Stochastic Hill Climbing, Gradients, and Free Lunches

Alexander G. Ororbia II

Biologically-Inspired Intelligent Systems

CSCI-633

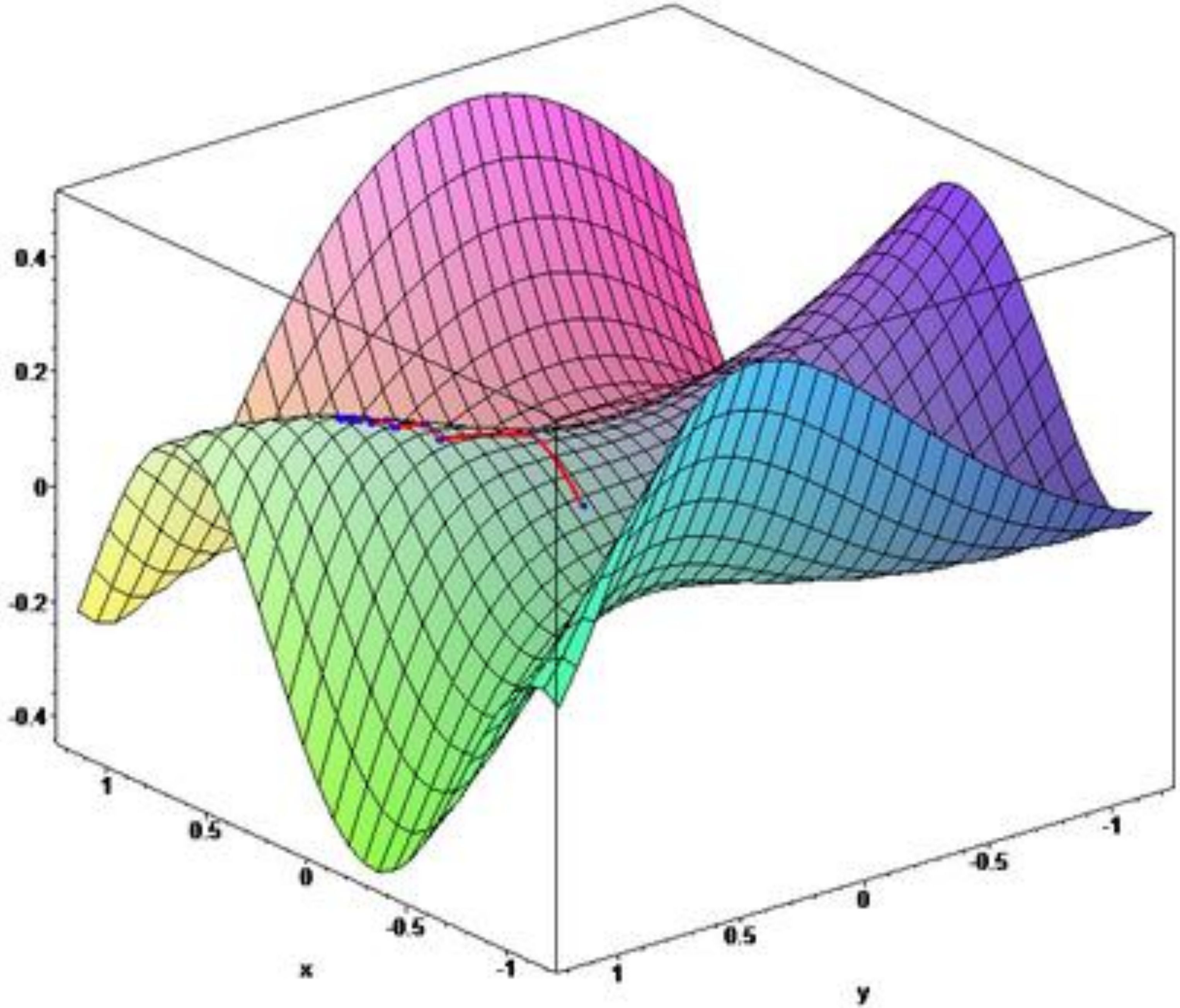
1/23/2024

Quick Logistic Note

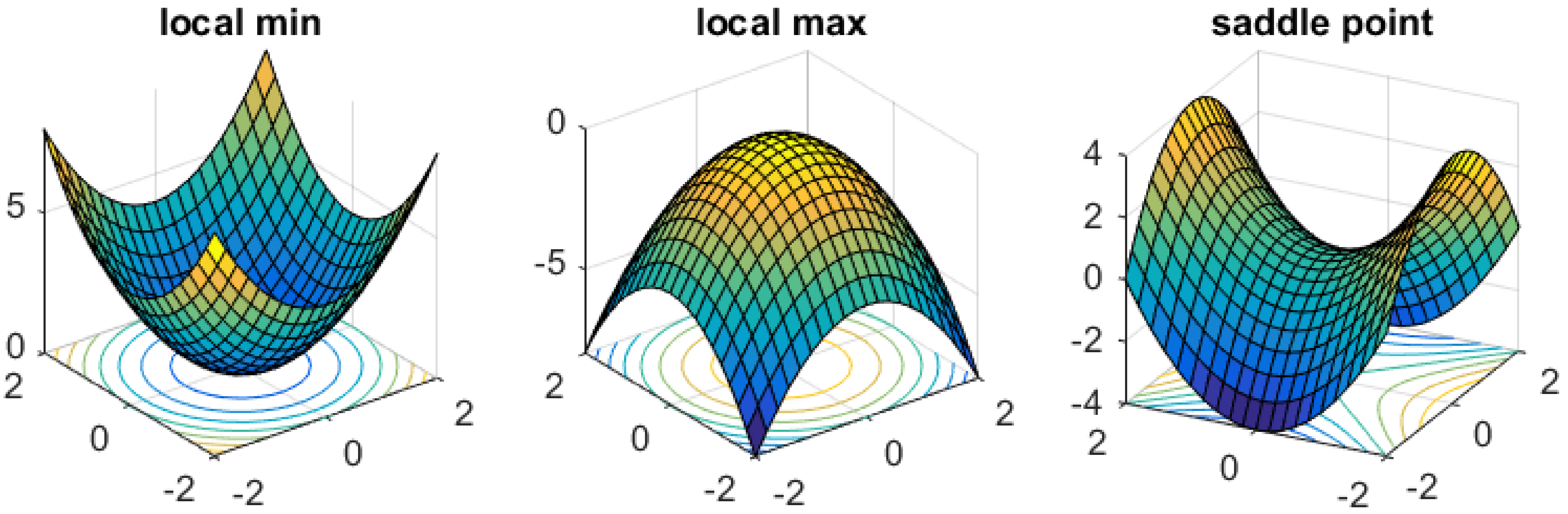
- Make sure you pick your teams by this Thursday evening! (or get random assignment)
  - We will load balance to get as many to size 3 as needed
- Start thinking of your semester project/final topic
  - Rubric is up
- Start searching for possible papers your team will be interested in presenting for the weekly talks
  - Can look at schedule and peruse textbook for things not covered
  - Papers must be published in quality venues/journals and be squarely about metaheuristic optimization

Why Optimization Again?

- Assume a state (or solution) with many variables
- Assume some function that you want to maximize/minimize value of
  - E.g. a “goodness” function
- Searching entire space is too complicated
  - Cannot evaluate every possible combination of variables
  - Function might be difficult to evaluate analytically

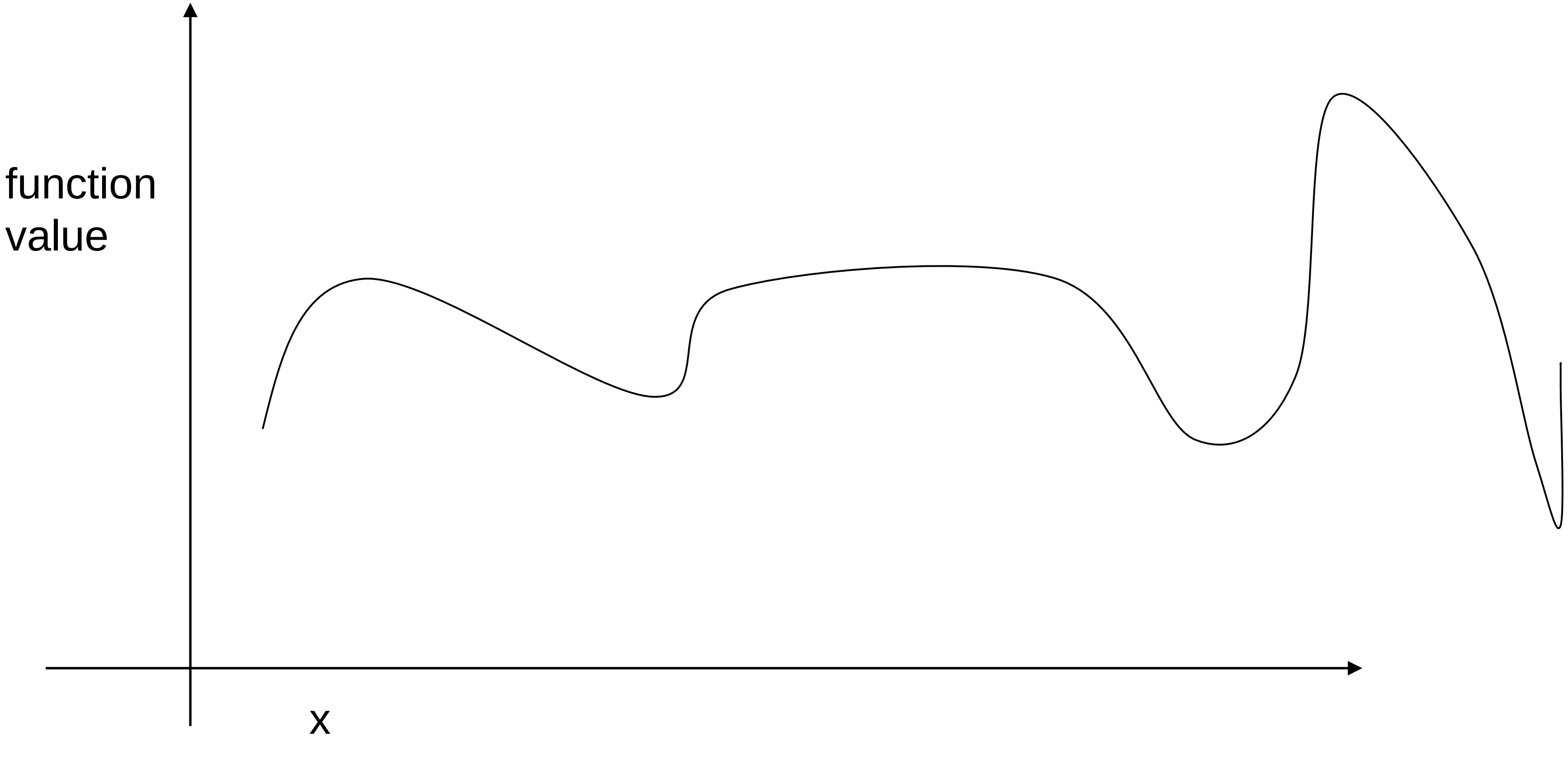

0.4
0.2
0
-0.2
-0.4
1
0.5
0
-0.5
-1
x
1
0.5
0
-0.5
-1
y

Problems!!


local min
5
0
2
0
-2
-2
0
2
local max
0
-5
2
0
-2
-2
0
2
saddle point
4
2
0
-2
-4
2
0
-2
-2
0
2

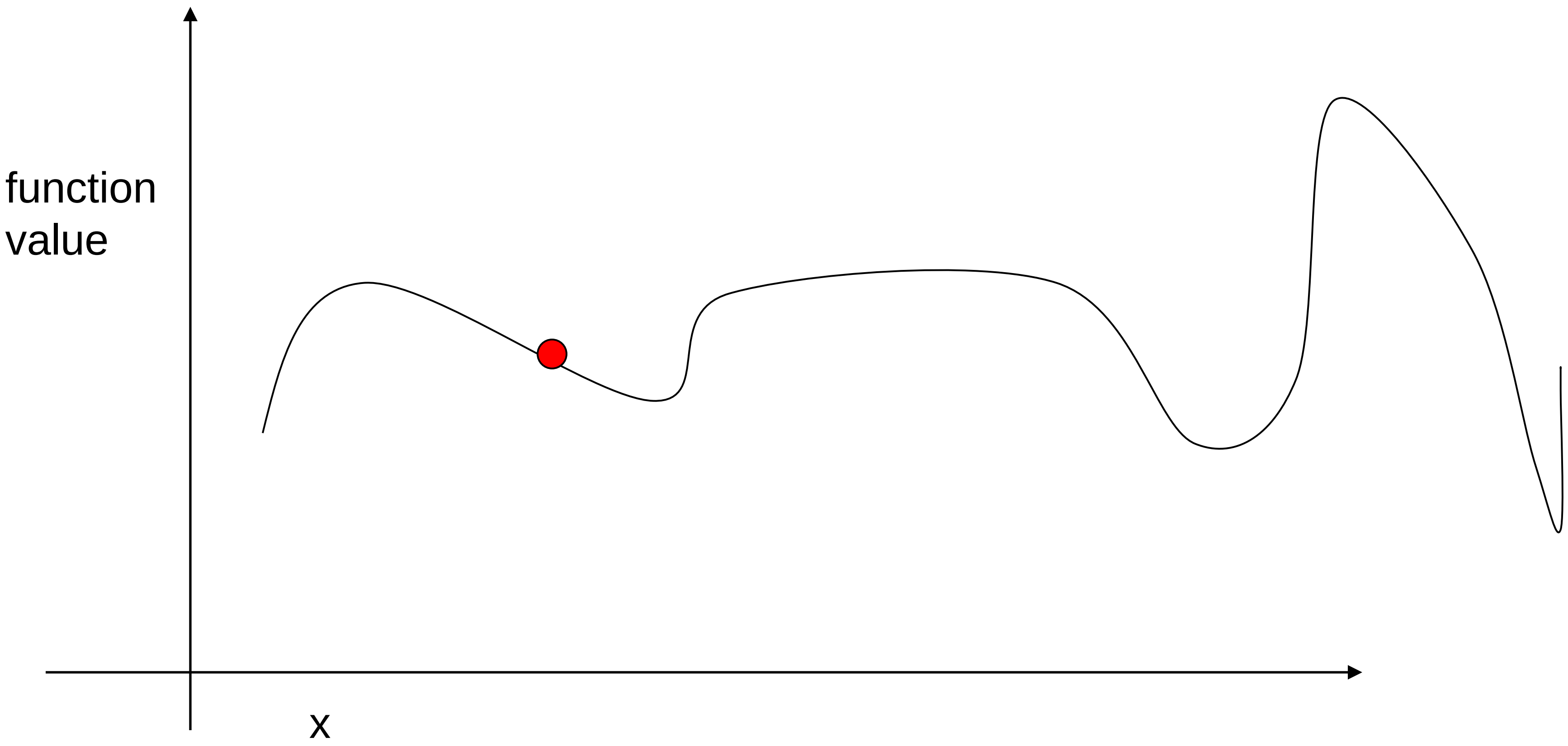
Simple Example: The Idealized Climb

- One dimension (typically use more):



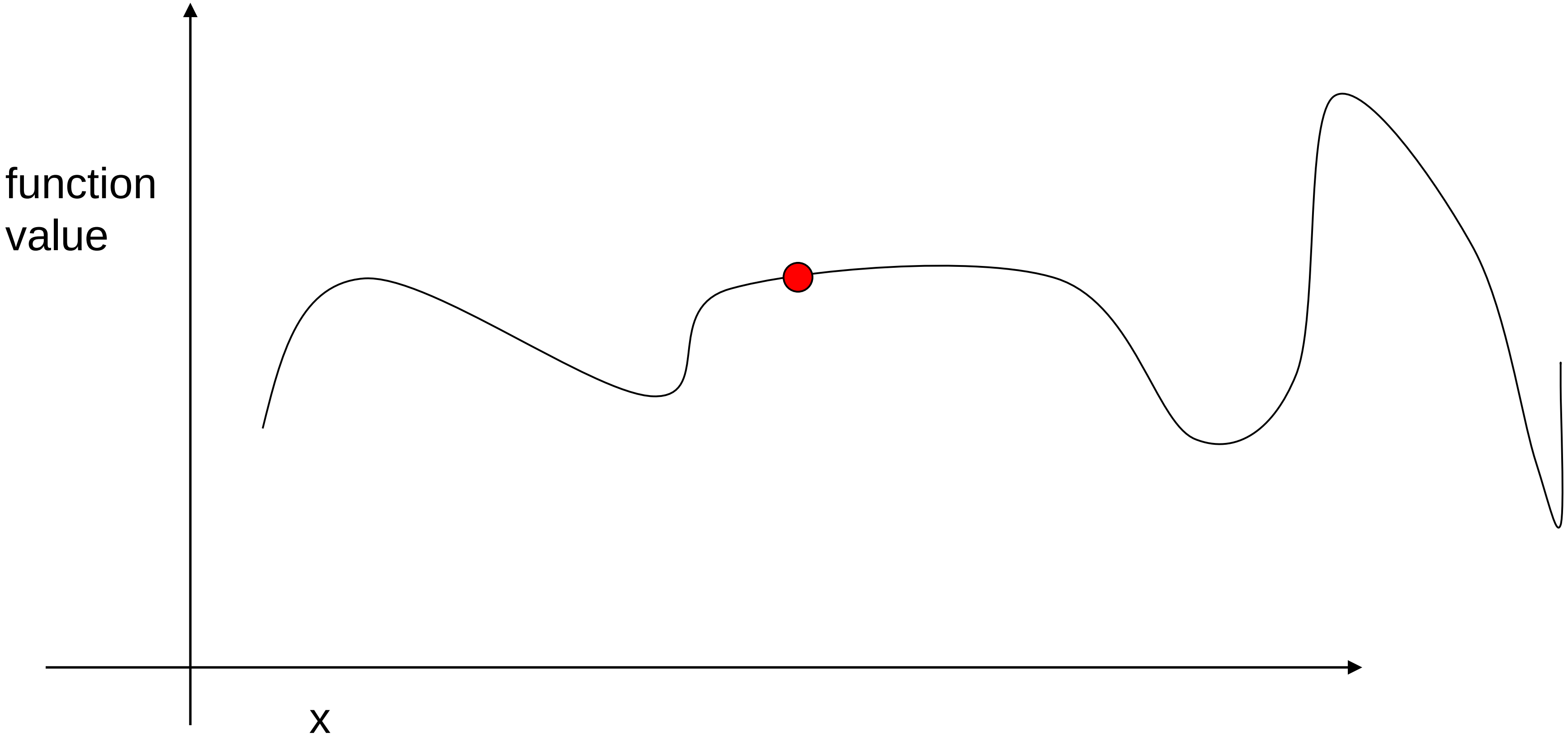
Simple Example: The Idealized Climb

- Start at a valid state, try to maximize



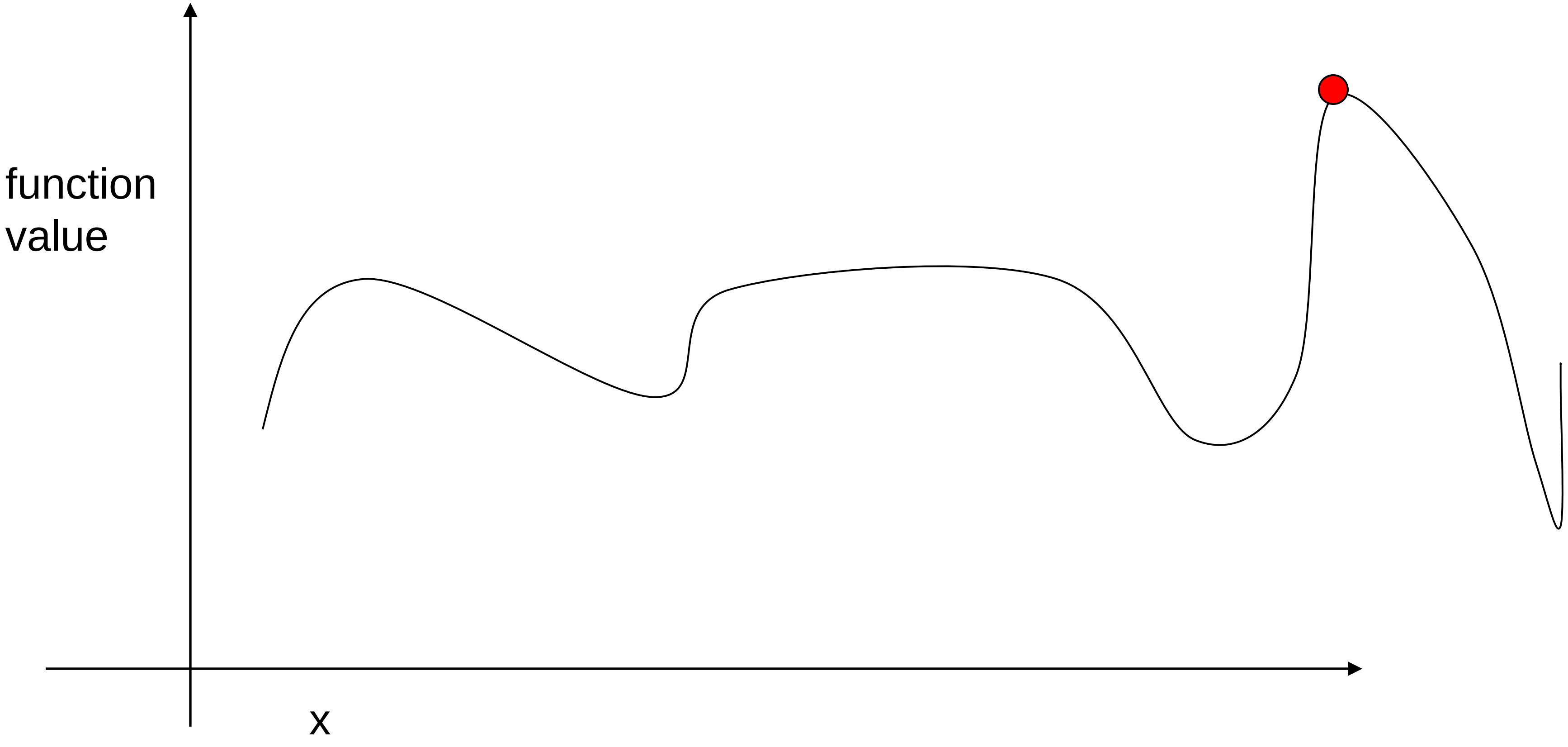
Simple Example: The Idealized Climb

- Move to better state



Simple Example: The Idealized Climb

- Try to find maximum



Stochastic Hill-Climbing Search

- Steepest ascent, but random selection/generation of neighbor candidates/positions (*<u>variations</u>*: first-choice hill climbing, **random-restart hill-climbing**)

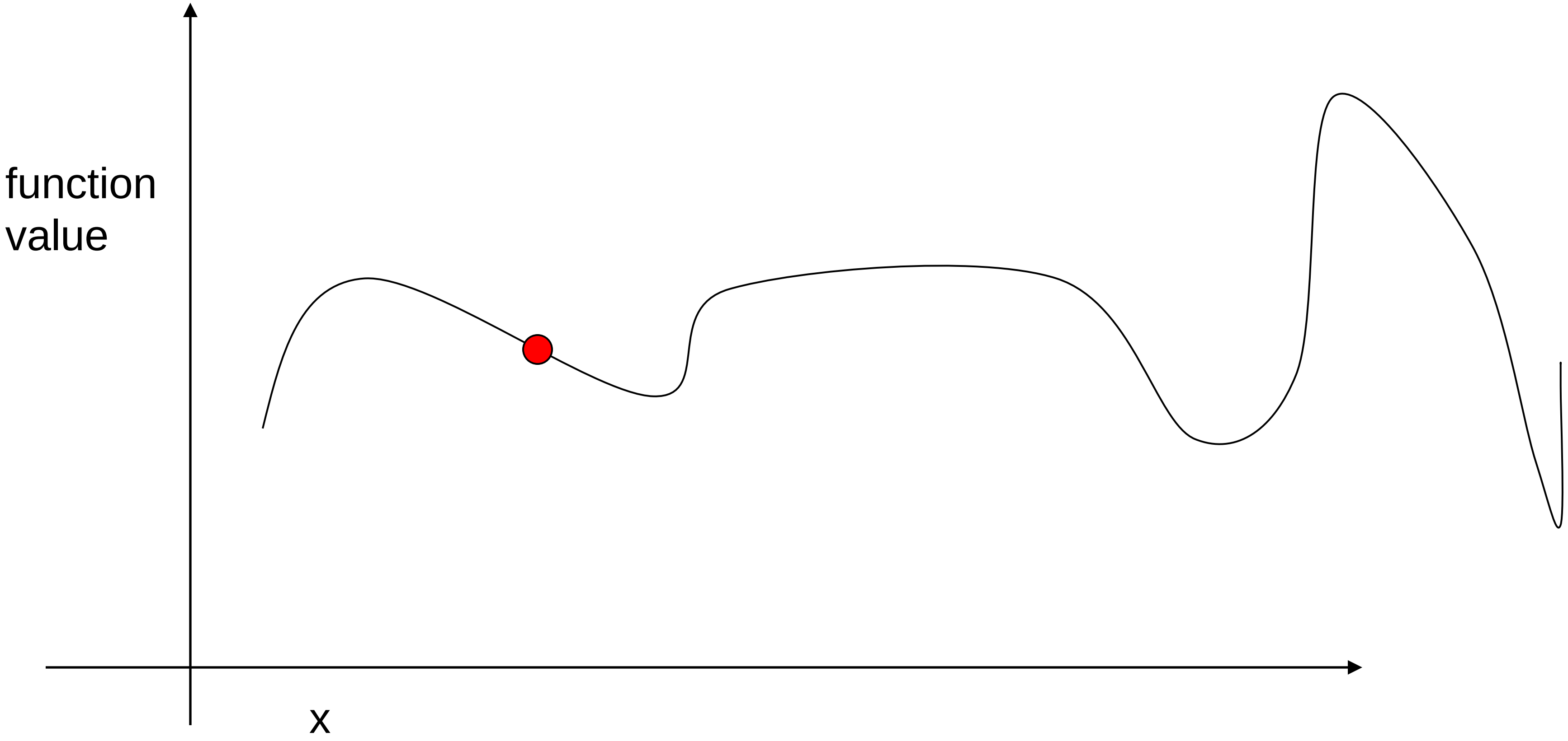
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Generate a random sample/set of neighbors around *current*, choose highest-valued among them

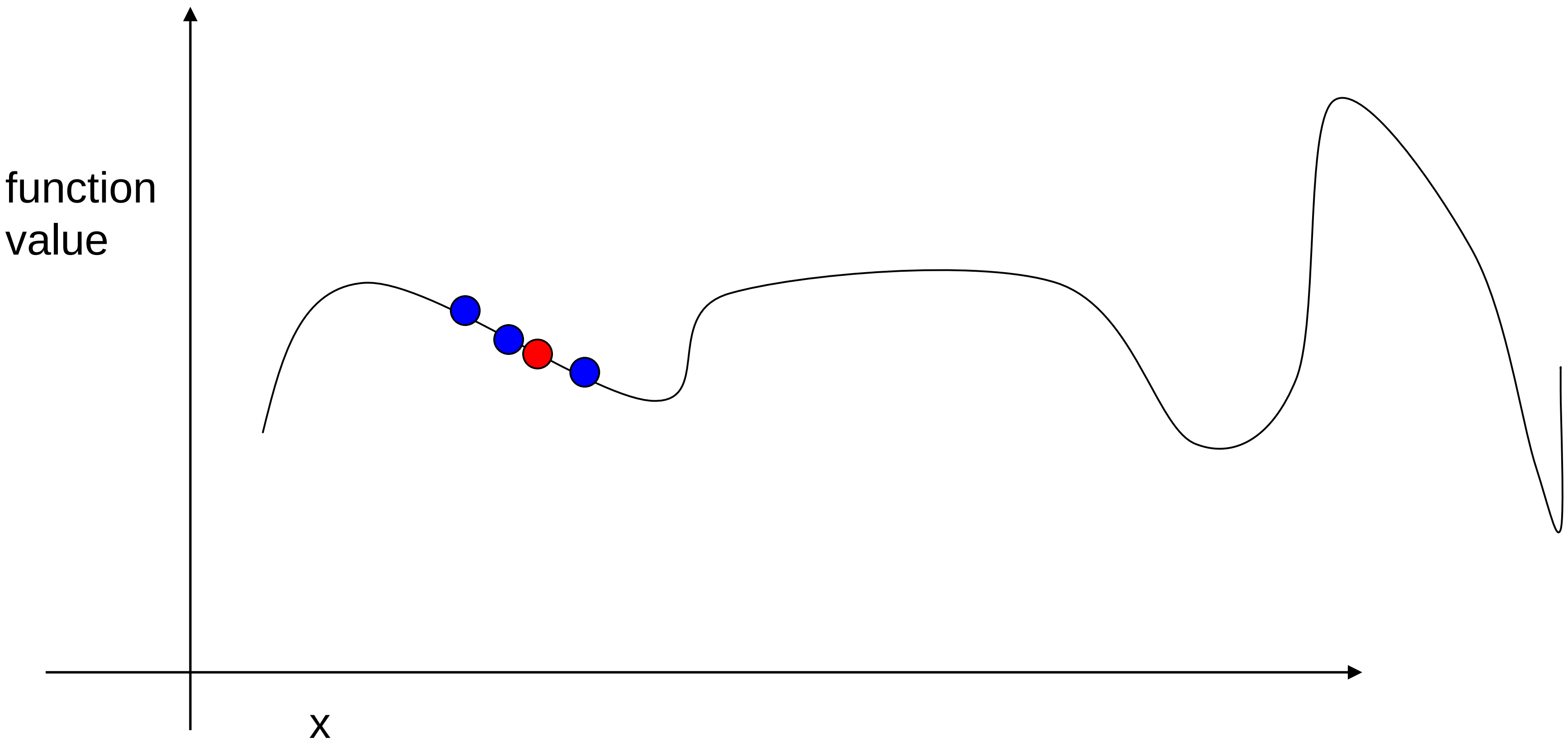
Simple Example

- Random Starting Point



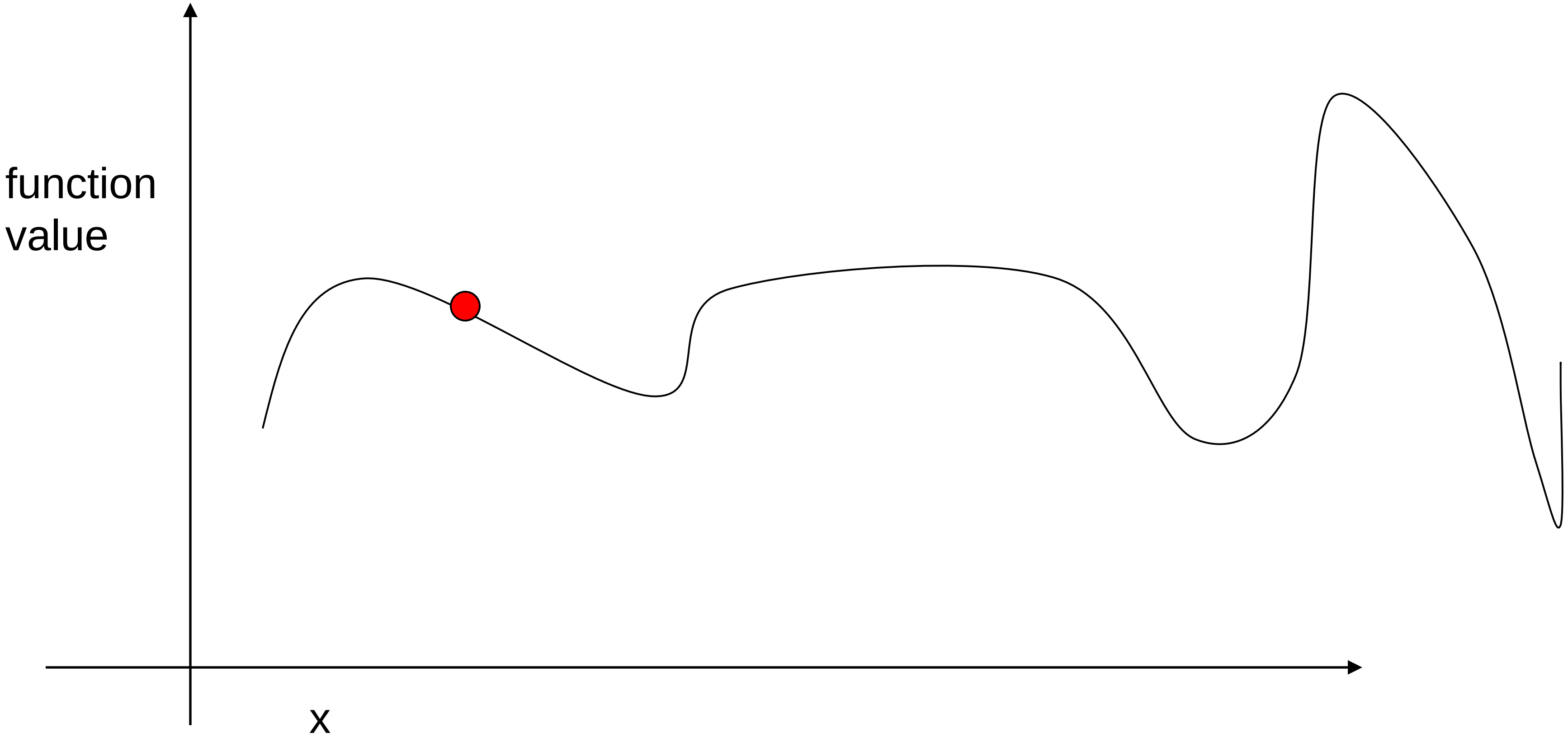
Simple Example

- Three random steps



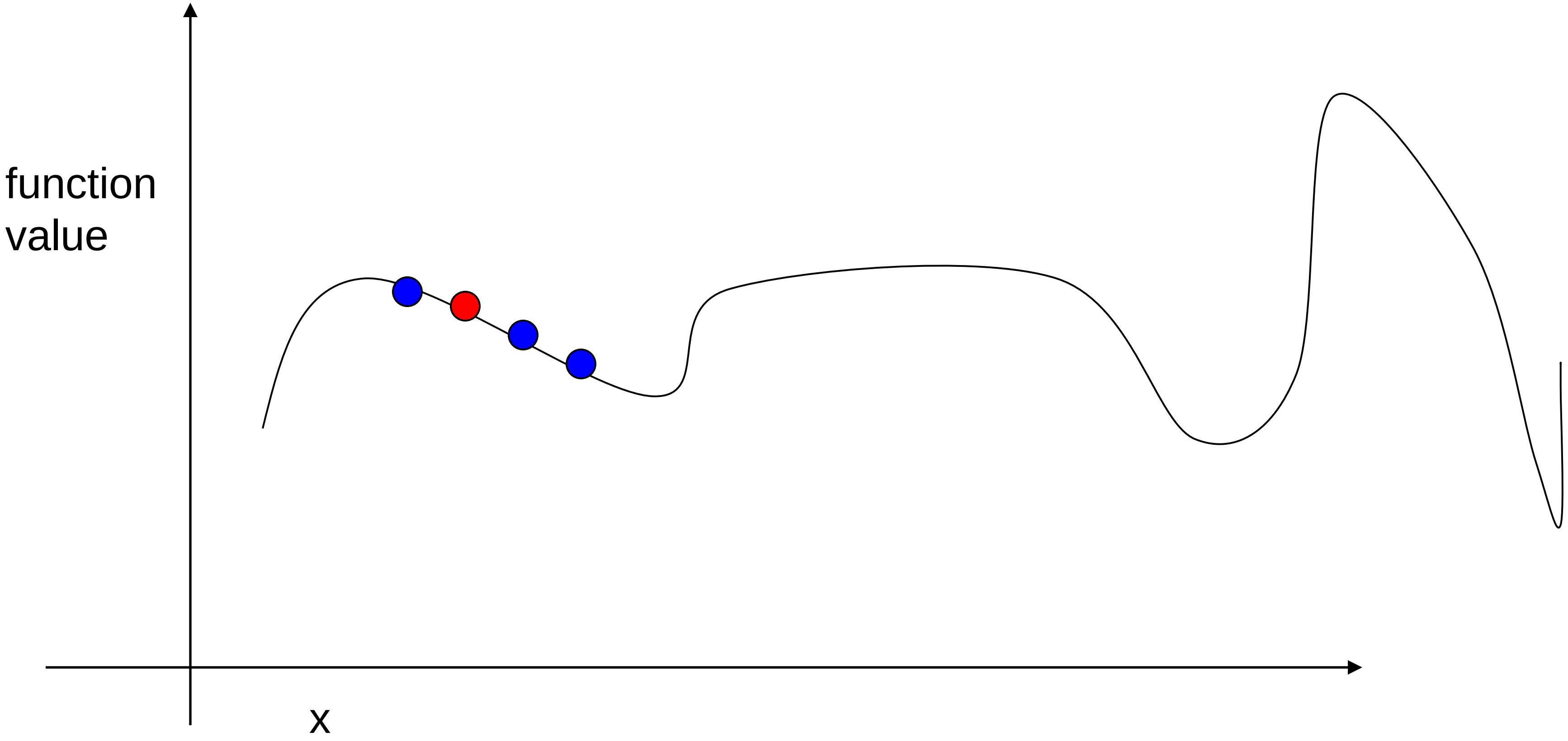
Simple Example

- Choose Best One for new position



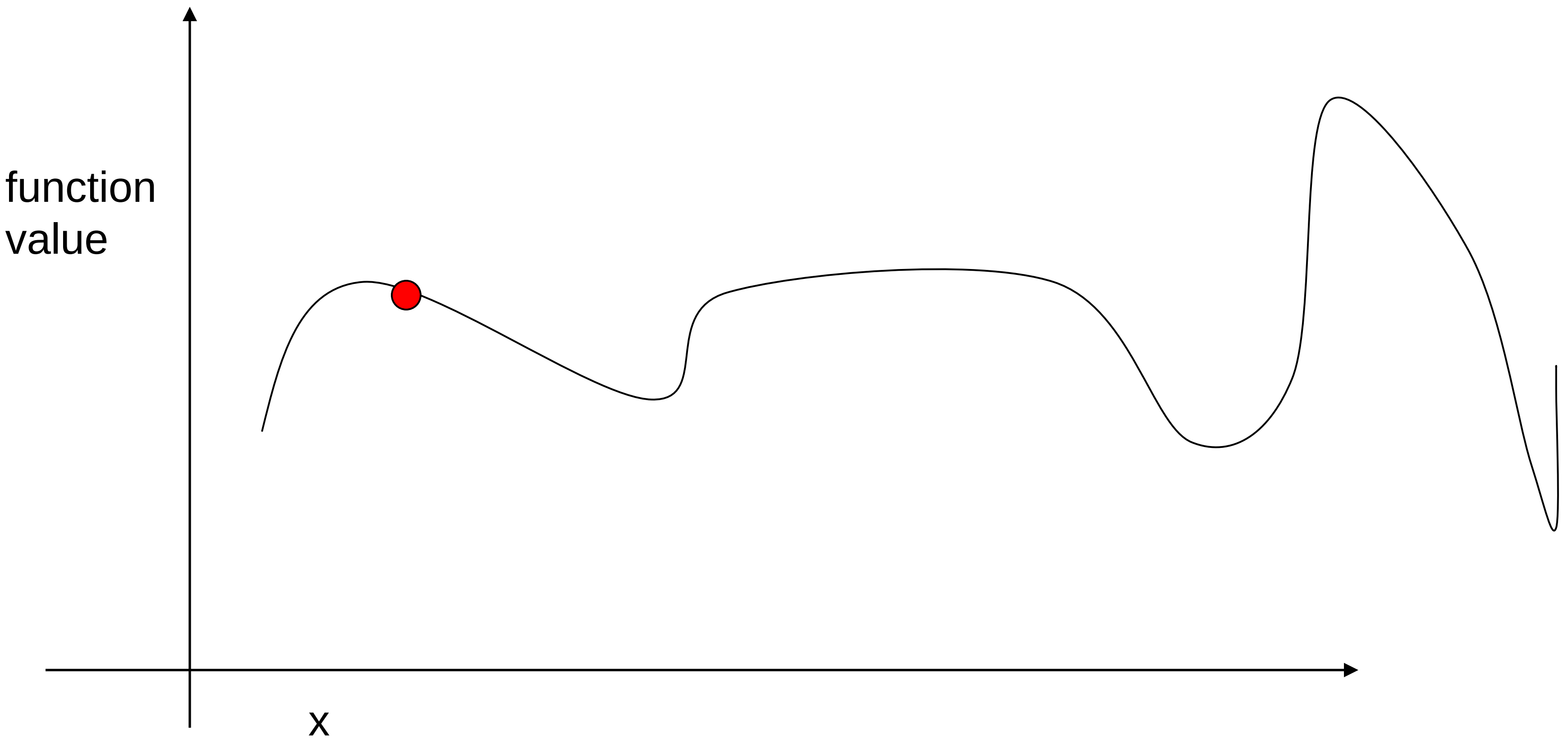
Simple Example

- Repeat



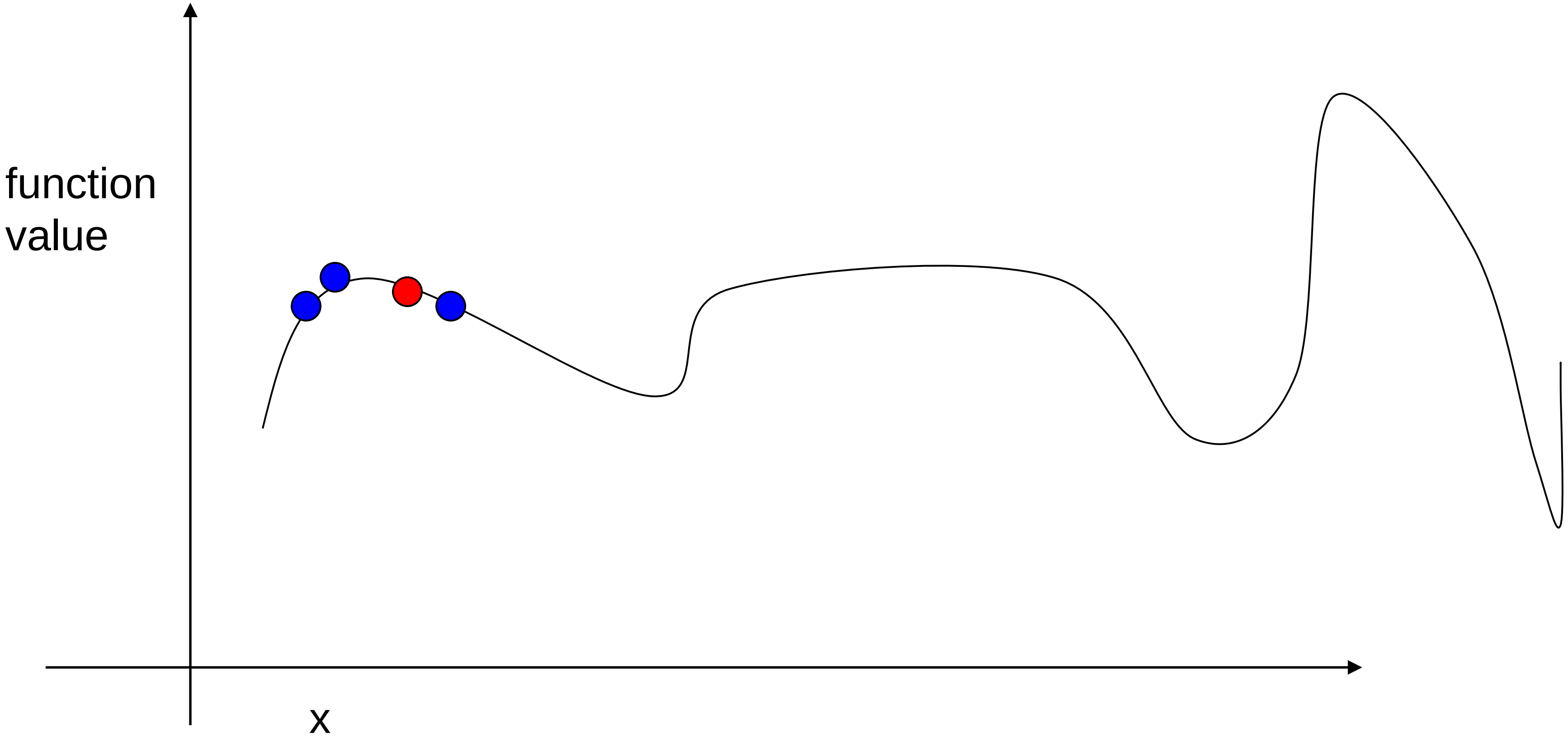
Simple Example

- Repeat



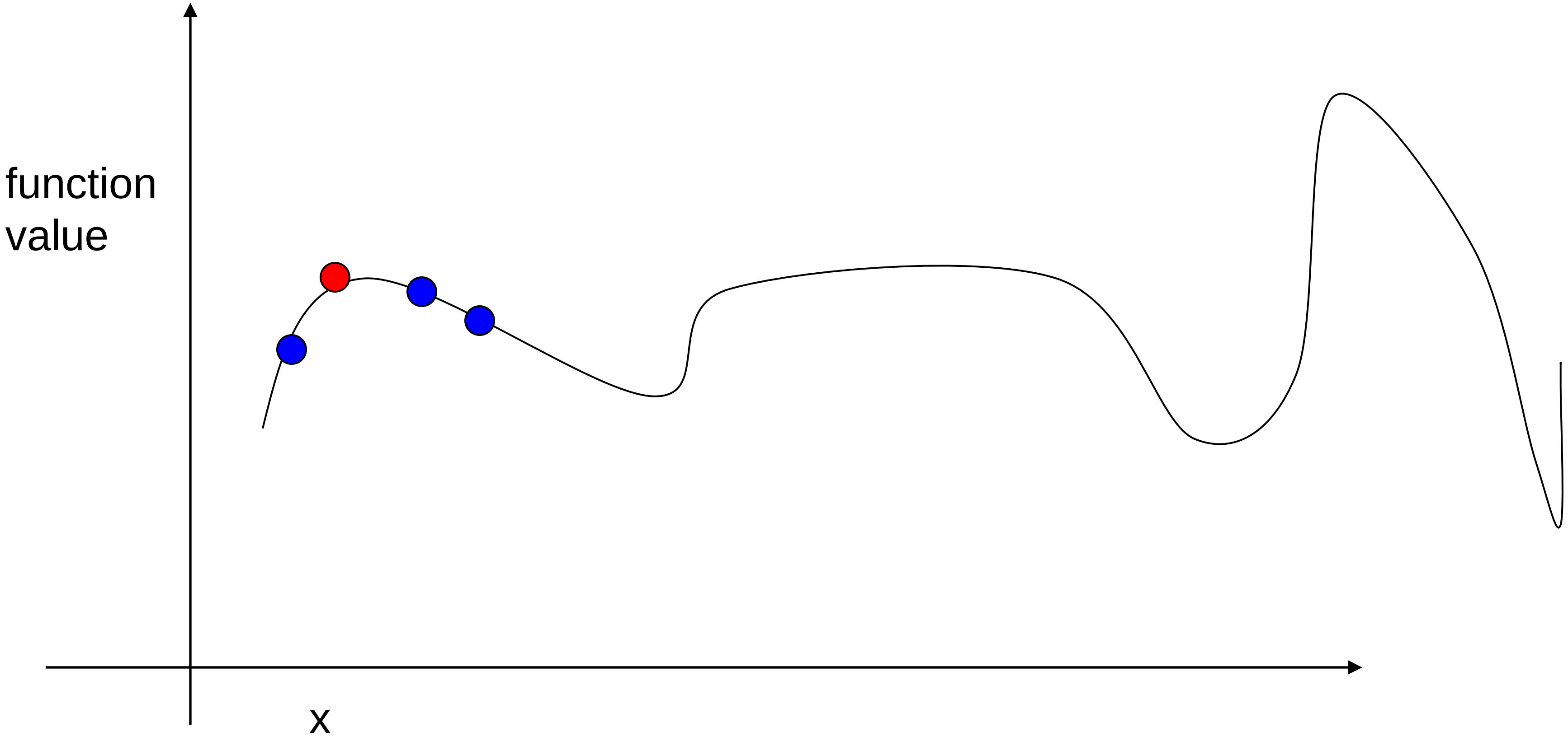
Simple Example

- Repeat



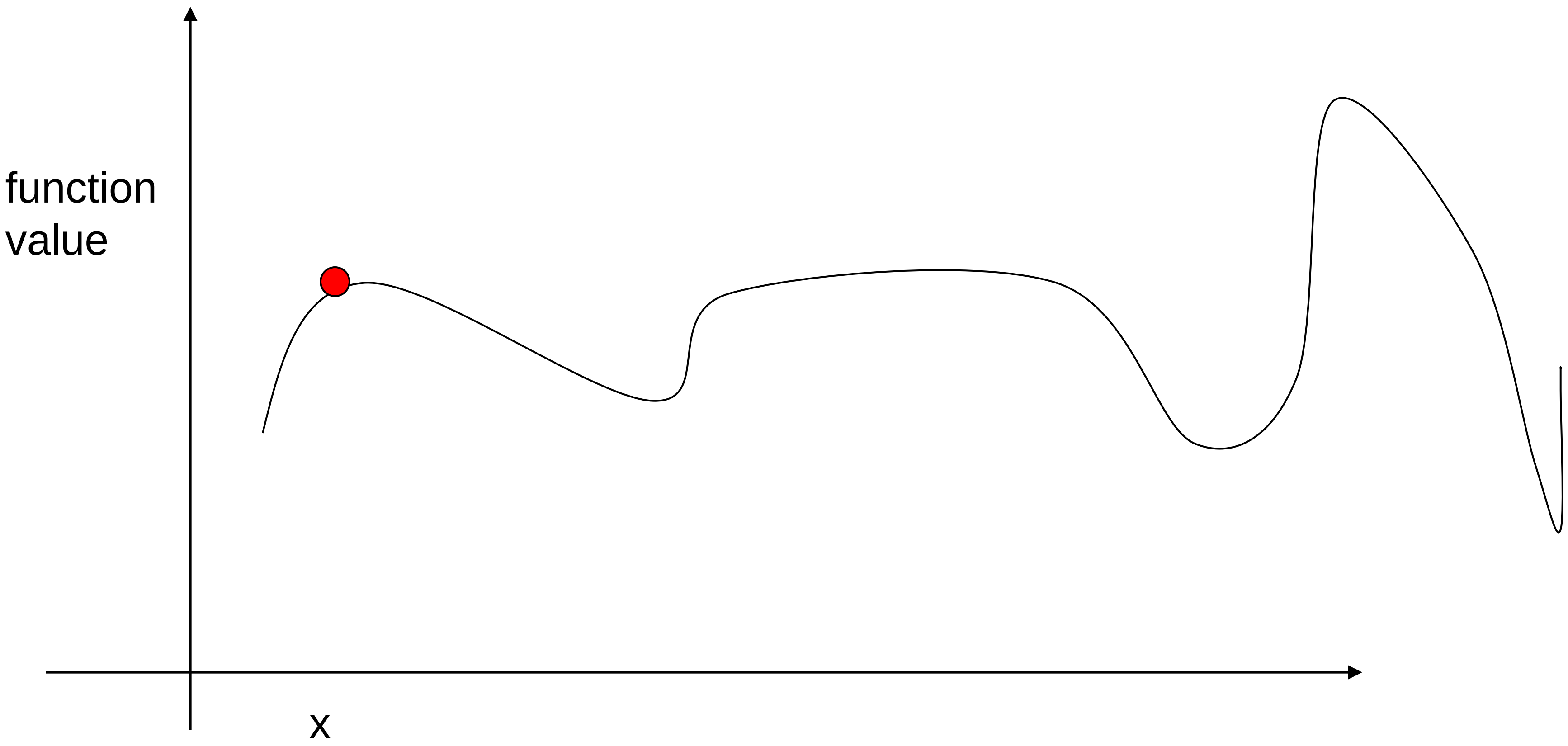
Simple Example

- Repeat



Simple Example

- No Improvement, so stop.



Some Theoretical Guidance

No Free Lunch (NFL) Theorem(s)

You can only get generalization through assumptions. No one algorithm will solve all problems (some will work better than others in some instances).

NFL Theorem (Wolpert & Macready, 1997)

- If any algo A outperforms another algo B in search for an extremum of objective function, then algo B will outperform algo A on other objective functions
  - Where any θ (discrete/contus/mixed) maps cost function into a finite set
  - Applies to both deterministic & stochastic problems
- Suggests that **average** performance over **all** possible cost functions is same for **all** search algorithms
  - Universally best method does not exist for all optimization problems
  - BUT – this does not mean all algo's perform equally well over some specific functions or specific set of problems
- ***[White board notes]***

NFL Theorem Implications/Issues

- Not proven yet for multi-objective functions (just single)
- For specific problem w/ specific cost functions, there usually exist some algo’s more efficient than others
  - **IF** we do not need to measure their average performance (otherwise, no better than random search on average)
- Auger & Teytaud (2010) – contus problems can be free (perhaps)
- Mashall & TG Hinton (2010) – assumption of time-ordered set of *m* distinct points/visits not valid for real-life algo’s (which violates basic assumption of non-revisitation, etc.)

Algorithm Decisions/Choices

- For given type of problem, what is the best algo to use? (very hard!)
- ***Issue***
  - Might not know efficiency of algo before trying it
  - Some algorithms do not yet “exist” (need development/modification)
  - Meta/hyper-parameters – depends on decision-maker, resources, problem type
- For given algo, what kinds of problems can it solve?
- ***Issue***
  - Explore algo on different problems, compare & rank w.r.t. efficiency
  - Find advantages/disadvantages to guide algo choice
  - Domain knowledge **always** helps in choosing best/most efficient methods
  - Ex: Airplanes – if start from shape of bird/fish, design will likely be more useful

Gradient Descent (or Ascent)

- Simple modification to hill climbing
  - Generally assumes a continuous state space
- Idea is to take more intelligent steps
- Look at local gradient: the direction of largest change
- Take step in that direction
  - Step size should be proportional to gradient
- Tends to yield much faster convergence to maximum

Discretization methods turn continuous space into discrete space, e.g., empirical gradient considers $\pm\delta$ change in each coordinate

Gradient methods compute

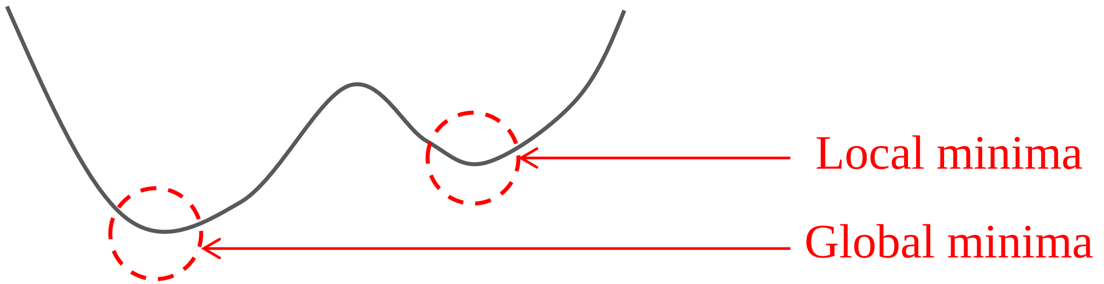
$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

to increase/reduce f, e.g., by $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

The Value of Derivatives

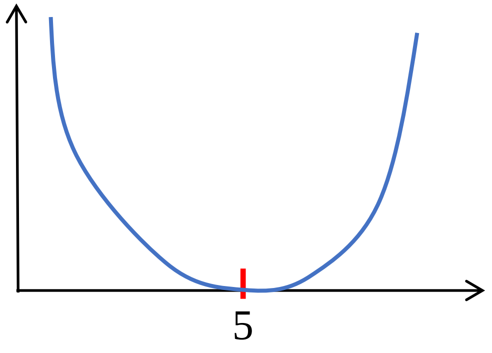
- How do you solve the following problems?

(1)



(2) $\boxed{\min_{w} f(w) = (w-5)^2}$

(a) Plot



(b) Take ***derivatives***, **check = 0**

You want to build a simple univariate regression model for predicting profits y for a food truck. Furthermore, you decide to restrict yourself to a linear hypothesis space and construct a model that adheres to the following form:

$$f_\Theta(x) = \theta_0 + \theta_1 x$$

Given the data you have collected, represented as a set of m complete (y, x) pairs, your goals will be to estimate the parameters of this model $\Theta = \{\theta_0, \theta_1\}$ (where $\theta_{j>0}$ is the vector of learnable coefficients that weight the observed variables, and θ_0 is a single bias coefficient) using the method of steepest gradient descent. The cost function to minimize is the well-known mean squared error (MSE) defined as follows:

$$\mathcal{J}(\Theta) = \frac{1}{2m} \sum_{i=1}^{m} (f_\Theta(x^i) - y^i)^2$$

What is a useful constrained version of this cost to get "smaller" coefficients?

You want to build a simple univariate regression model for predicting profits y for a food truck. Furthermore, you decide to restrict yourself to a linear hypothesis space and construct a model that adheres to the following form:

$$f_\Theta(x) = \theta_0 + \theta_1 x$$

Given the data you have collected, represented as a set of m complete (y, x) pairs, your goals will be to estimate the parameters of this model $\Theta = \{\theta_0, \theta_1\}$ (where $\theta_{j>0}$ is the vector of learnable coefficients that weight the observed variables, and θ_0 is a single bias coefficient) using the method of steepest gradient descent. The cost function to minimize is the well-known mean squared error (MSE) defined as follows:

$$\mathcal{J}(\Theta) = \frac{1}{2m} \sum_{i=1}^{m} (f_\Theta(x^i) - y^i)^2$$

What is a useful constrained version of this cost to get "smaller" coefficients?
(*Hint*: Tikhonov regularization)

Linear Regression & Gradient Descent

The gradient of the negative log likelihood with respect to the model parameters $\Theta = \{\theta_0, \theta_1\}$, after application of the chain rule, takes the general form:

$$\frac{\partial \mathcal{J}(\Theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(f_\Theta(x^i) - y^i)x_j^i, j = 0, 1, 2, \cdots, n$$

where j indexes a particular parameter, noting that $x_0 = 1$.[1] In the univariate (single-variable) case, this leads us to utilize the following two specific gradients:

$$\frac{\partial \mathcal{J}(\Theta)}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}(f_\Theta(x^i) - y^i)x_1^i$$

$$\frac{\partial \mathcal{J}(\Theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(f_\Theta(x^i) - y^i)(x_o^i = 1) = \frac{1}{m}\sum_{i=1}^{m}(f_\Theta(x^i) - y^i)$$

where we see that the partial derivative of the loss with respect to θ_0 (the bias b) takes a simpler form given that the feature it weights is simply $x_0 = 1$ (we are essentially augmenting the pattern x with a bias of one, which allows us to model the mean μ of the data's distribution, assuming that it is Gaussian distributed).

To learn model parameters using batch gradient descent, we will need to implement the following update rule (for each θ_j in Θ):

$$\theta_j = \theta_j - \alpha\frac{\partial \mathcal{J}(\Theta)}{\partial \theta_j} = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(f_\Theta(x^i) - y^i)x_j^i, j = 0, 1, 2, \cdots, n.$$

The regularized loss function (making this ridge regression) takes the following form:

$$\mathcal{J}(\Theta) = \frac{1}{2m} \sum_{i=1}^{m} (f_\Theta(x^i) - y^i)^2 + \frac{\beta}{2m} \sum_{j=1}^{n} \theta_j^2$$

A "soft" constraint!

where β is another meta-parameter for us to set (through trial and error, or, in practice, through proper cross-fold validation). The gradient of this regularized form of the loss with respect to parameters Θ is straightforward:

$$\frac{\partial \mathcal{J}(\Theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (f_\Theta(x^i) - y^i) x_j^i + \frac{\beta}{m} \theta_j$$

except for the case of $j = 0$ (which indexes the bias parameter), on which the penalty is not applied.

Questions?

Questions?