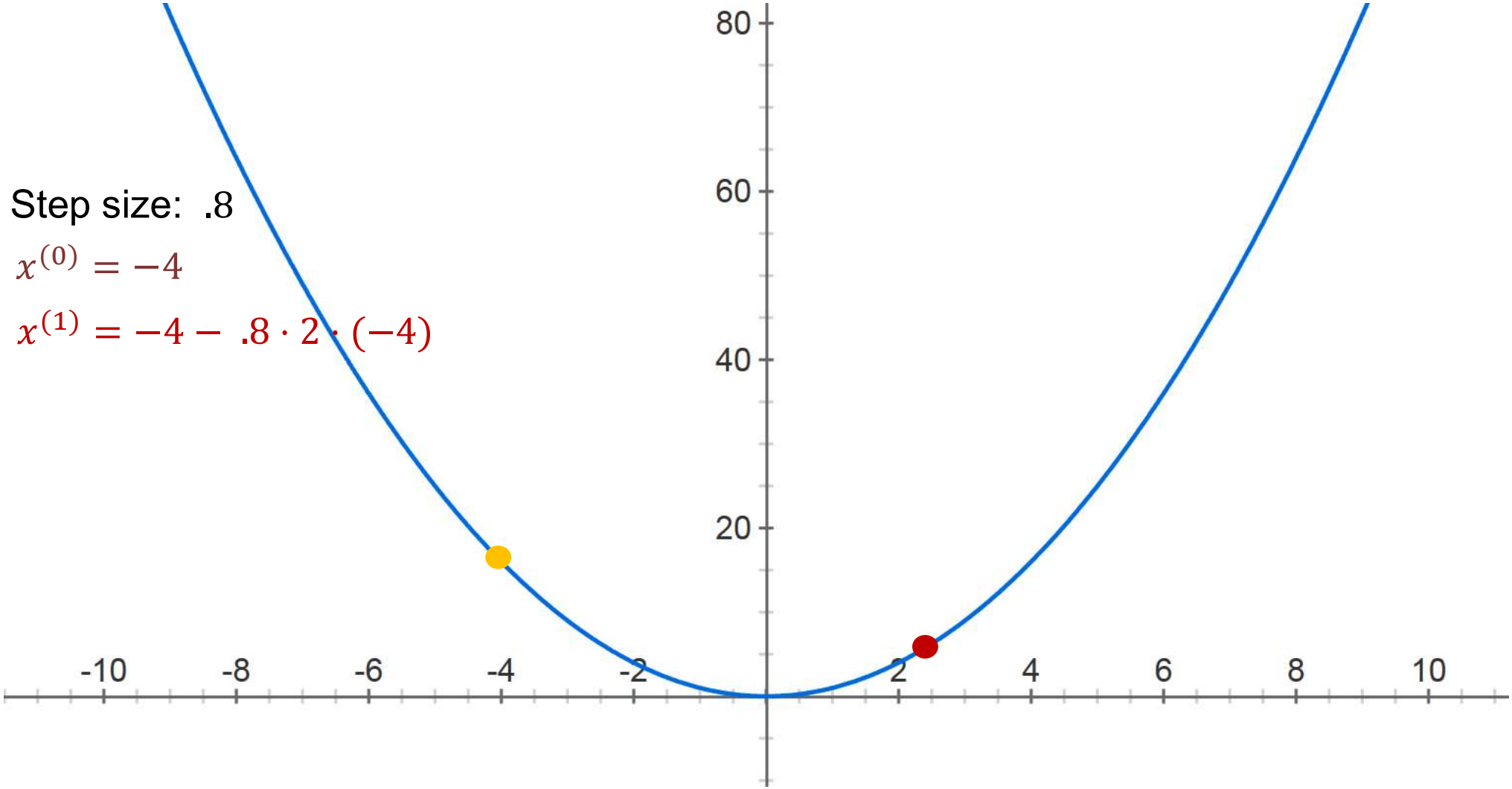
Minimizing Functions

$f(x) = x^2$

Step size: .8

$x^{(0)} = -4$

$x^{(1)} = -4 - .8 \cdot 2 \cdot (-4)$



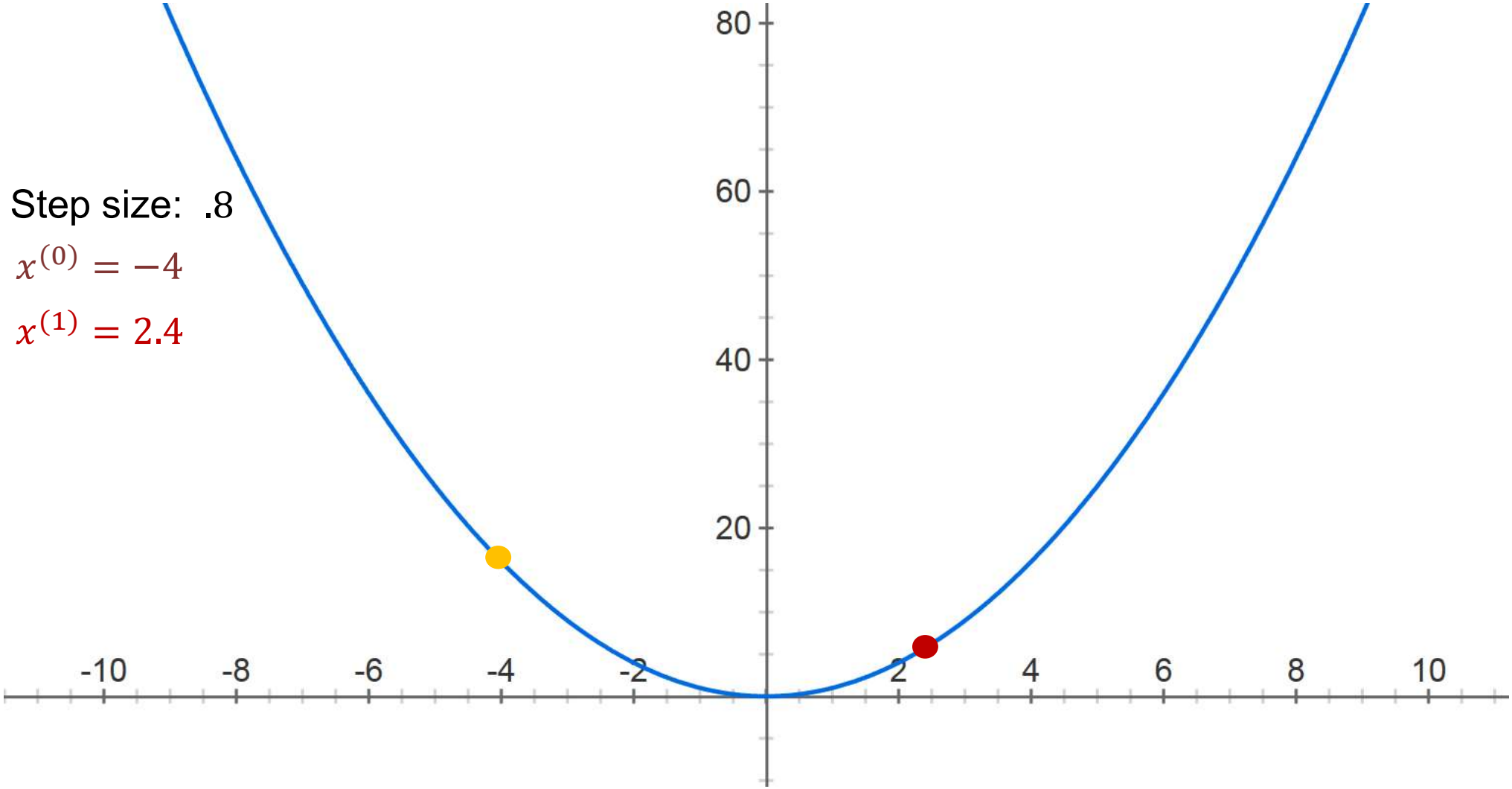
Minimizing Functions

$f(x) = x^2$

Step size: .8

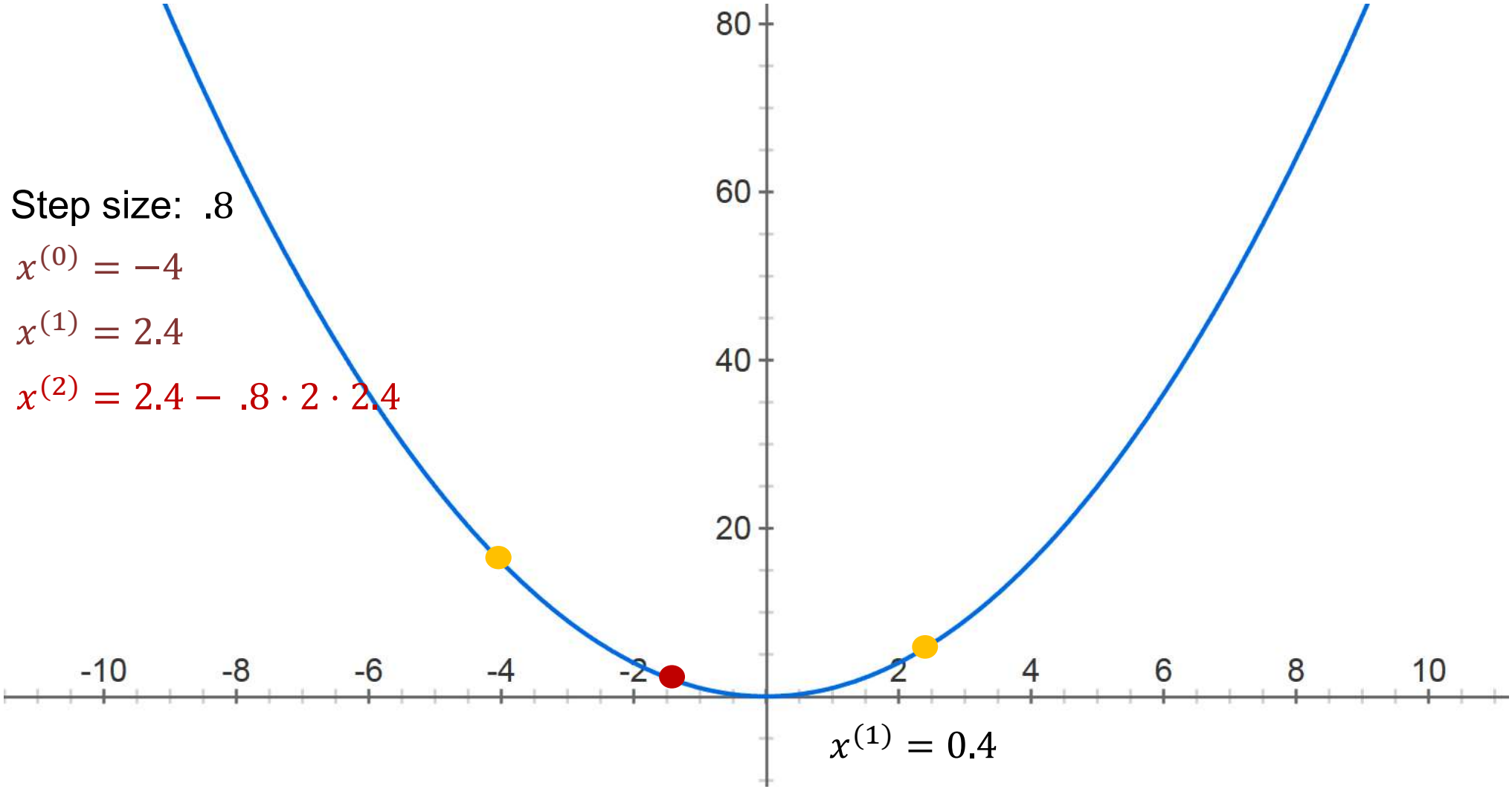
$x^{(0)} = -4$

$x^{(1)} = 2.4$



Minimizing Functions

$f(x) = x^2$



Minimizing Functions

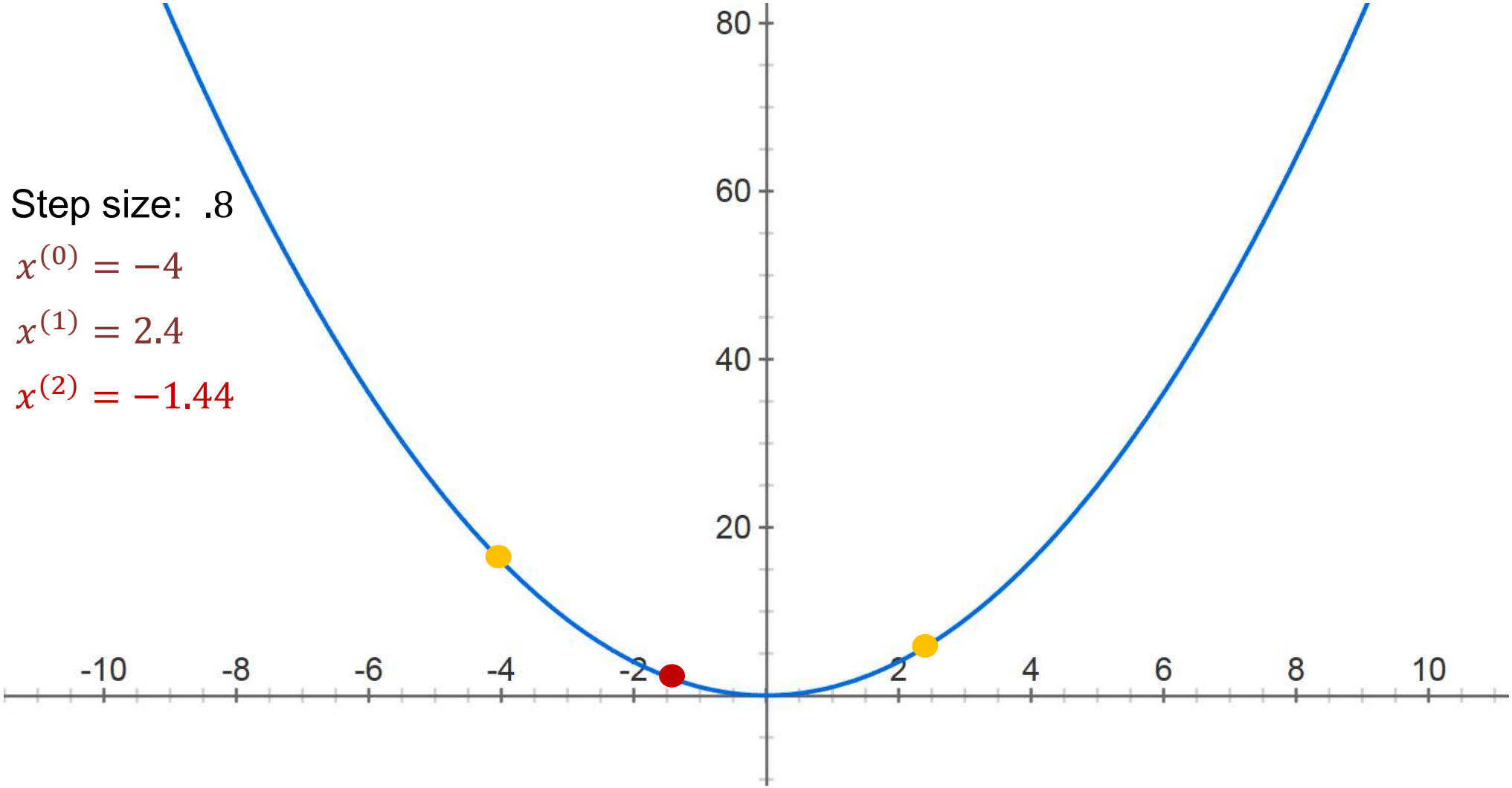
$f(x) = x^2$

Step size: .8

$x^{(0)} = -4$

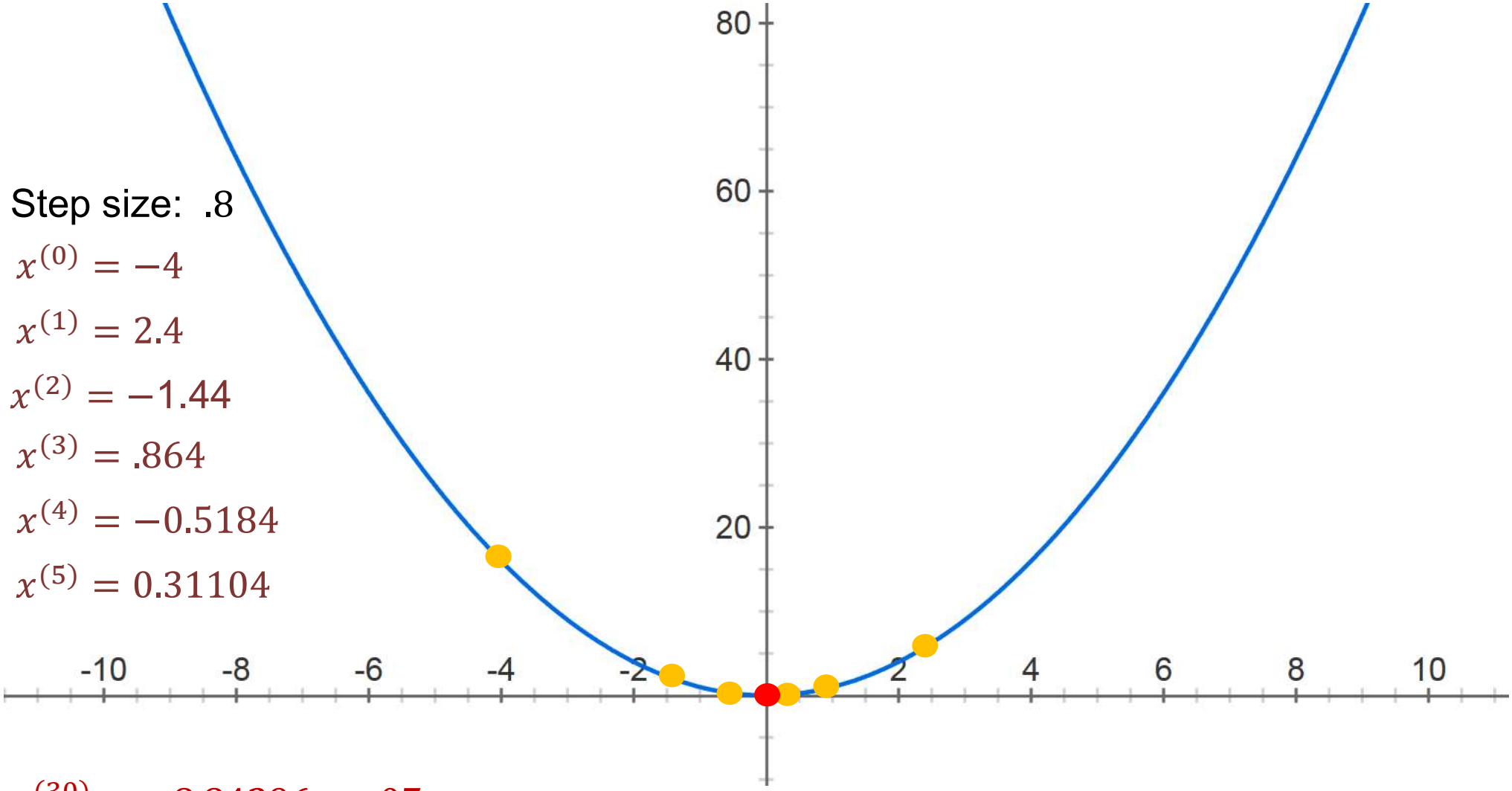
$x^{(1)} = 2.4$

$x^{(2)} = -1.44$



Minimizing Functions

$f(x) = x^2$



Step size: .8

$x^{(0)} = -4$

$x^{(1)} = 2.4$

$x^{(2)} = -1.44$

$x^{(3)} = .864$

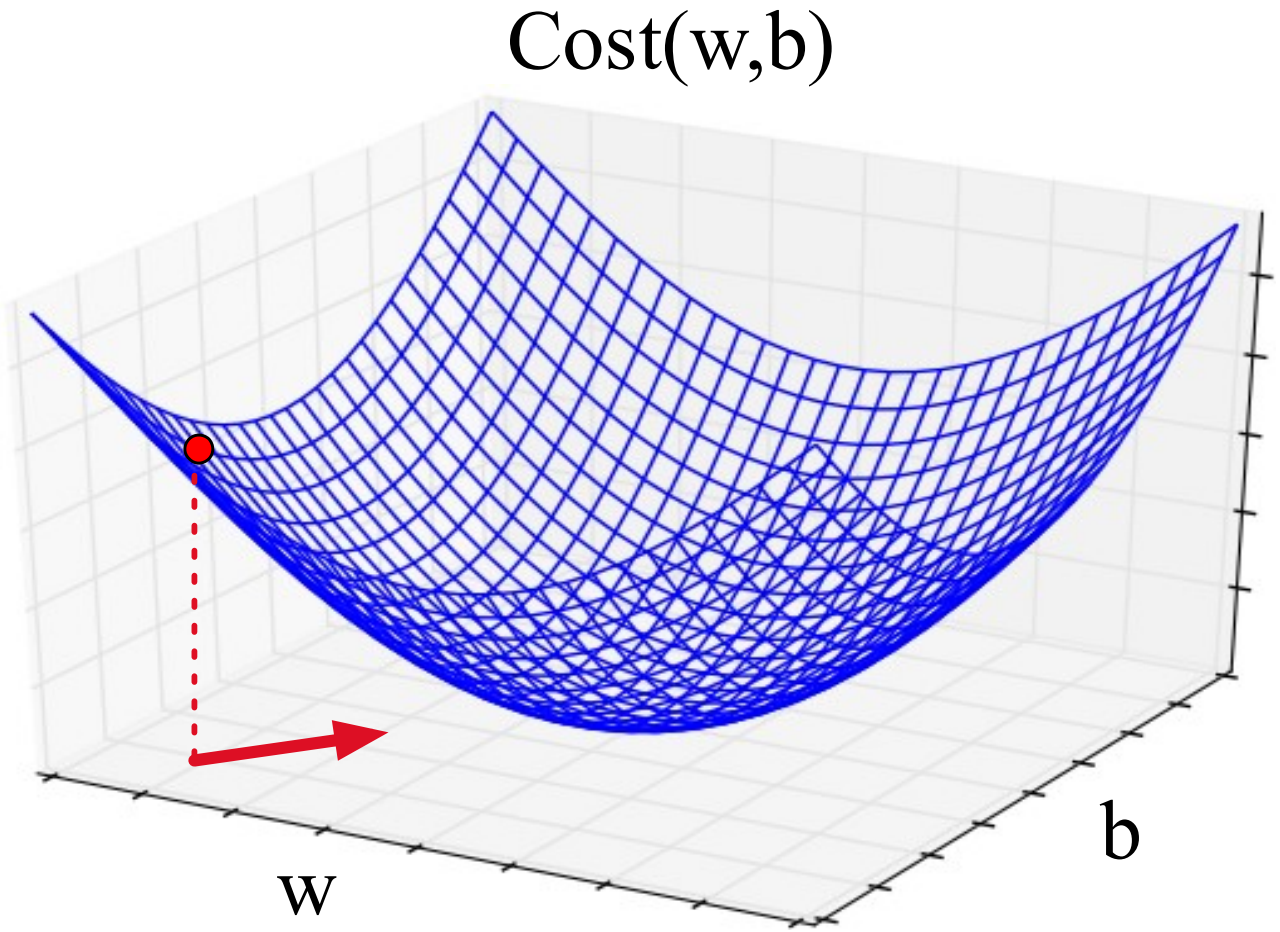
$x^{(4)} = -0.5184$

$x^{(5)} = 0.31104$

$x^{(30)} = -8.84296e - 07$

Gradients: Visualized

Visualizing the gradient vector at the **red** point



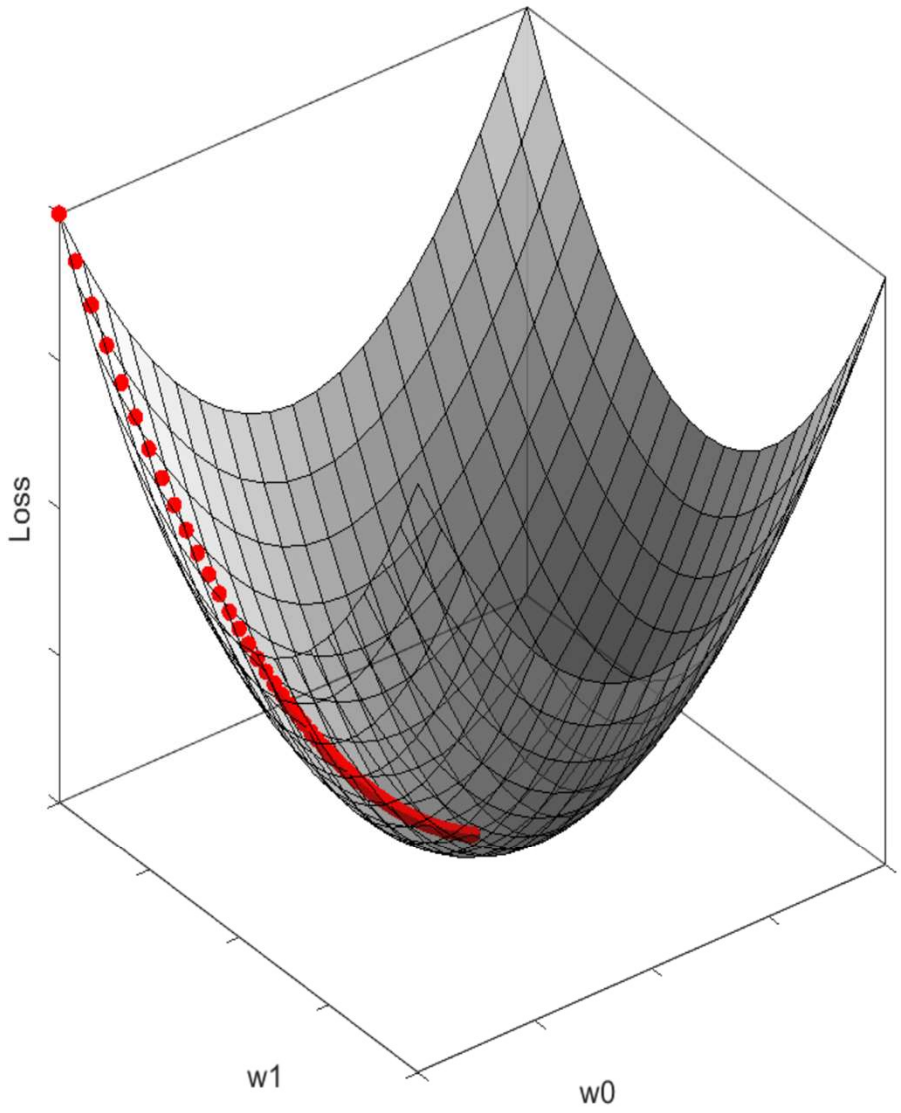
Gradients and Learning Rate / Step

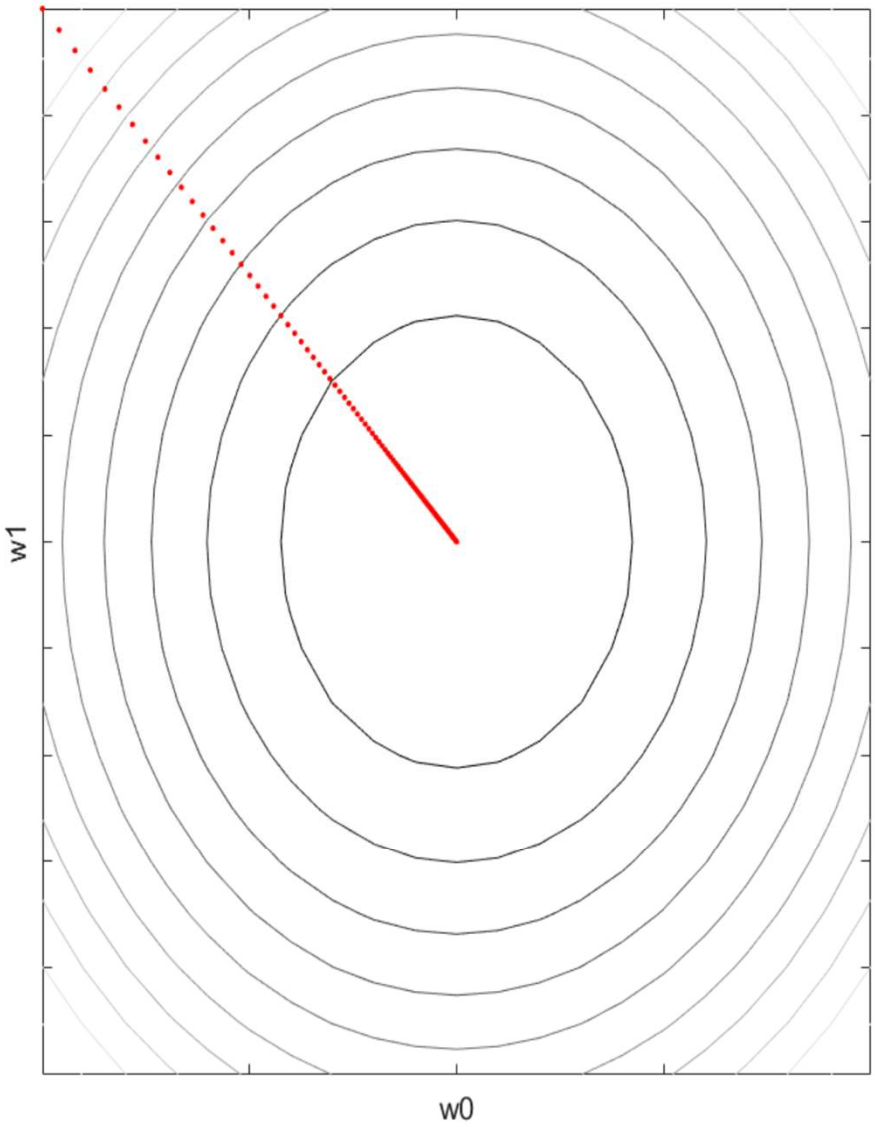
- The value of the gradient (slope in our example) $\frac{d}{dw} L(f(x;w), y)$ weighted by a **learning rate / step** η

- Higher learning rate means move **w** faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)$$

Gradient Descent





Gradients and Gradient Descent

Gradient Descent Algorithm:

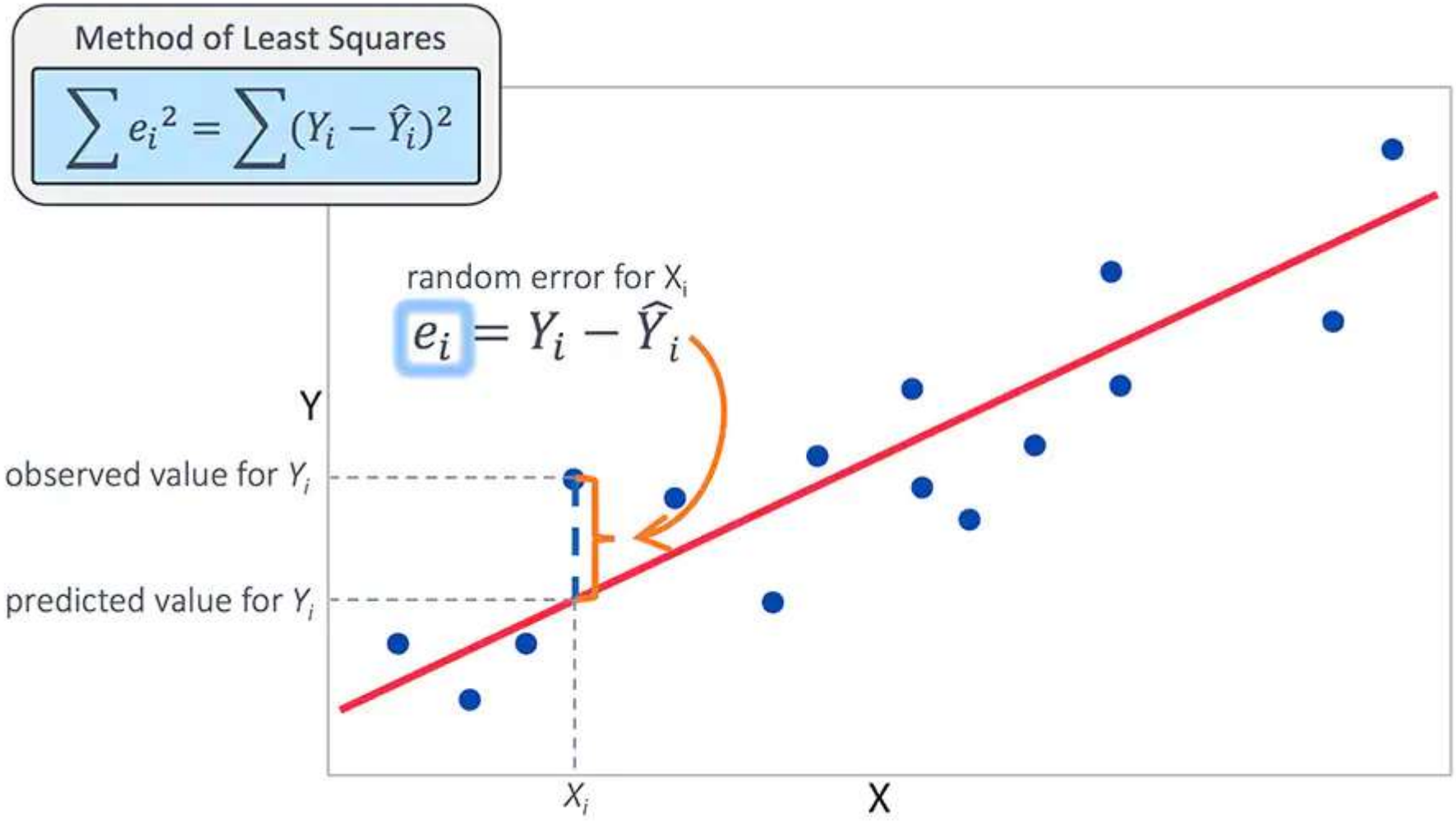
- **Pick an initial point x_0**
- **Iterate until convergence**

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

where γ_t is the t^{th} step size (sometimes called learning rate)

When to stop?

Linear Regression Using Least-Squares



The goal is to find the line y = ax + b that minimizes the amount of error.

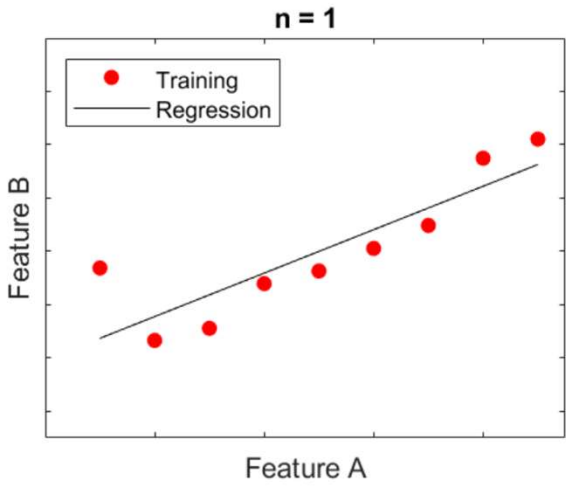
Source: https://www.jmp.com/en_us/statistics-knowledge-portal/what-is-multiple-regression/fitting-multiple-regression-model.html

Origins of ‘Regression’ Term

Source: https://en.wikipedia.org/wiki/Francis_Galton

Sir Francis Galton, an English polymath studied, among other things, heredity in humans. In one experiment he compared children height to their parent heights. He observed that children heights **regressed** towards the average height of an adult.

Univariate Linear Regression



Real function: $y = w_1 * x + w_0$

Hypothesis / model: $h_{\boldsymbol{w}}(x) = w_1 * x + w_0$

where $\boldsymbol{w} = \langle w_0, w_1 \rangle$ **are coefficients / weights.**

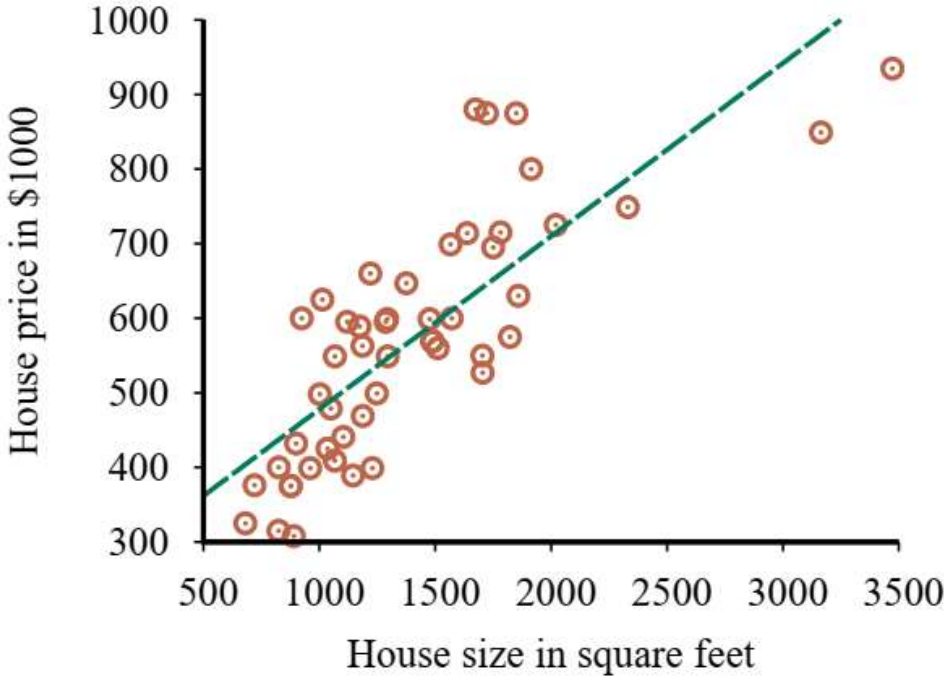
Squared-error loss function:

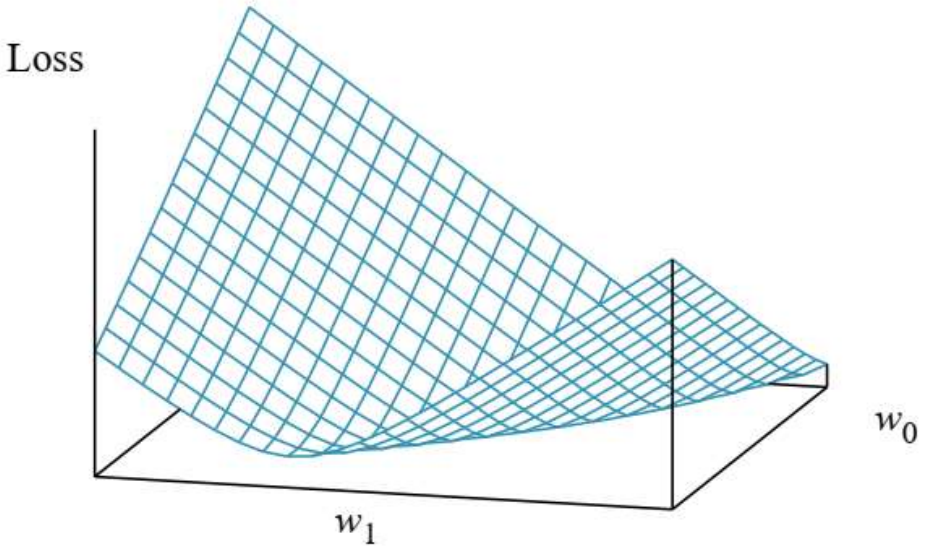
$$Loss(h_{\boldsymbol{w}}) = \sum_{j=1}^{N} \left(y_j - h_{\boldsymbol{w}}(x_j)\right)^2 = \sum_{j=1}^{N} \left(y_j - (w_1 * x_j + w_0)\right)^2$$

We want to find $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{argmin}\, Loss(h_{\boldsymbol{w}})$:

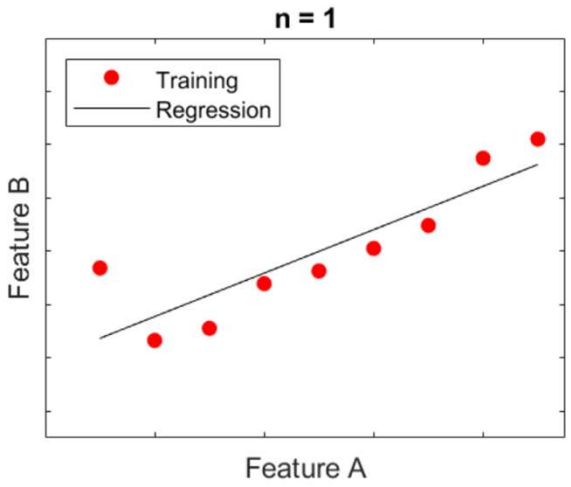
solve $\frac{\partial Loss(h_{\boldsymbol{w}})}{\partial w_0} = 0$, $\frac{\partial Loss(h_{\boldsymbol{w}})}{\partial w_1} = 0$

Weight / Parameter Space





Linear Regression: Gradient Descent



Given a squared-error loss function:

$$Loss(h_{\boldsymbol{w}}) = \sum_{j=1}^{N} \left(y_j - h_{\boldsymbol{w}}(x_j)\right)^2 = \sum_{j=1}^{N} \left(y_j - (w_1 * x_j + w_0)\right)^2$$

We want to find $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{argmin}\, Loss(h_{\boldsymbol{w}})$:

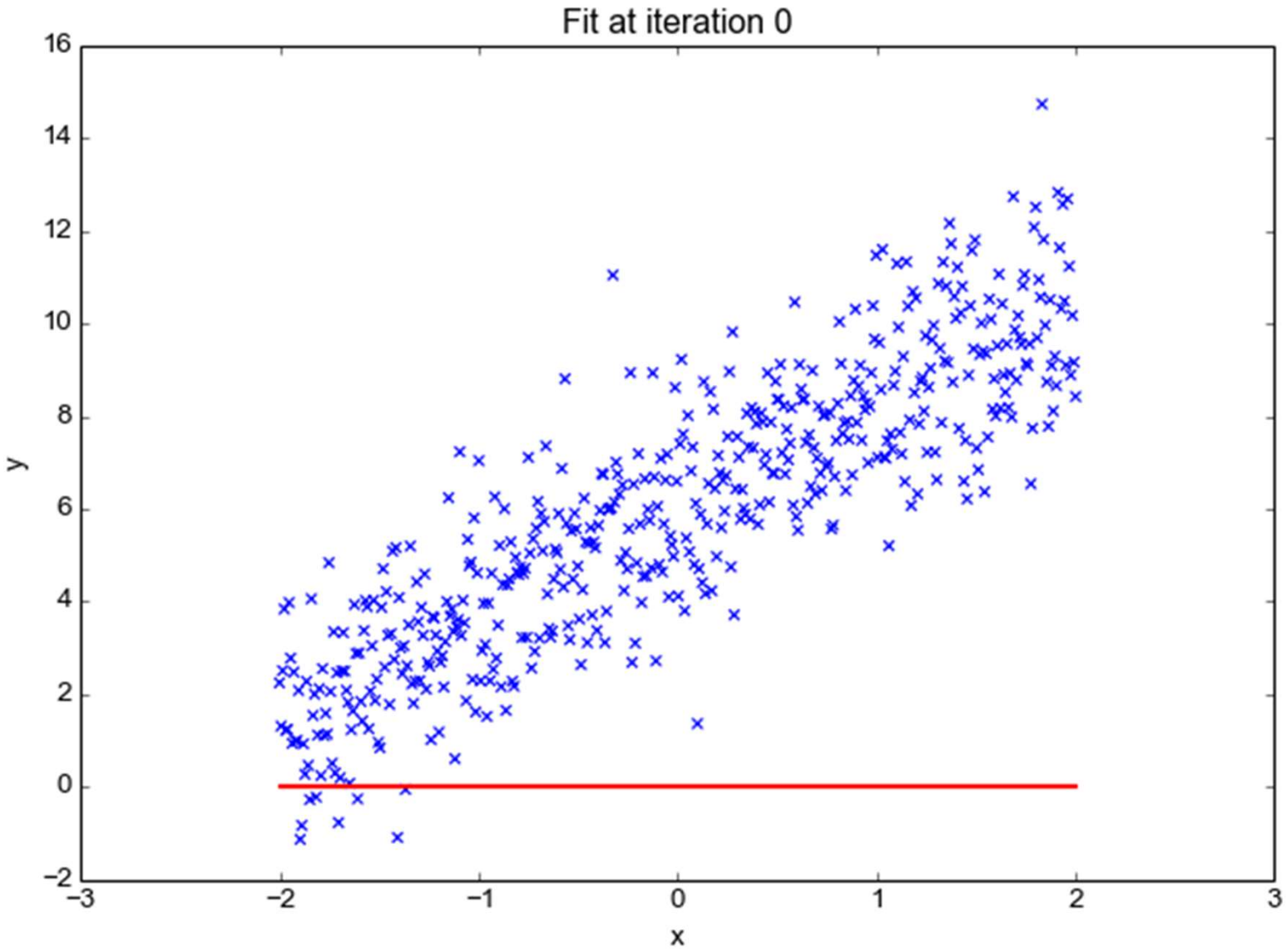
solve $\frac{\partial Loss(h_{\boldsymbol{w}})}{\partial w_0} = 0\,, \frac{\partial Loss(h_{\boldsymbol{w}})}{\partial w_1} = 0$

Sometimes these will be hard (or impossible) to solve. Gradient descent technique can be instead:

w ← **any point in the parameter space**
while not converged **do**
for each w_i **in w do** $w_i \leftarrow w_i - \alpha * \frac{\partial Loss(\boldsymbol{w})}{\partial\, w_i}$

α **- step size / learning rate**

Linear Regression: Gradient Descent



Source: Andrew Ng

Stochastic Gradient Descent

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$, ..., $x^{(m)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$, ..., $y^{(m)}$

$\theta \leftarrow 0$

repeat til done

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

1. Optional (for reporting):
   Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
   Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
2. $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?
3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead

return θ

Learning Rate

The learning rate η is a **hyperparameter**

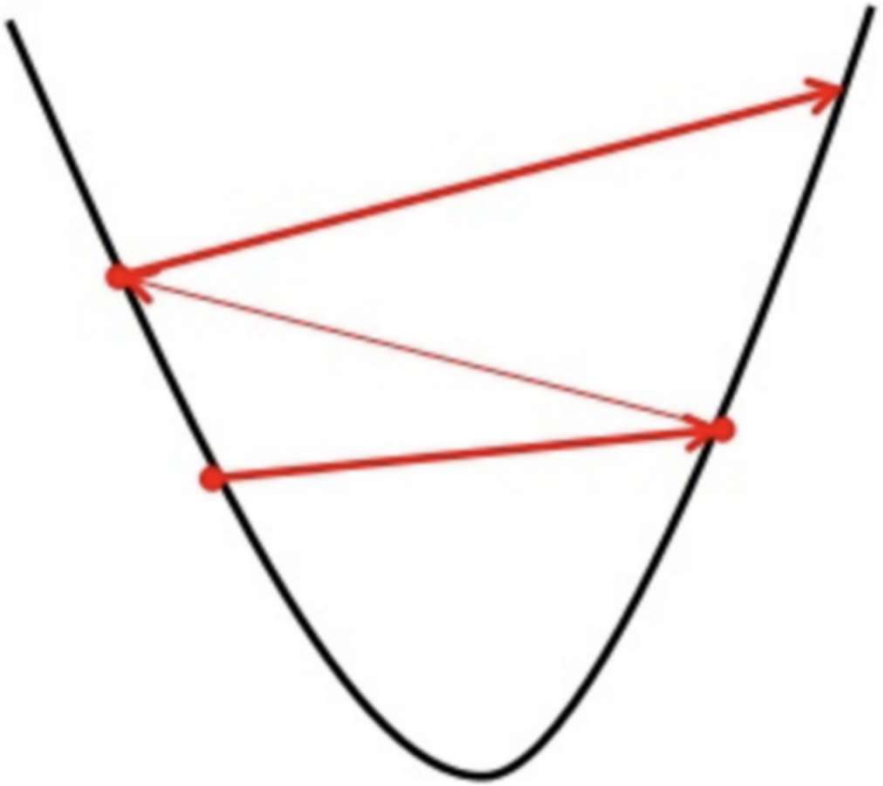
- too high: the learner will take big steps and overshoot
- too low: the learner will take too long

Hyperparameters:

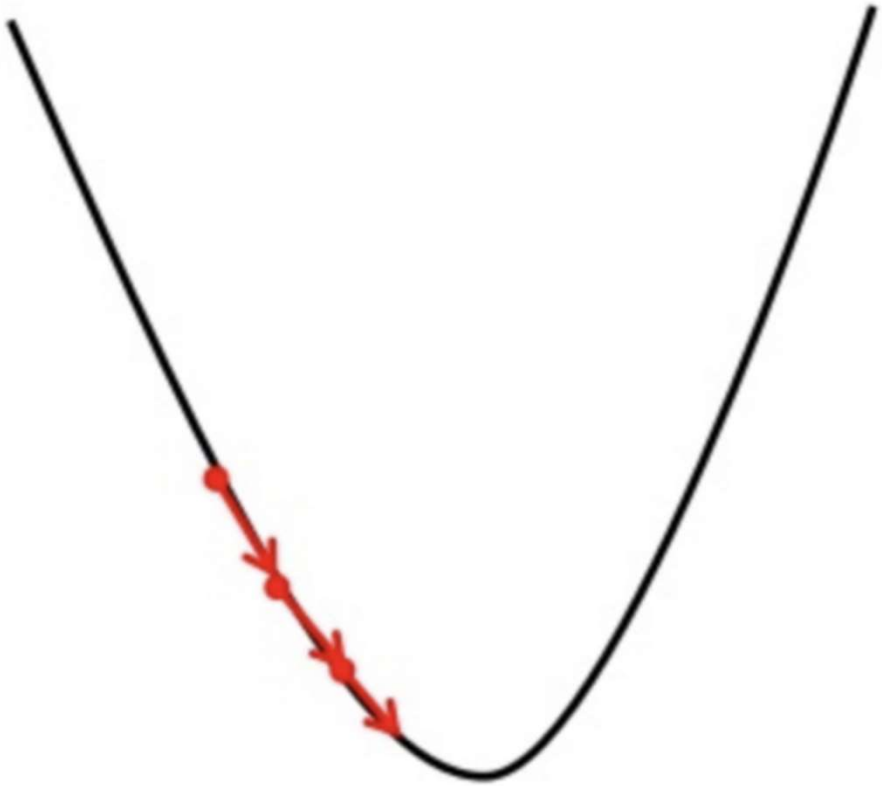
- Briefly, a special kind of parameter for an ML model
- Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.

Learning Rate / Step

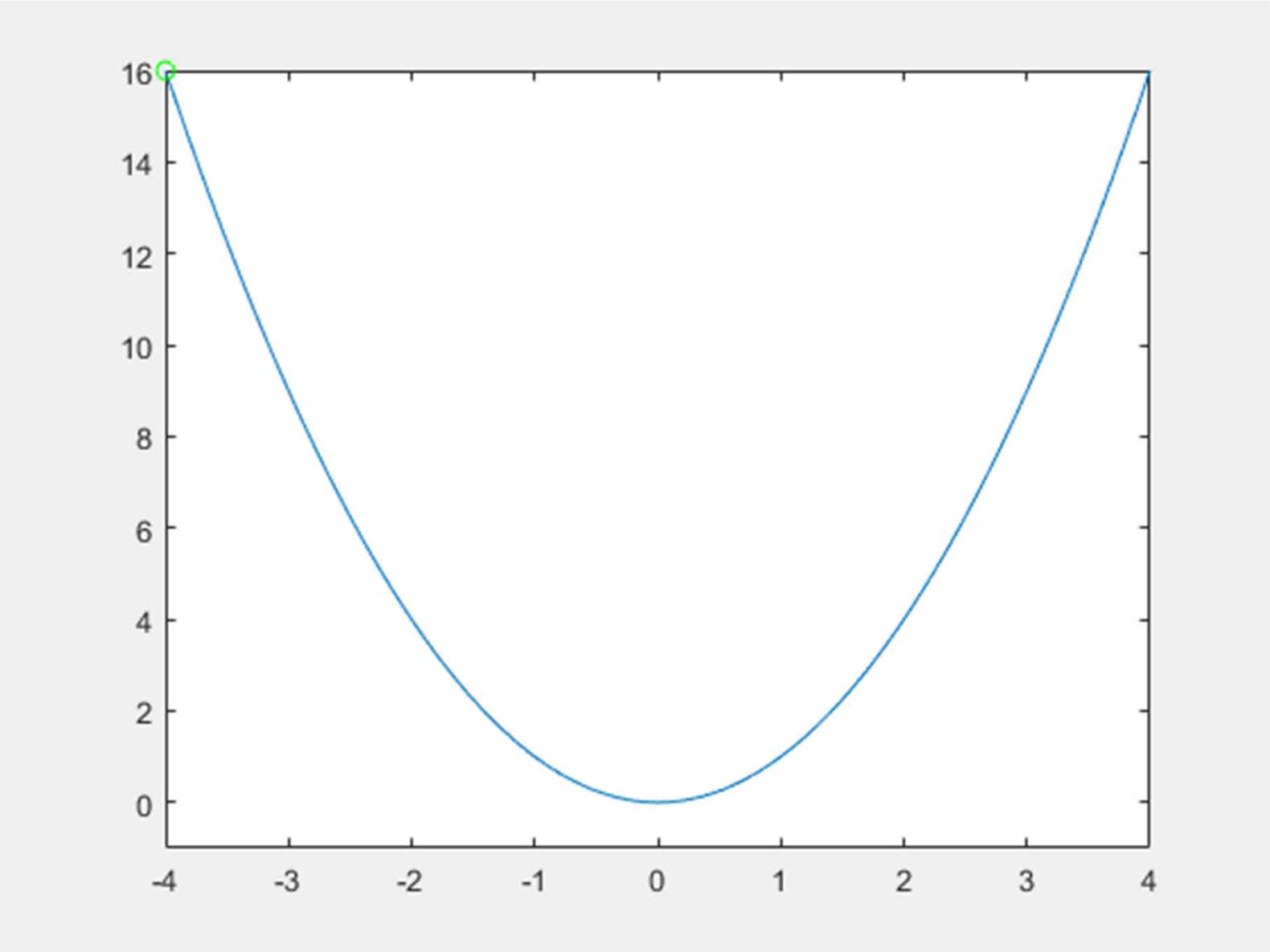
High Learning Rate / Step



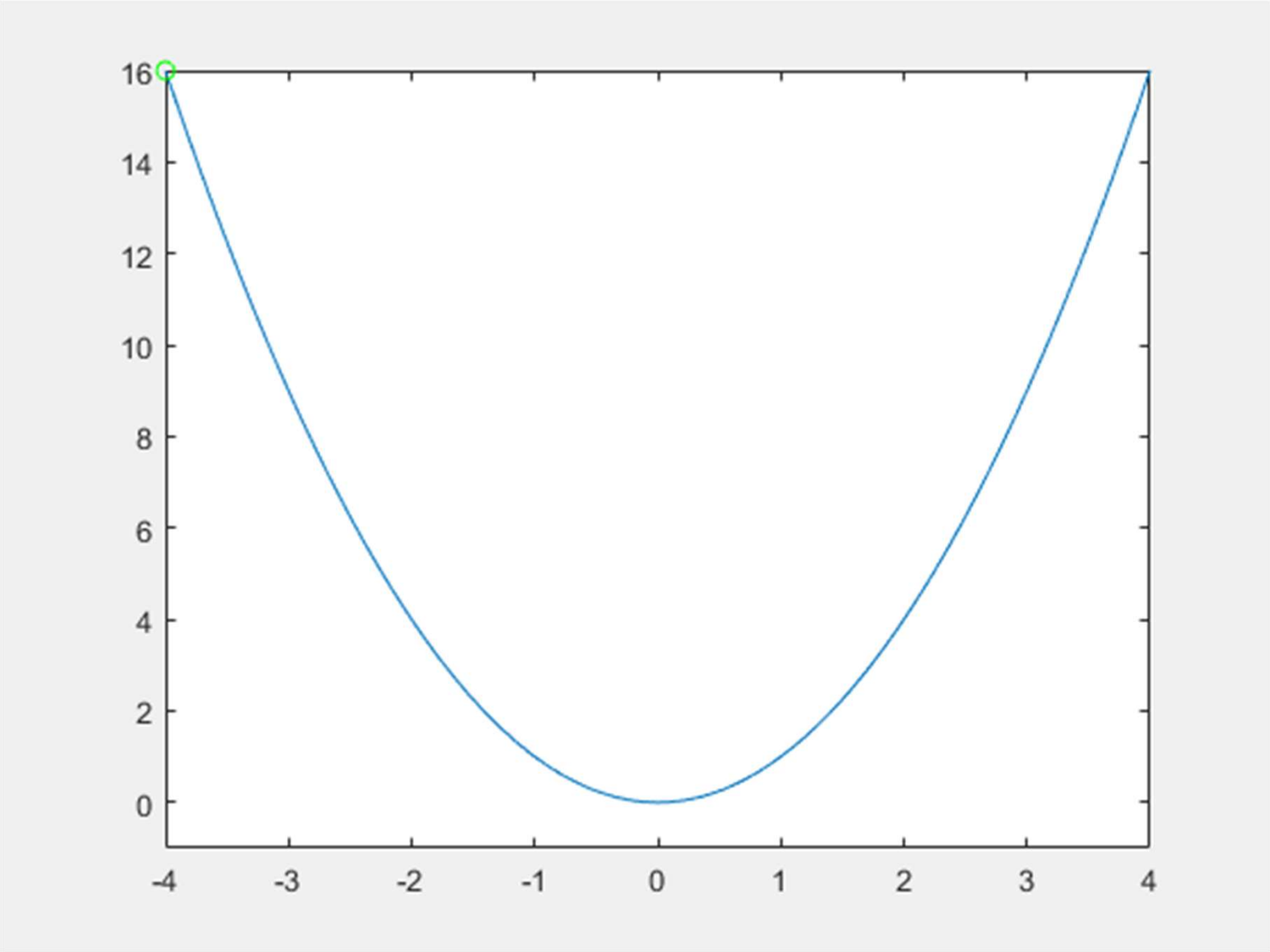
Low Learning Rate / Step



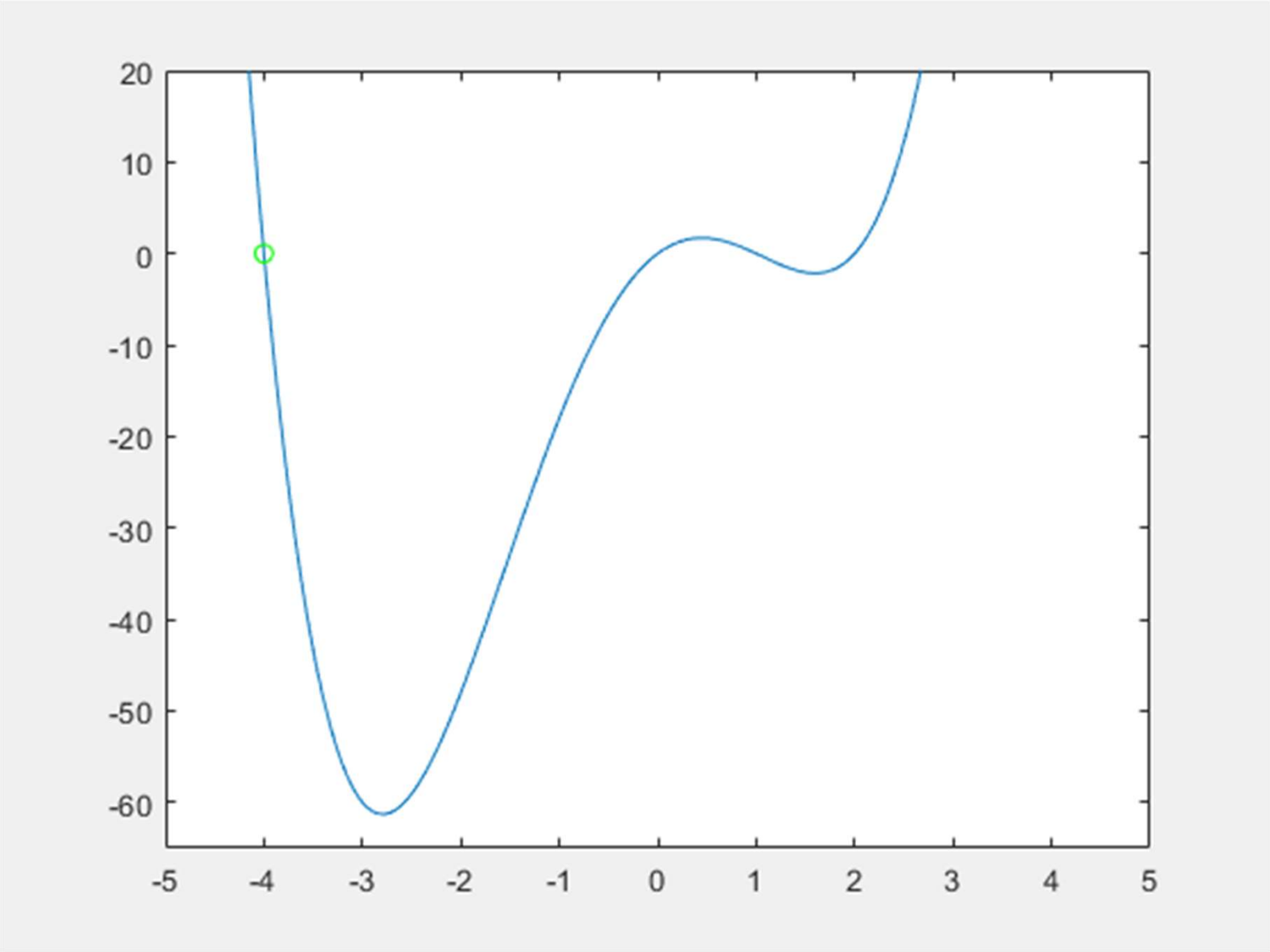
Rate / Step Size Matters



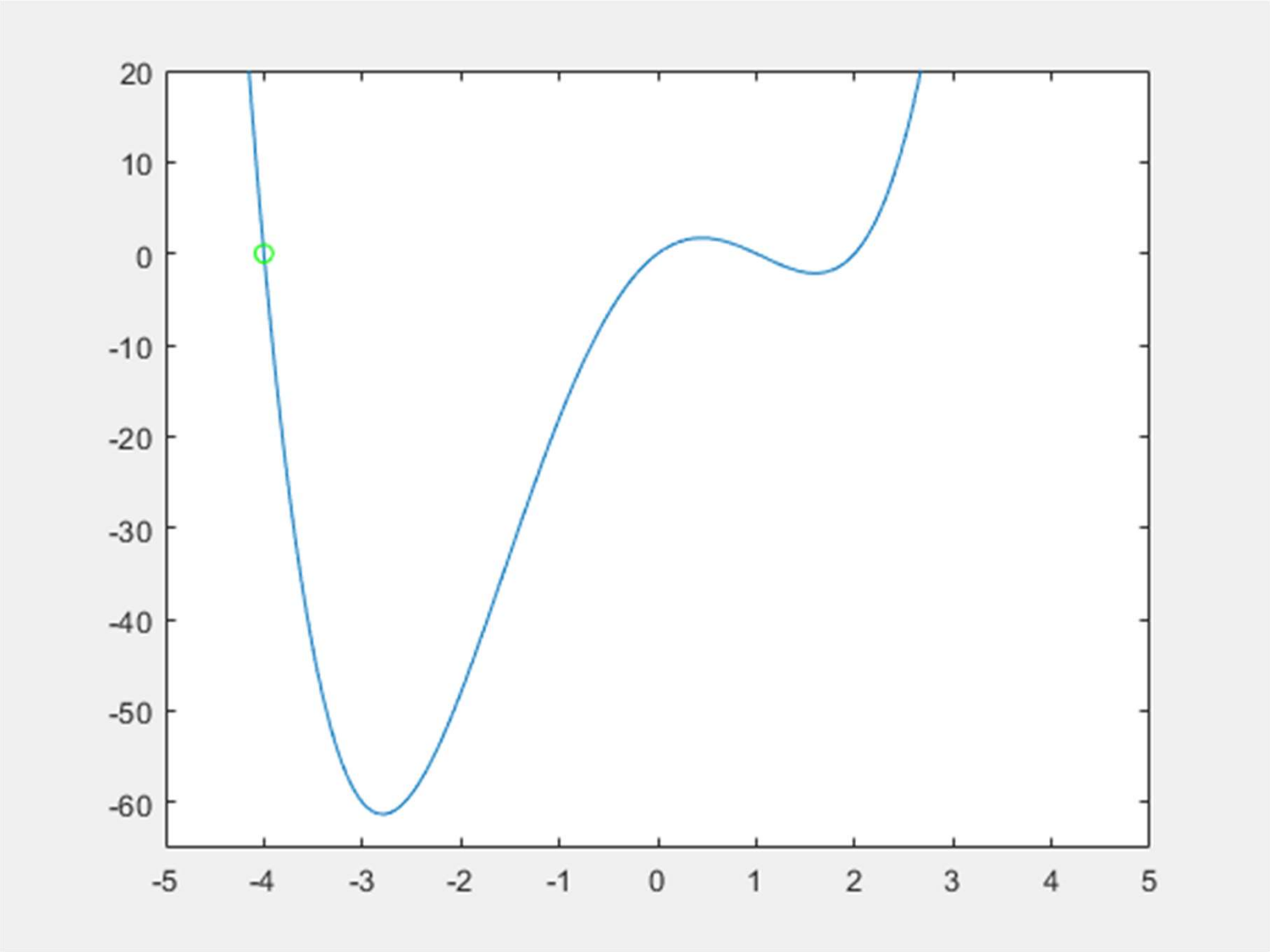
Rate / Step Size Matters



Rate / Step Size Matters

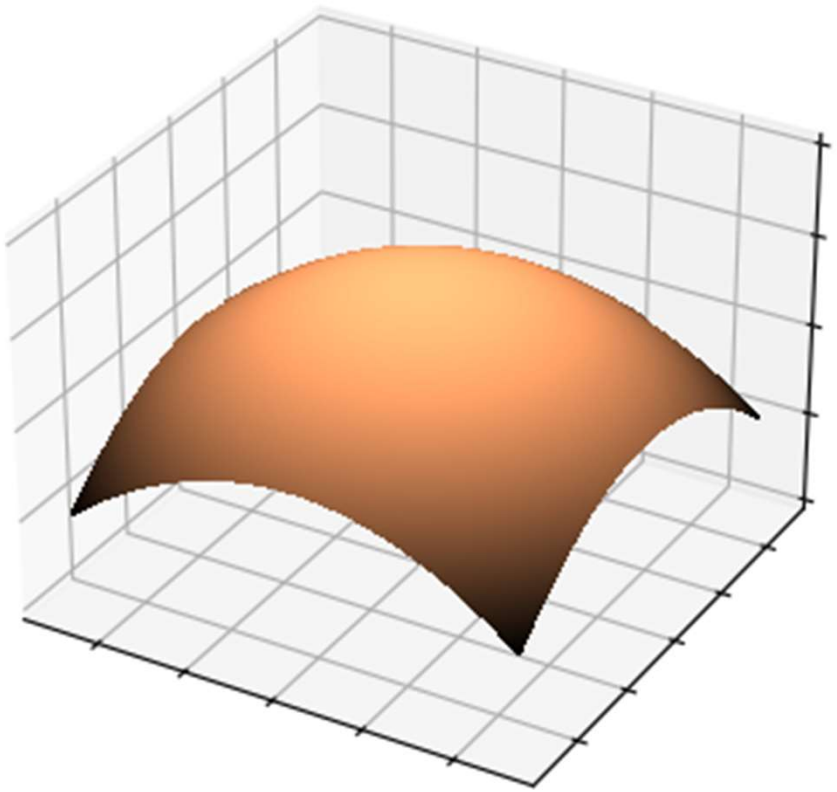


Rate / Step Size Matters



Concave vs. Non-Convex Functions

Concave Function



Non-Convex Function