Applied Optimization

Alexander G. Ororbia II

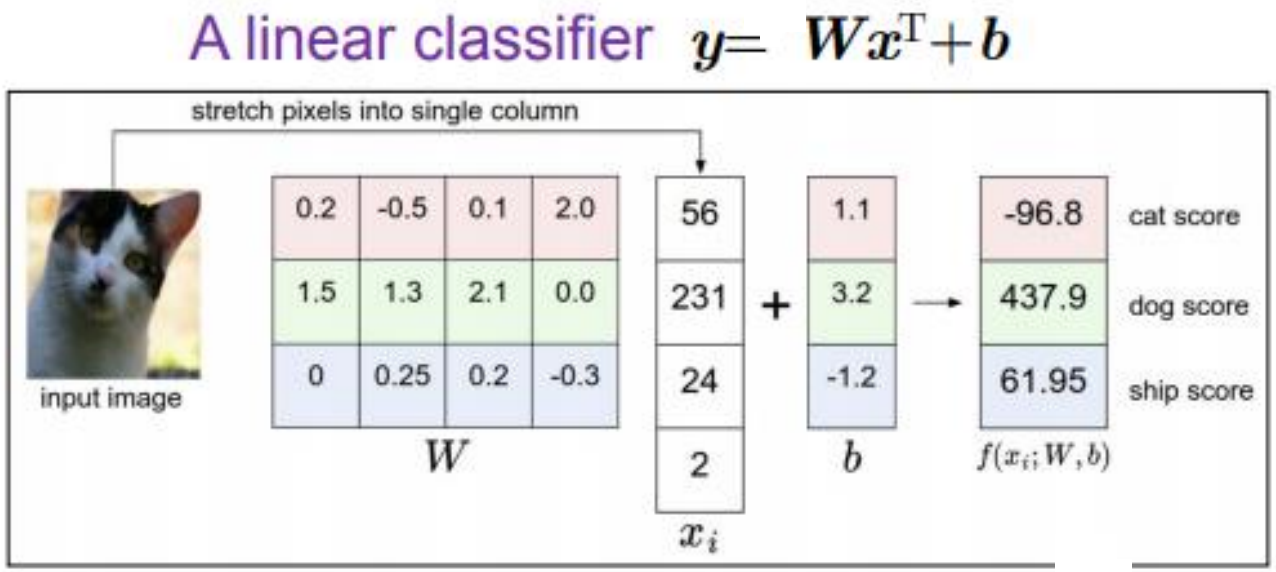
Introduction to Machine Learning

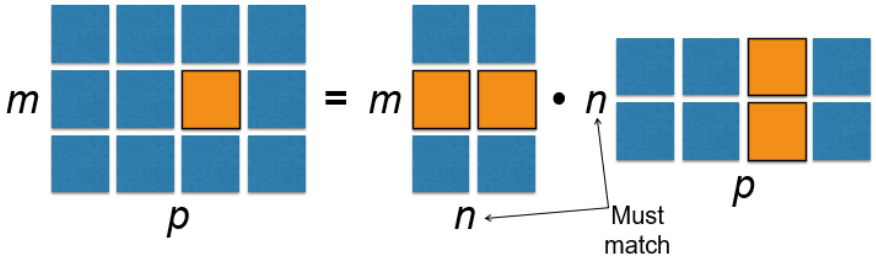
CSCI-335

9/2/2026

Tensors in Statistical Learning

Vector x is converted into vector y by multiplying x by a matrix W





Optimization and Decision-Making Problems

1. Get a precise definition of problem, all relevant data and information on it
    - Uncontrollable factors (“random” variables)
    - Controllable inputs (“decision” variables)

2. Construct a mathematical (optimization) model of problem
    - Build objective functions and constraints

3. Solve model
    - Apply most appropriate algorithms for given problem

4. Implement solution

Mathematical Optimization in the "Real World"

Mathematical Optimization is a branch of applied mathematics which is useful in many different fields. Here are a few examples:

- Manufacturing
- Production
- Inventory control
- Transportation
- Scheduling
- Networks
- Finance
- Engineering
- Mechanics
- Economics
- Control engineering
- Marketing
- Policy Modeling

Optimization Vocabulary

Your basic optimization problem consists of…

- The objective function, f(x), which is the output you're trying to maximize or minimize.

Optimization Vocabulary

Your basic optimization problem consists of…

- The objective function, f(x), which is the output you're trying to maximize or minimize.
- Variables, x_1 x_2 x_3 and so on, which are the inputs – things you can control. They are abbreviated x_n to refer to individuals or x to refer to them as a group.

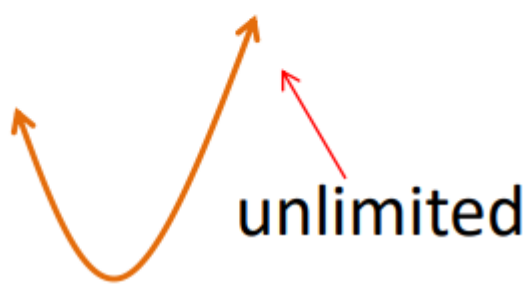
Optimization Vocabulary

Your basic optimization problem consists of…

- The objective function, $f(x)$, which is the output you're trying to maximize or minimize.
- Variables, x_1 x_2 x_3 and so on, which are the inputs – things you can control. They are abbreviated x_n to refer to individuals or x to refer to them as a group.
- Constraints, which are equations that place limits on how big or small some variables can get. Equality constraints are usually noted $h_n(x)$ and inequality constraints are noted $g_n(x)$.

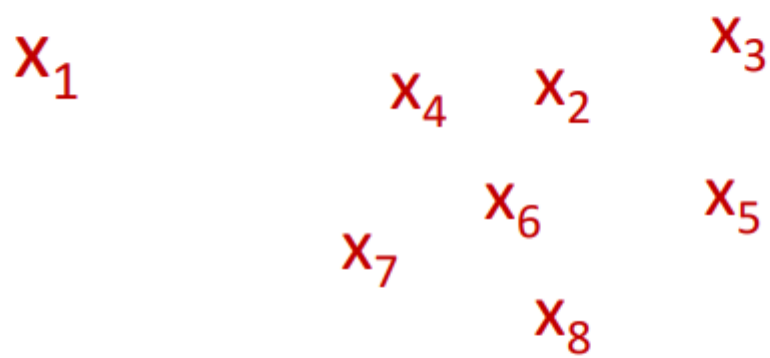
Types of Optimization Problems

- Some problems have constraints and some do not.



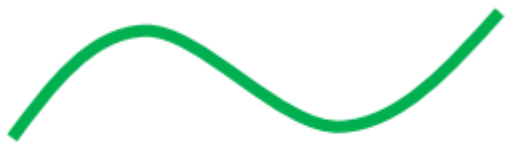
Types of Optimization Problems

- Some problems have constraints and some do not.
- There can be one variable or many.



Types of Optimization Problems

- Some problems have constraints and some do not.
- There can be one variable or many.
- Variables can be discrete (for example, only have integer values) or continuous.



Types of Optimization Problems

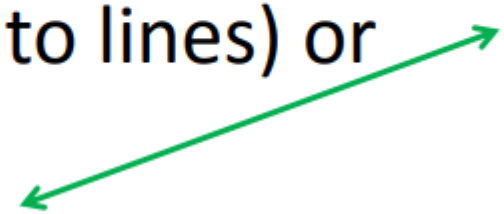
- Some problems have constraints and some do not.
- There can be one variable or many.
- Variables can be discrete (for example, only have integer values) or continuous.
- Some problems are static (do not change over time) while some are dynamic (continual adjustments must be made as changes occur).

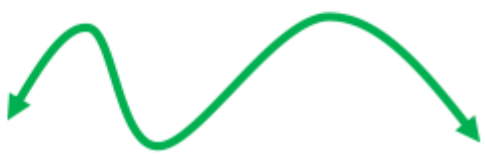
Types of Optimization Problems

- Some problems have constraints and some do not.
- There can be one variable or many.
- Variables can be discrete (for example, only have integer values) or continuous.
- Some problems are static (do not change over time) while some are dynamic (continual adjustments must be made as changes occur).
- Systems can be deterministic (specific causes produce specific effects) or stochastic (involve randomness/ probability).

Types of Optimization Problems

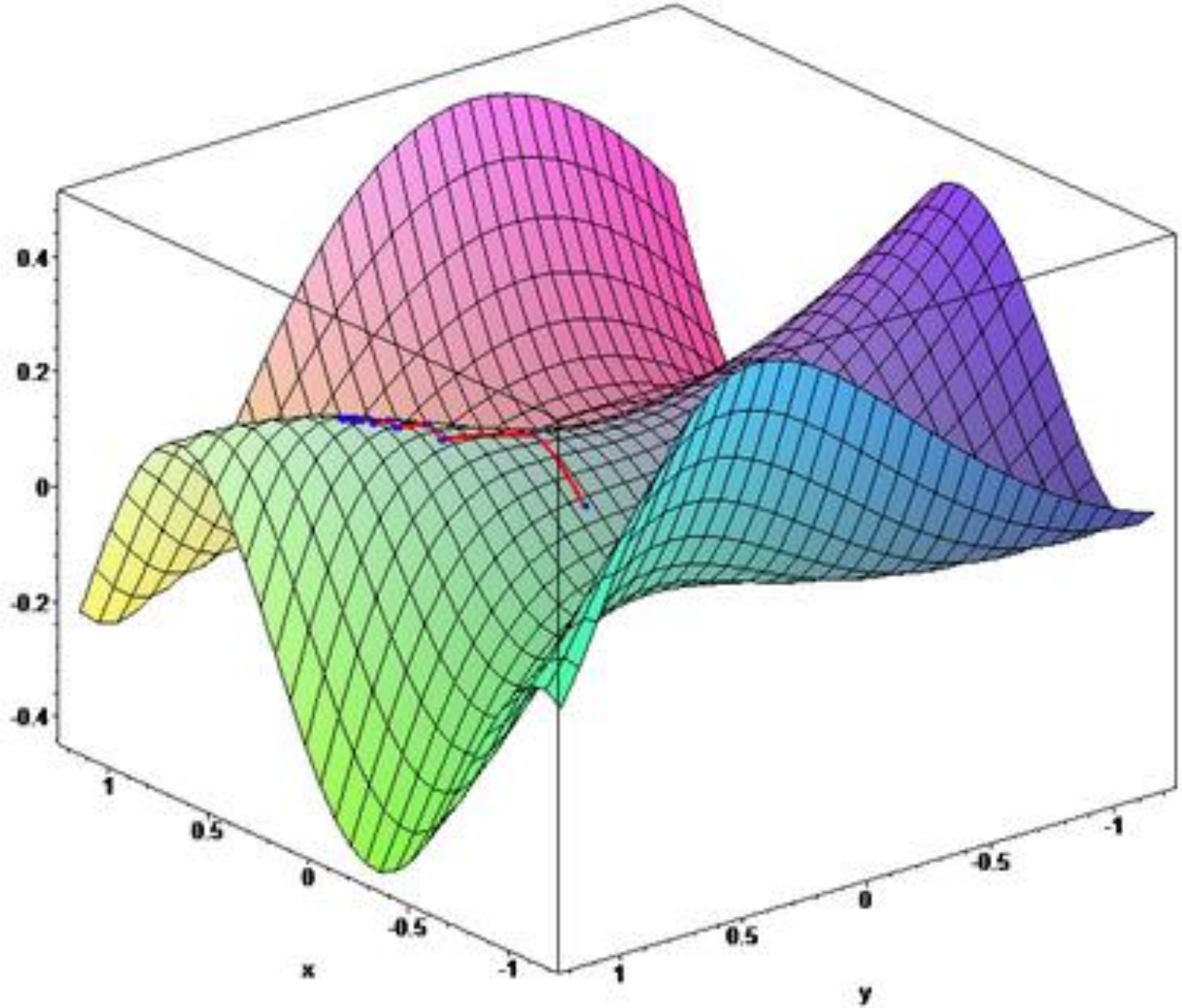
- Some problems have constraints and some do not.
- There can be one variable or many.
- Variables can be discrete (for example, only have integer values) or continuous.
- Some problems are static (do not change over time) while some are dynamic (continual adjustments must be made as changes occur).
- Systems can be deterministic (specific causes produce specific effects) or stochastic (involve randomness/ probability).
- Equations can be linear (graph to lines) or nonlinear (graph to curves)





Gradient-Based Optimization

- Most ML algorithms involve optimization
- Minimize/maximize a function $f(\boldsymbol{x})$ by altering $\boldsymbol{x}$
  - Usually stated a minimization
  - Maximization accomplished by minimizing $-f(\boldsymbol{x})$
- $f(\boldsymbol{x})$ referred to as objective function or criterion
  - In minimization also referred to as loss function cost, or error
  - Example is linear least squares $\boxed{f(x) = \frac{1}{2} || Ax - b ||^2}$
  - Denote optimum value by $\boldsymbol{x}^* = \text{argmin}\, f(\boldsymbol{x})$


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

Problem Specification

Suppose we have a cost function (or ***objective function***)

$$f(\mathbf{x}) : \mathbb{R}^N \longrightarrow \mathbb{R}$$

Our aim is to find values of the parameters (decision variables / ***features***) $\mathbf{x}$ that minimize this function

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} f(\mathbf{x})$$

(Often) subject to the following ***constraints***:

- **equality**: $c_i(\mathbf{x}) = 0$
- **nonequality**: $c_j(\mathbf{x}) \geq 0$

We will handle these a bit more "softly" (to be discussed later)!

If we seek a maximum of $f(\mathbf{x})$, it is equivalent to seeking a minimum of $-f(\mathbf{x})$

Equivalence between Minimum and Maximum

- If a point x^* corresponds to the minimum value of the function $f(x)$, the same point also corresponds to the maximum value of the negative of the function, $-f(x)$.
  - This means optimization can be re-interpreted to mean minimization since the maximum of a function can be found by seeking the minimum of the negative of the same function.

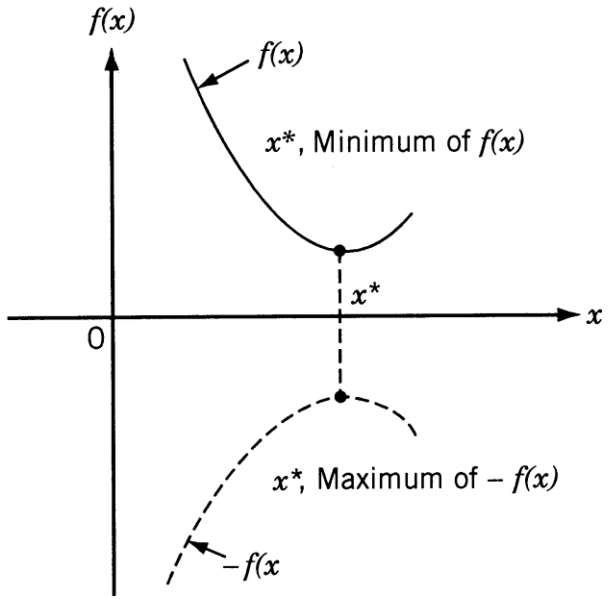
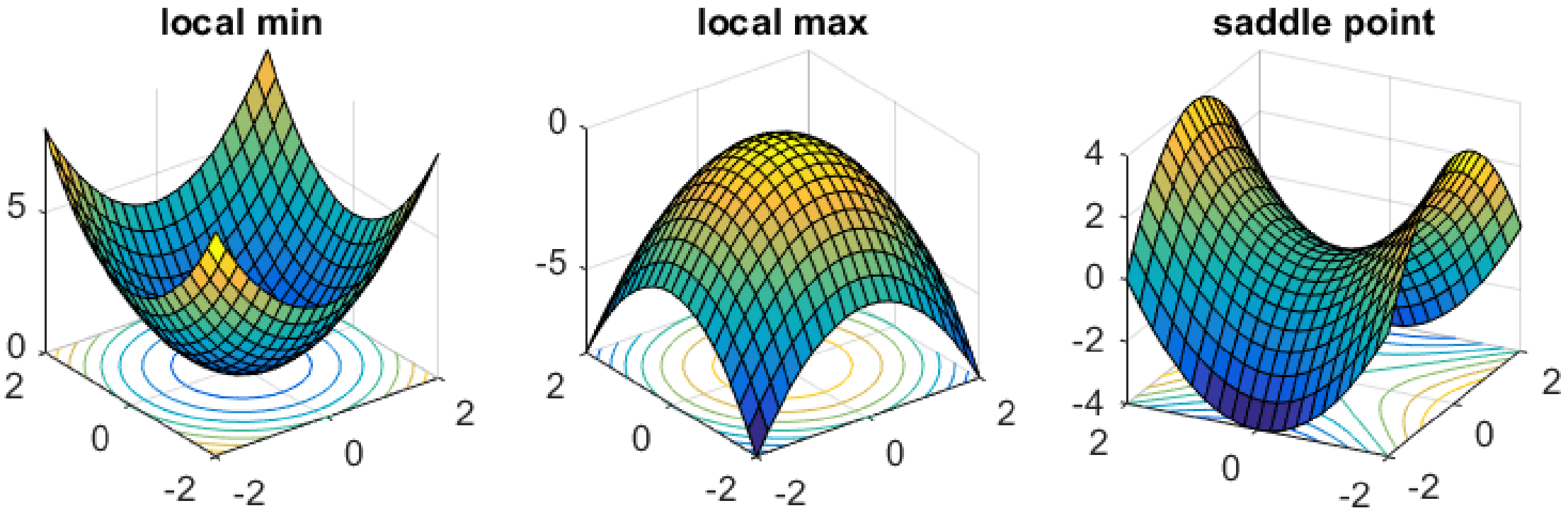


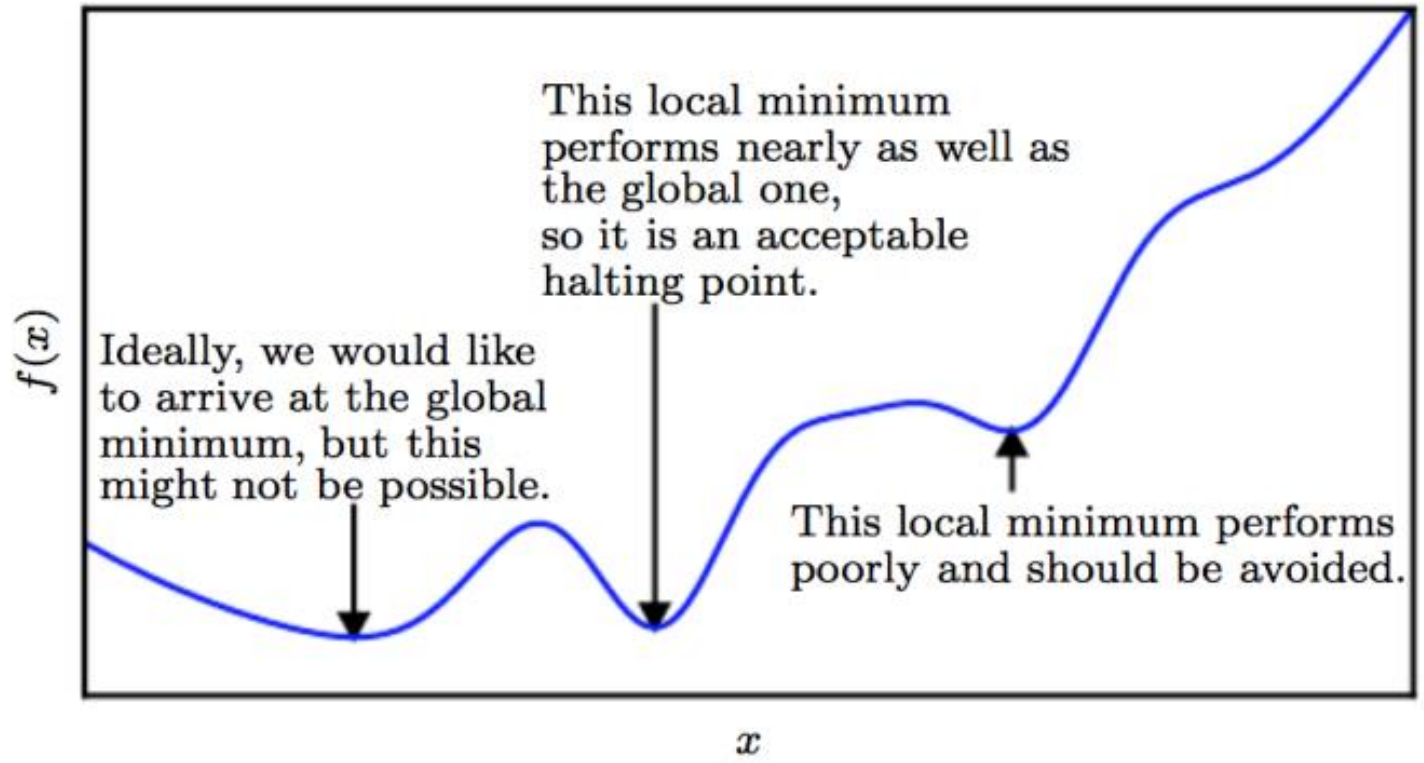
Figure 1.1 Minimum of $f(x)$ is same as maximum of $-f(x)$.

The Many Faces of Optimization!!



Presence of Multiple Optima (Minima)

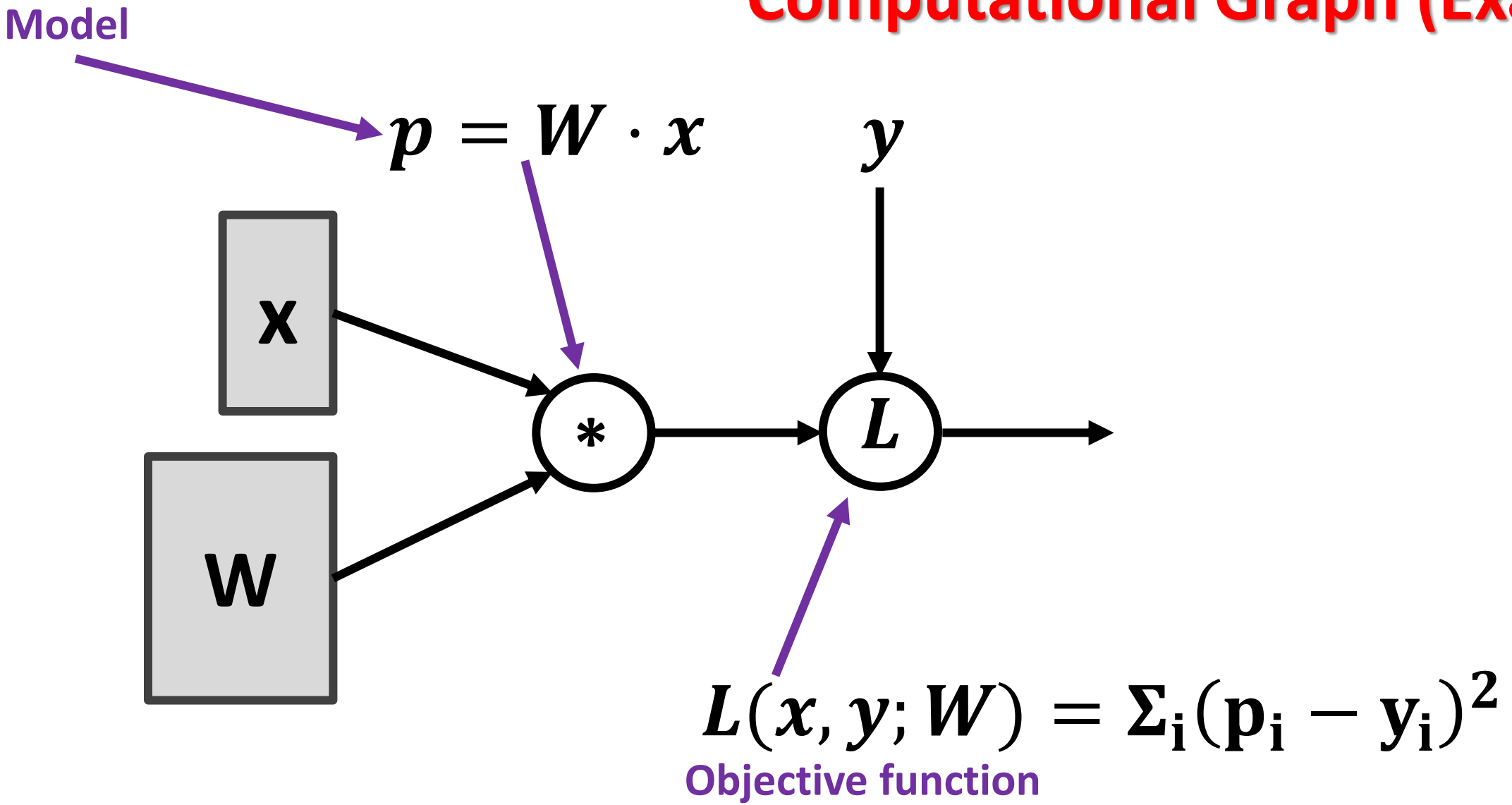
- Optimization algorithms may fail to find global minimum
- Generally accept such solutions



Minimizing with Multiple Inputs

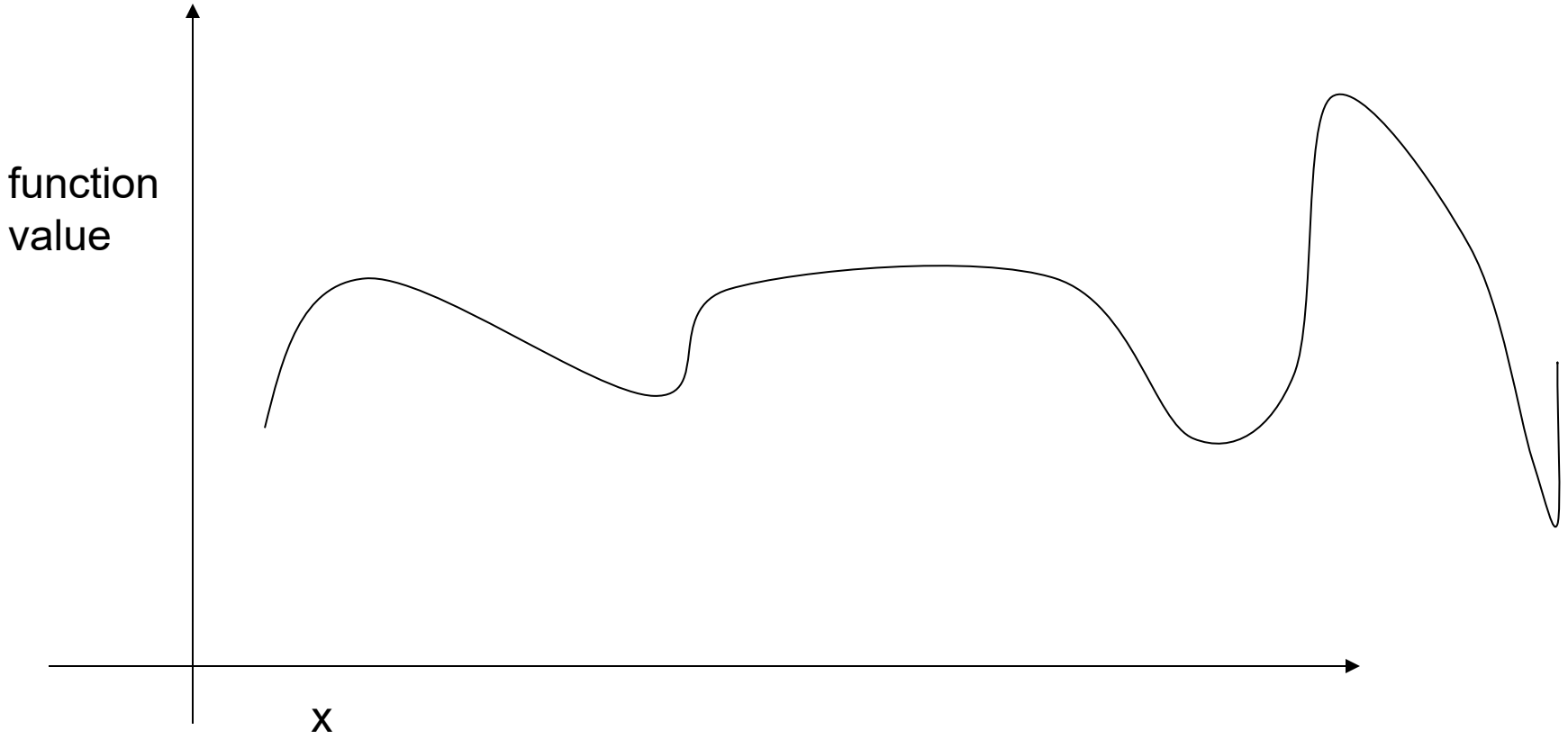
- We often minimize functions with multiple inputs: $f\colon R^n \rightarrow R$
- For minimization to make sense there must still be only one (scalar) output

Computational Graph (Example)



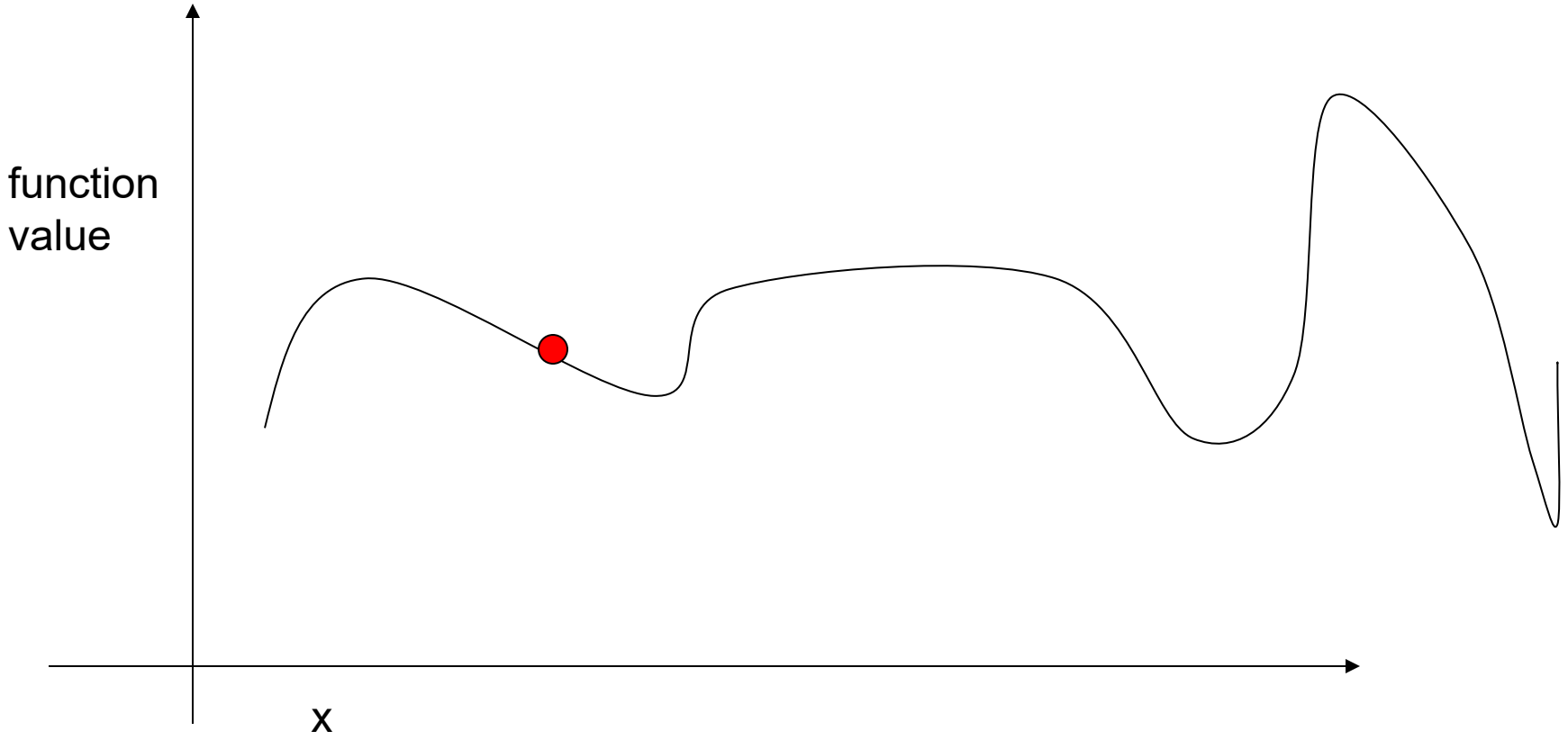
Simple Example: The Idealized Climb

- One dimension (typically use more):



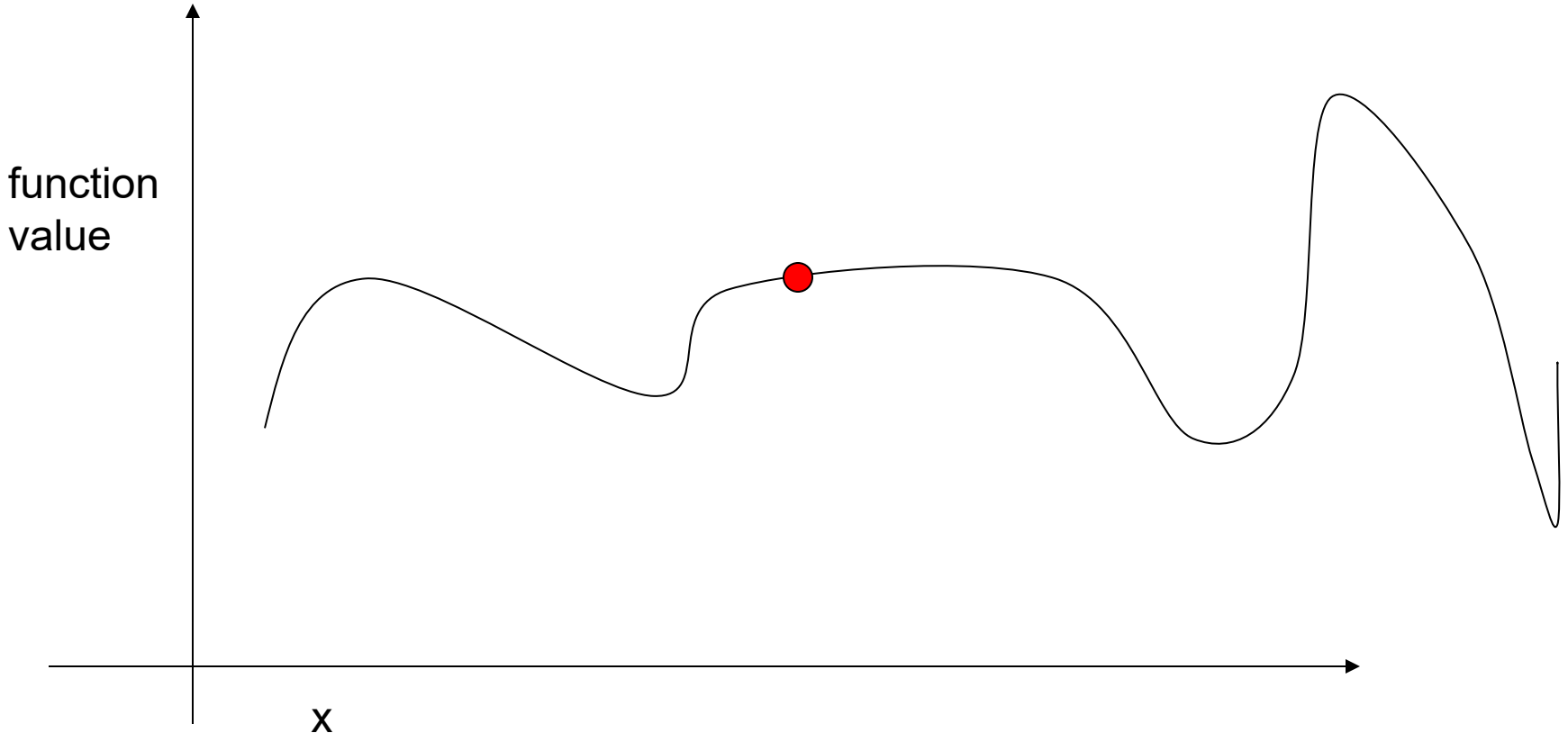
Simple Example: The Idealized Climb

- Start at a valid state, try to maximize



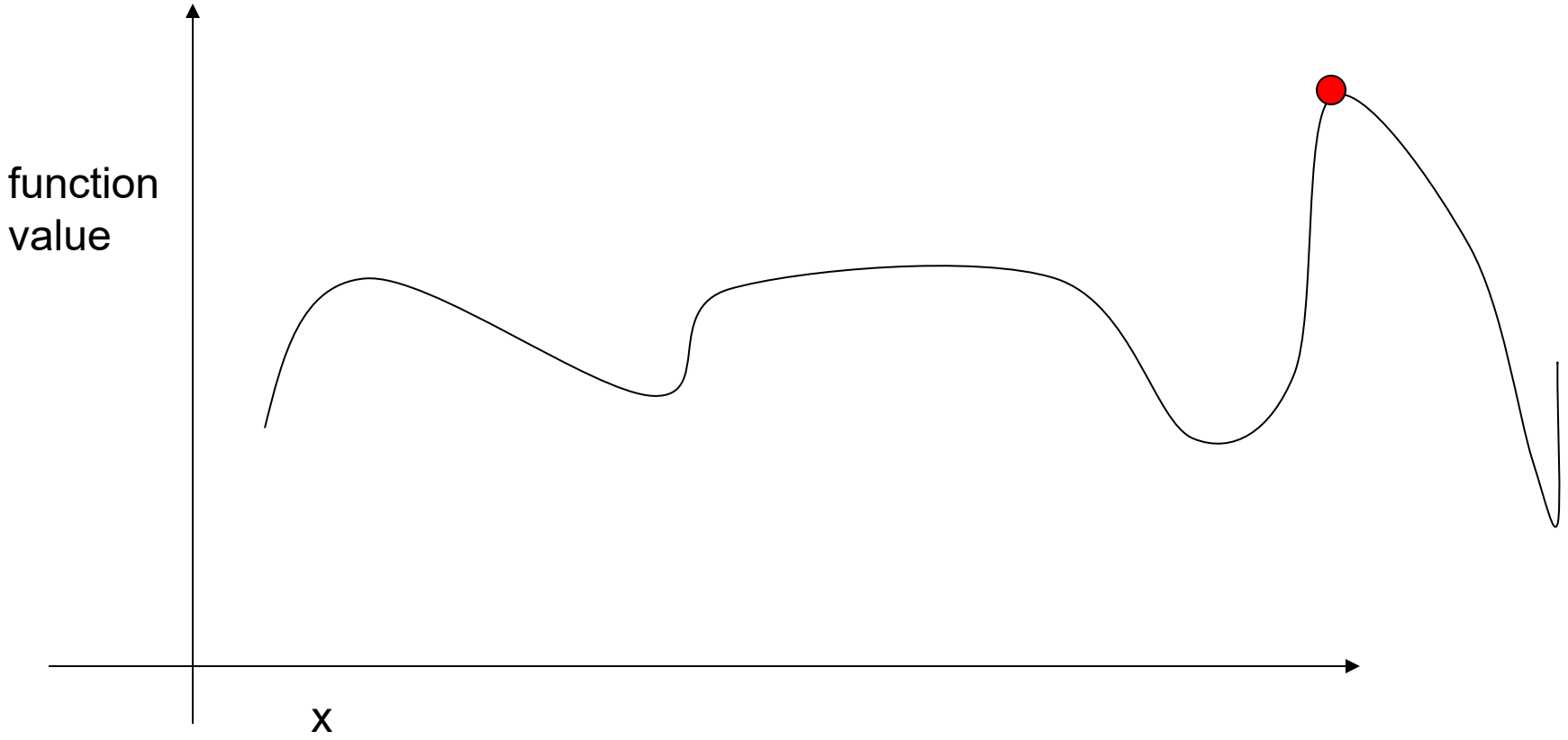
Simple Example: The Idealized Climb

- 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 (*variations*: first-choice hill climbing, **random-restart hill-climbing**)

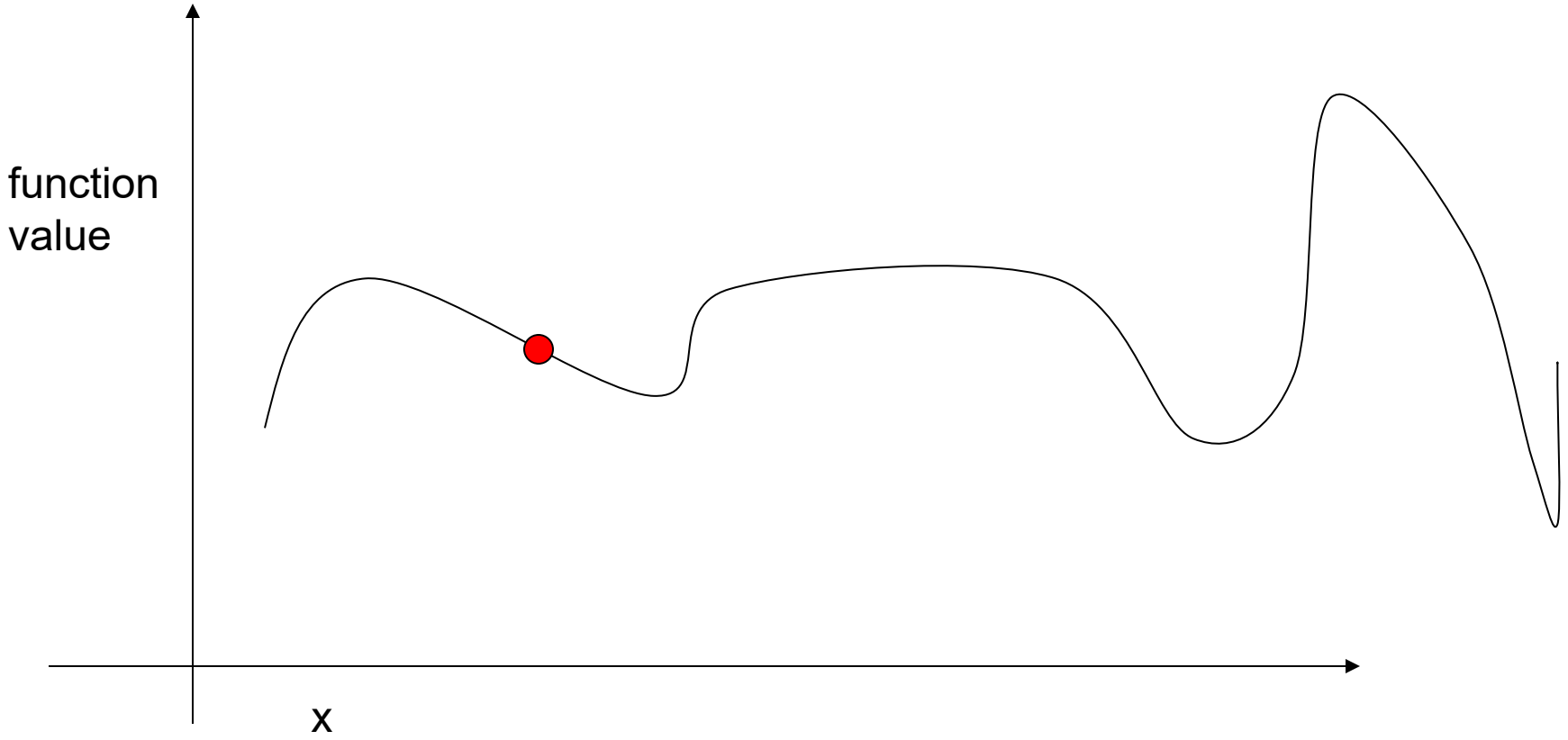
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    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

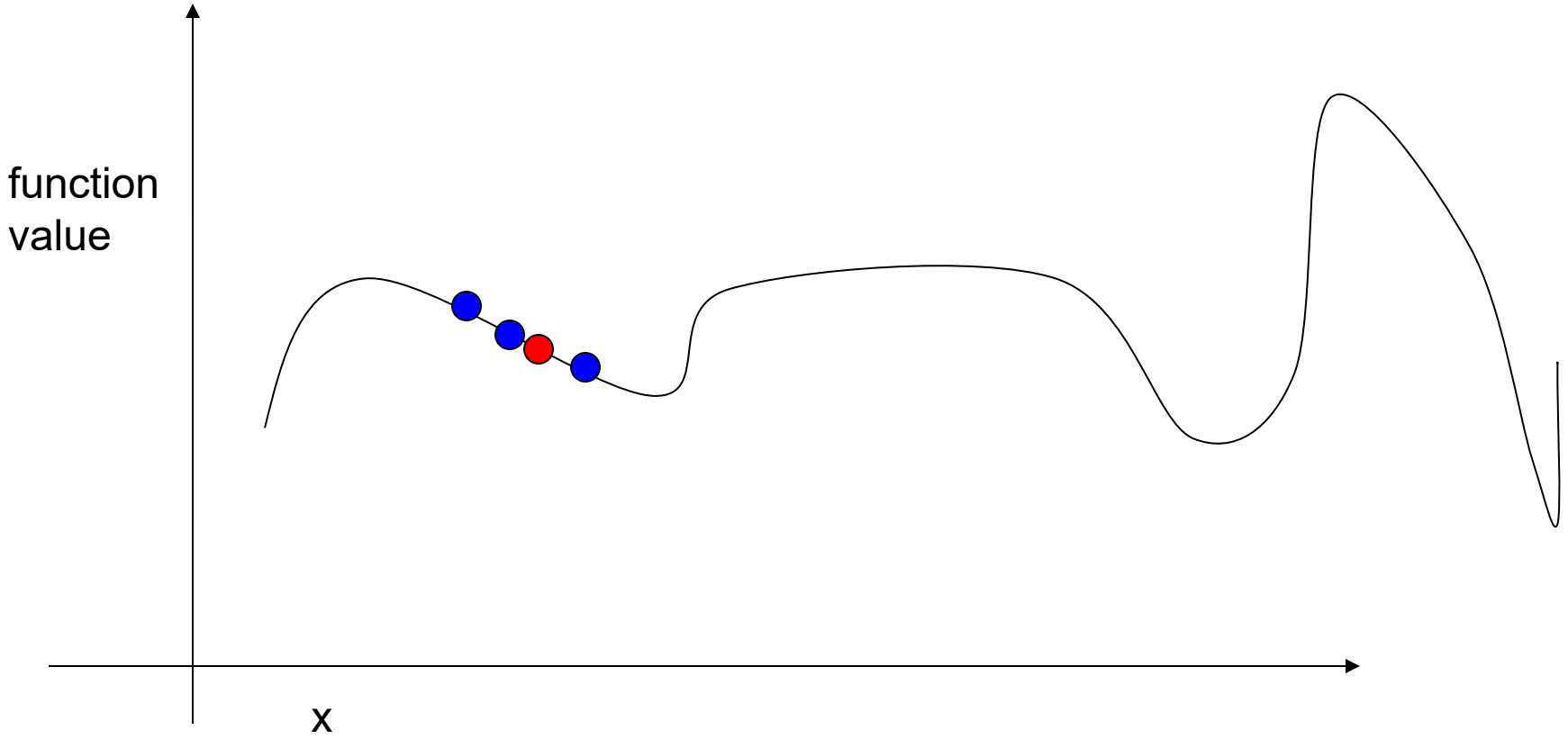
Stochastic Hill Climbing (Ascent)

- Random Starting Point



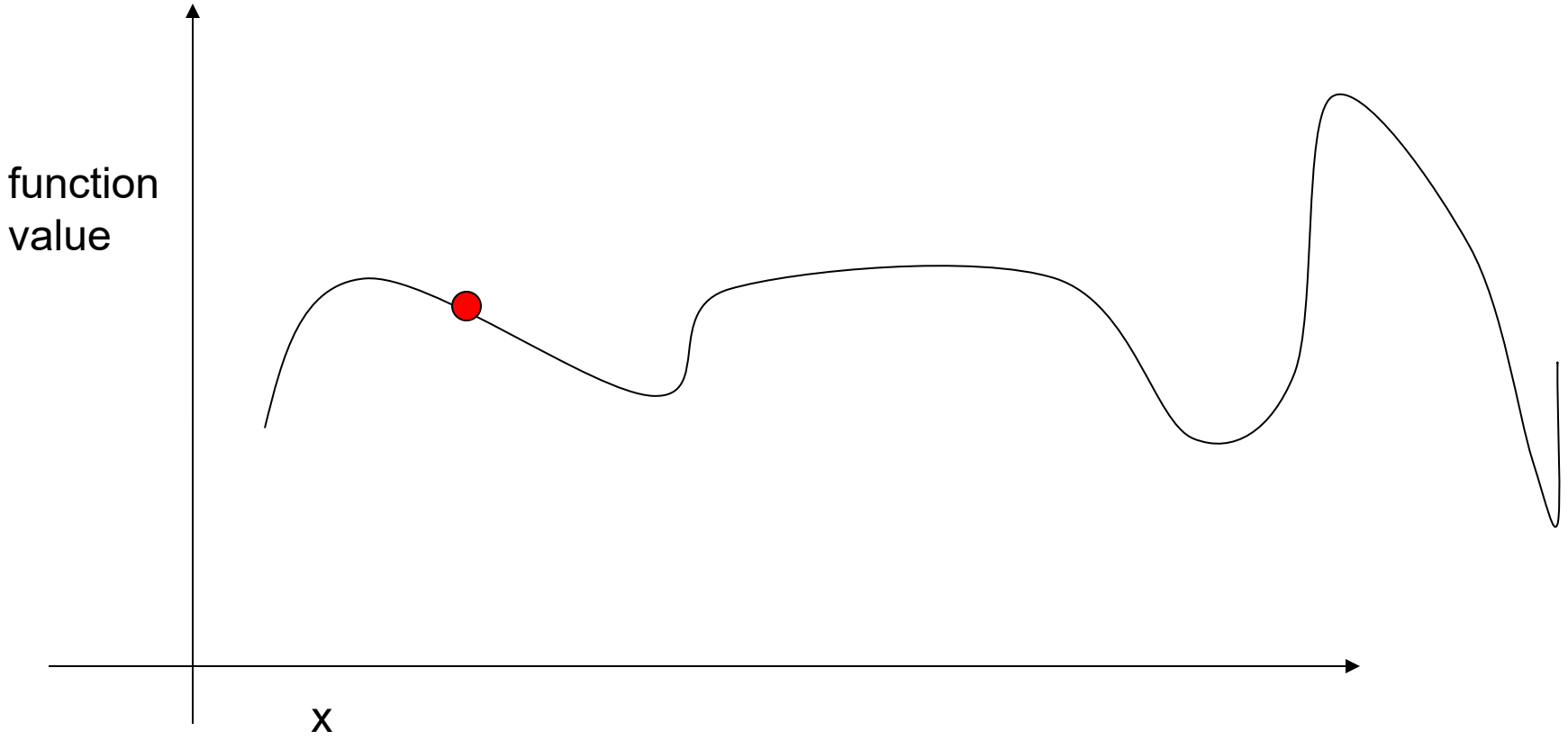
Stochastic Hill Climbing (Ascent)

- Three random steps



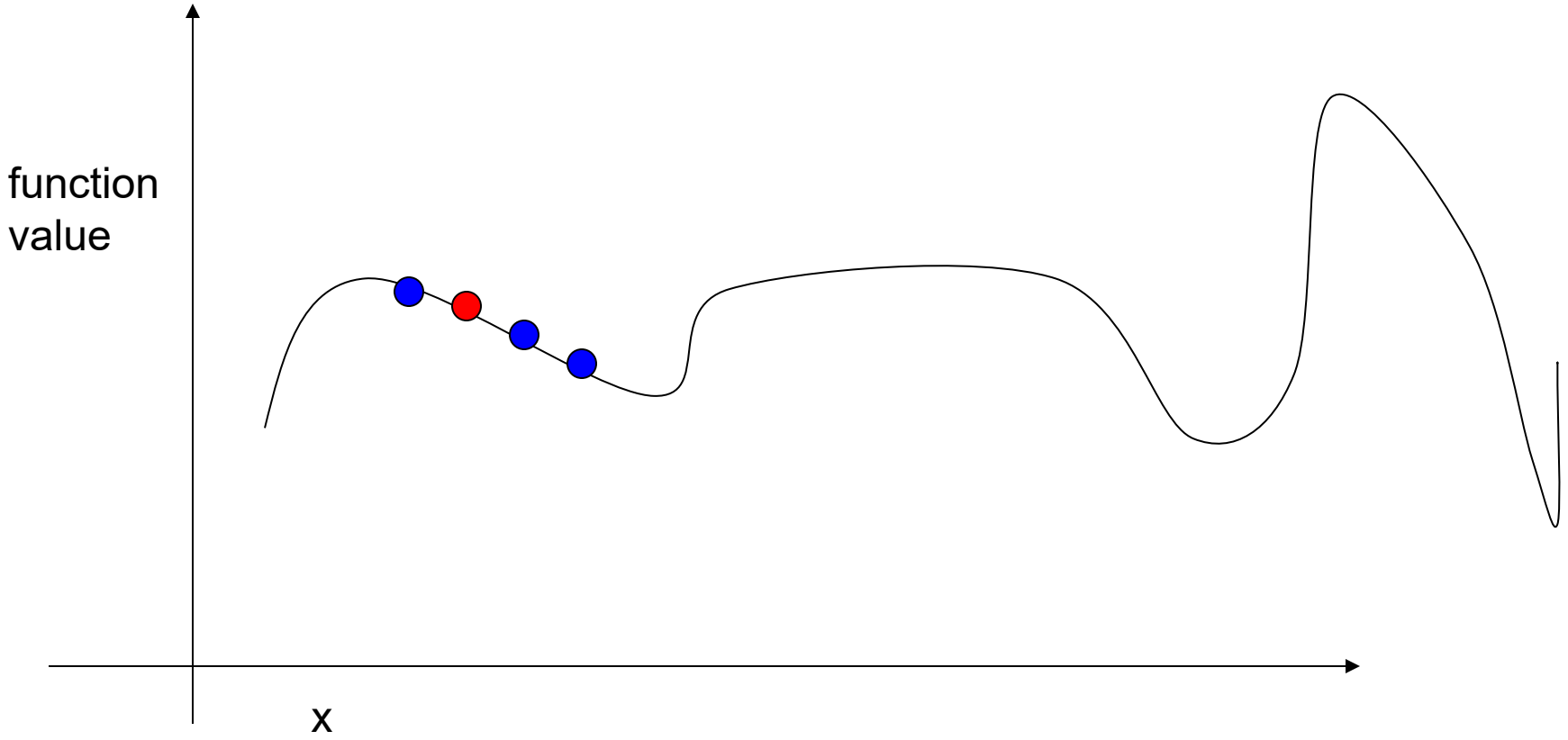
Stochastic Hill Climbing (Ascent)

- Choose Best One for new position



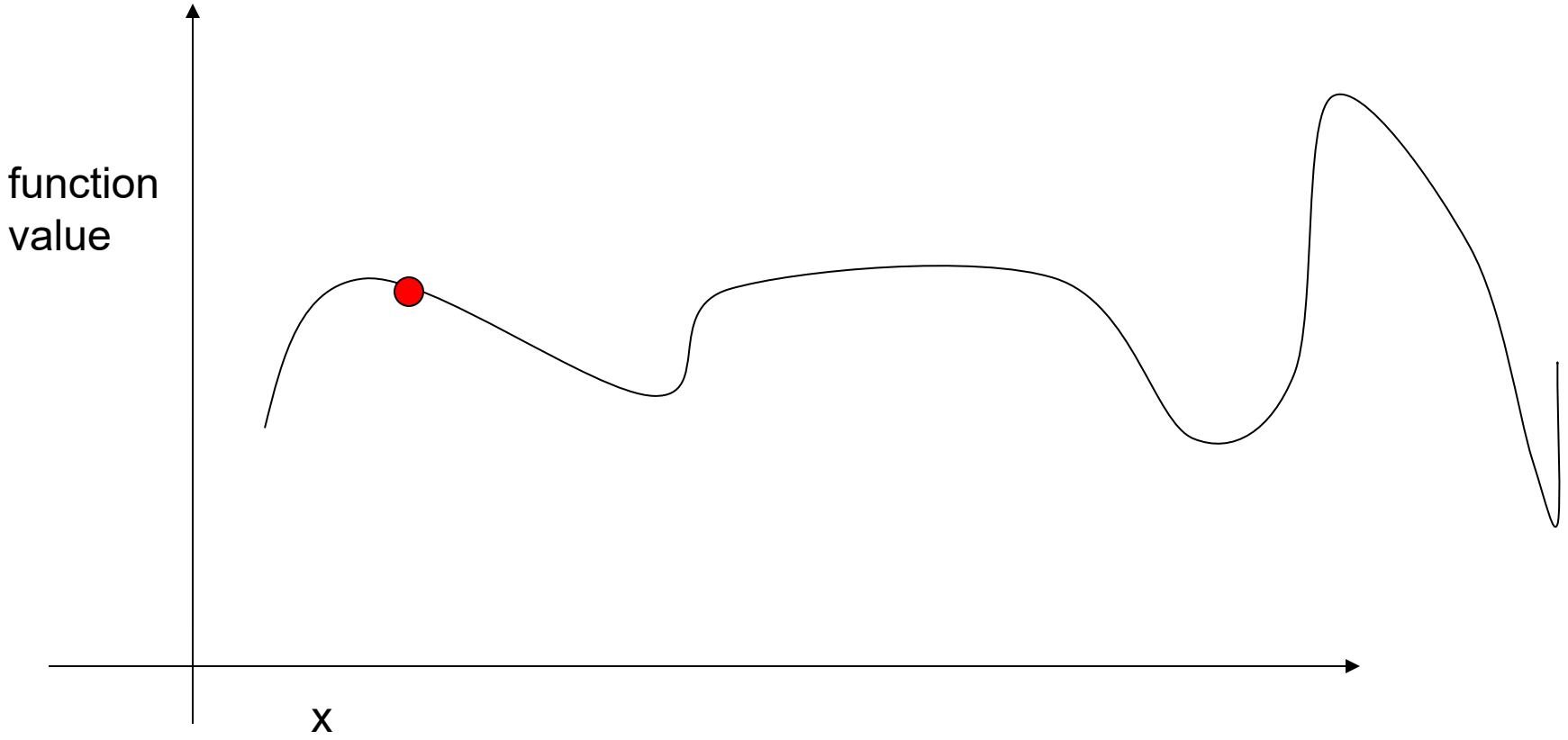
Stochastic Hill Climbing (Ascent)

- Repeat



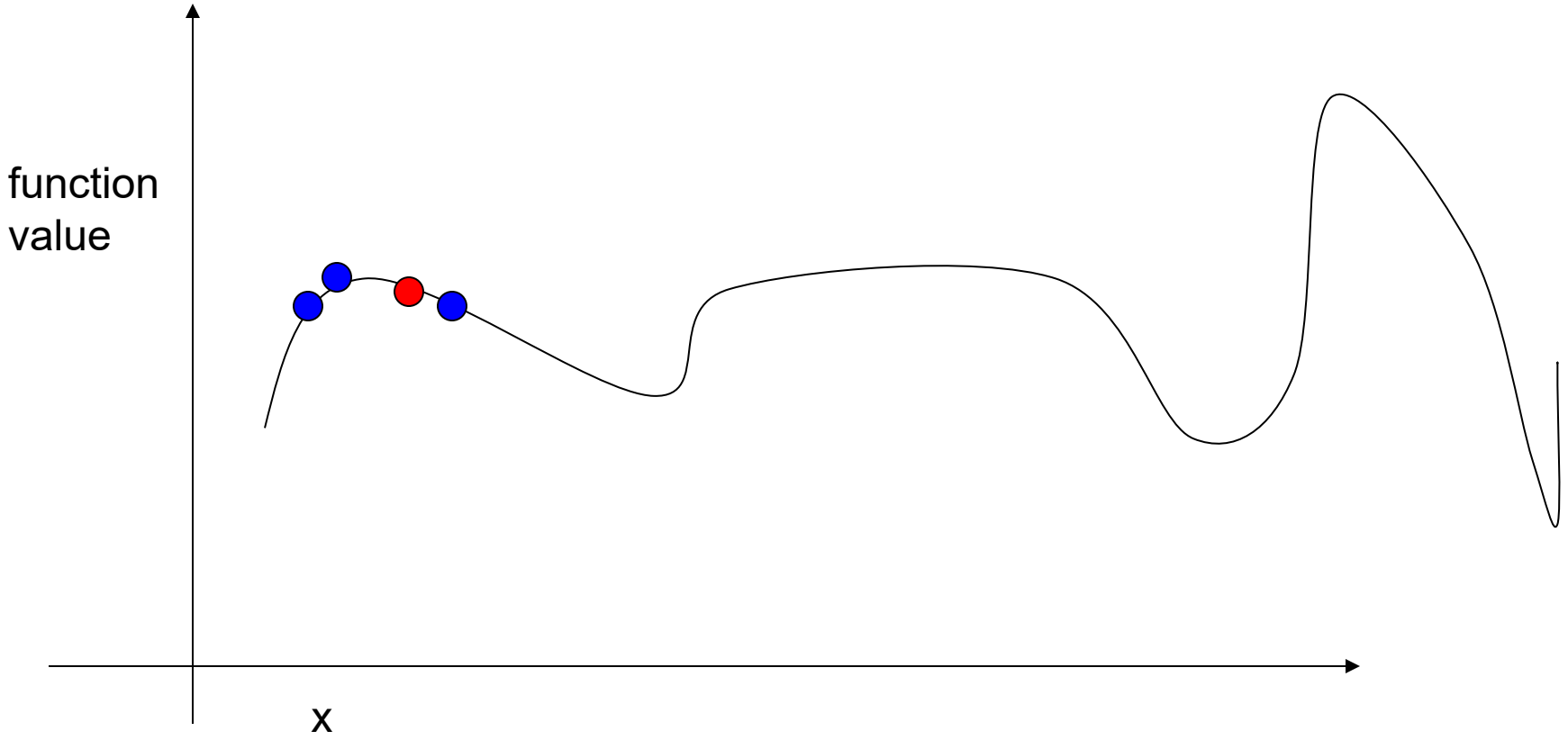
Stochastic Hill Climbing (Ascent)

- Repeat



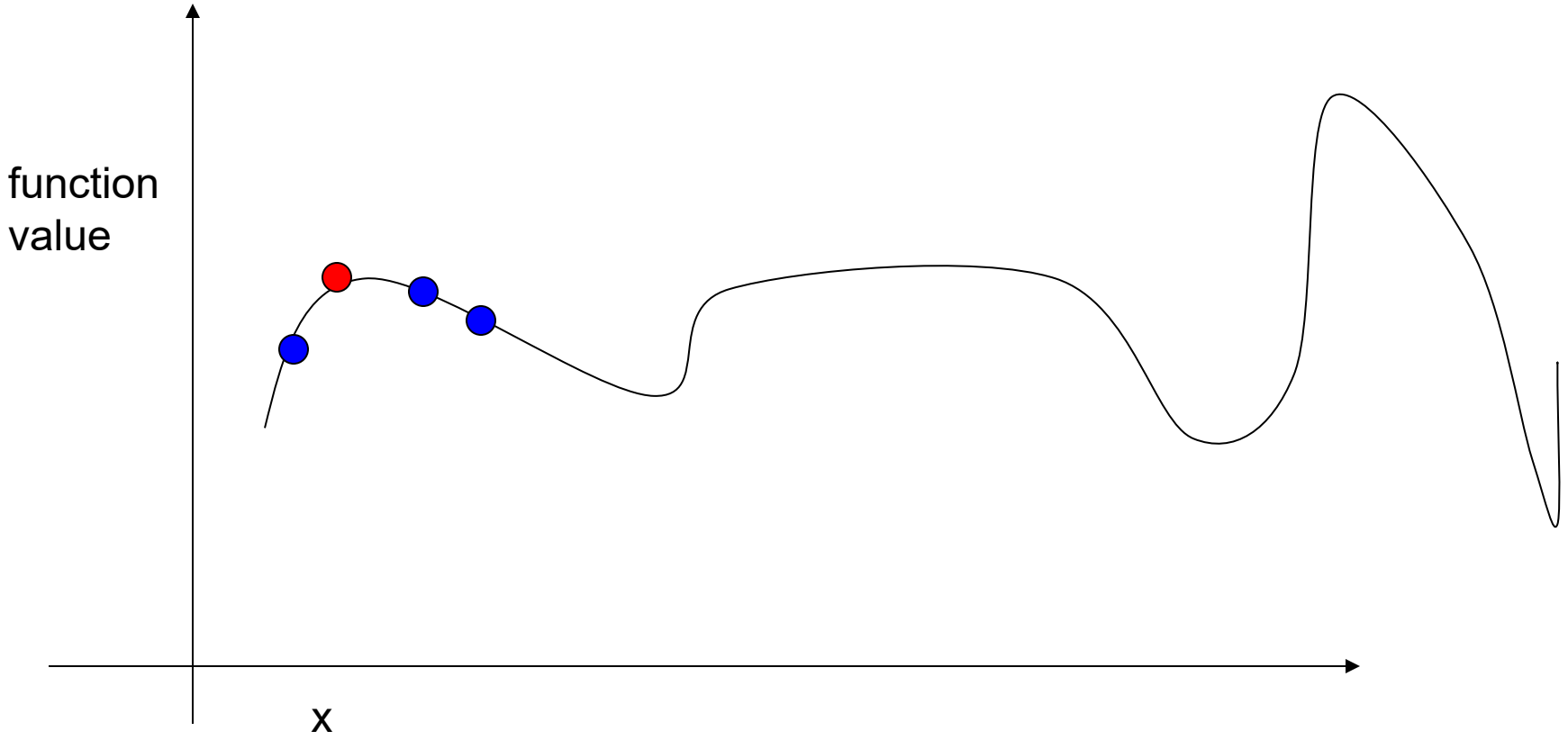
Stochastic Hill Climbing (Ascent)

- Repeat



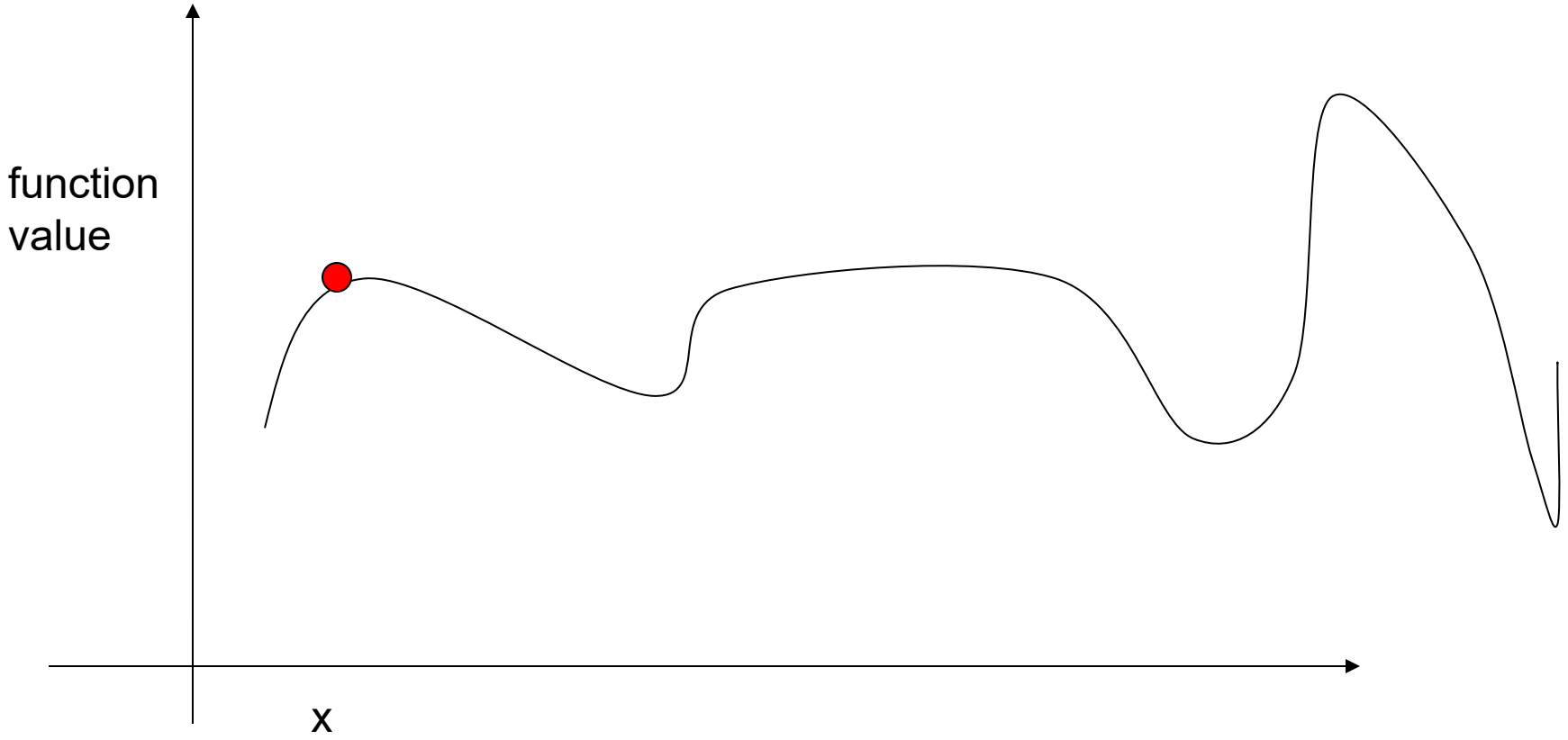
Stochastic Hill Climbing (Ascent)

- Repeat



Stochastic Hill Climbing (Ascent)

- No Improvement, so stop.



Gradient Descent/Ascent (What We Want!)

- 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

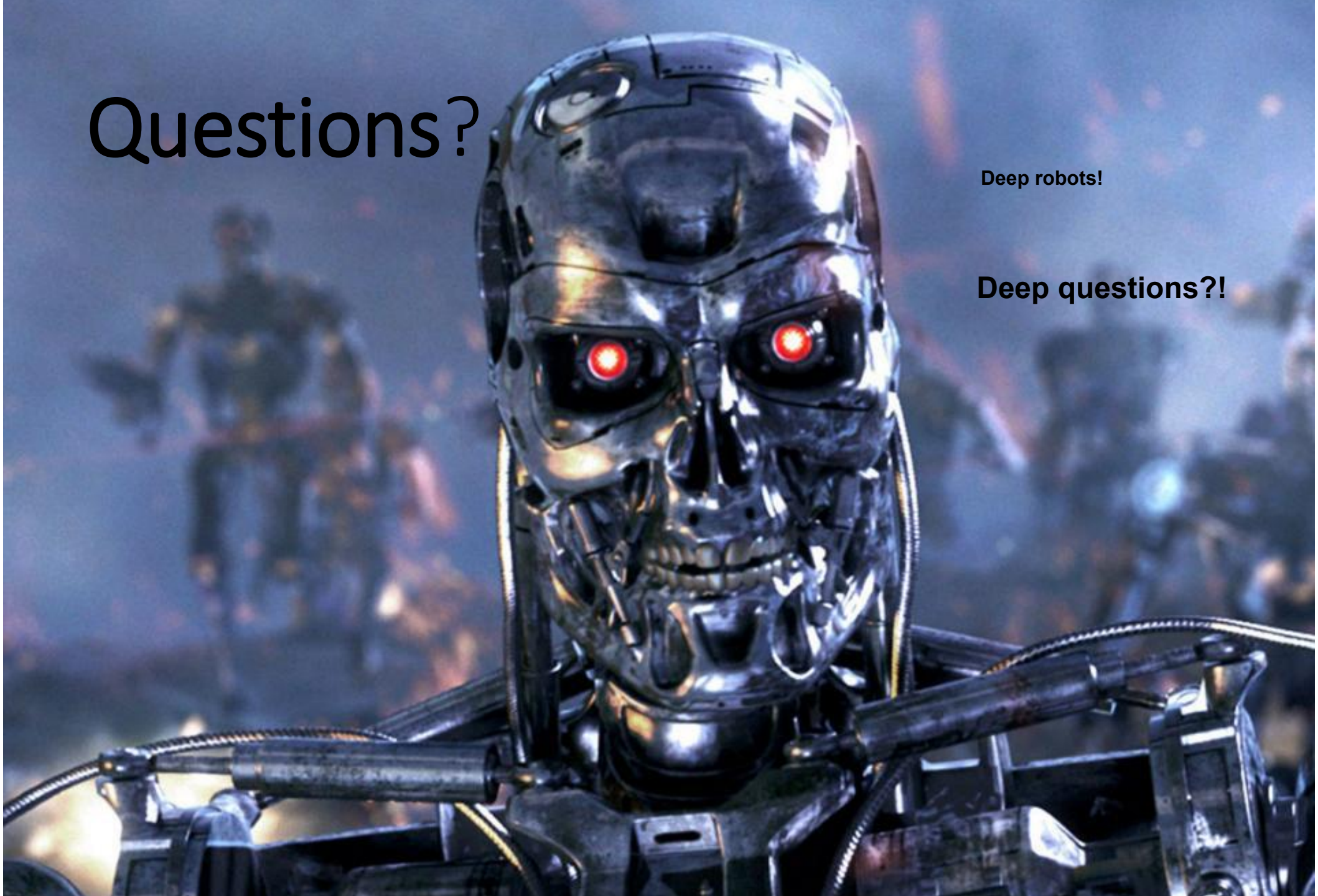
Gradient methods compute

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

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

Gradients Means Derivatives; This Means Some Calculus…

- [***See Crash Course on Derivatives***]


Questions?
Deep robots!
Deep questions?!

More Applied Optimization

Alexander G. Ororbia II

Introduction to Machine Learning

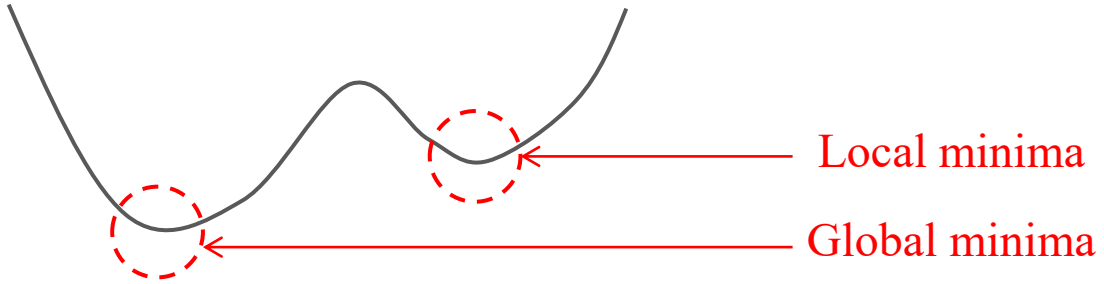
CSCI-335

9/9/2026

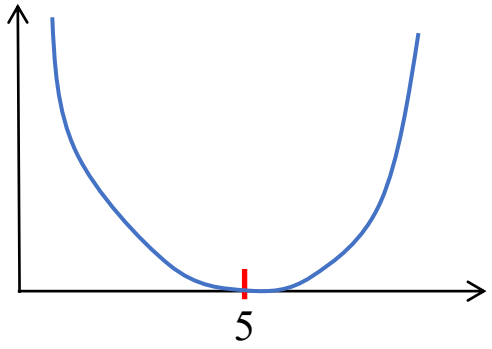
The Value of Derivatives

- How do you solve the following problems?

(1)



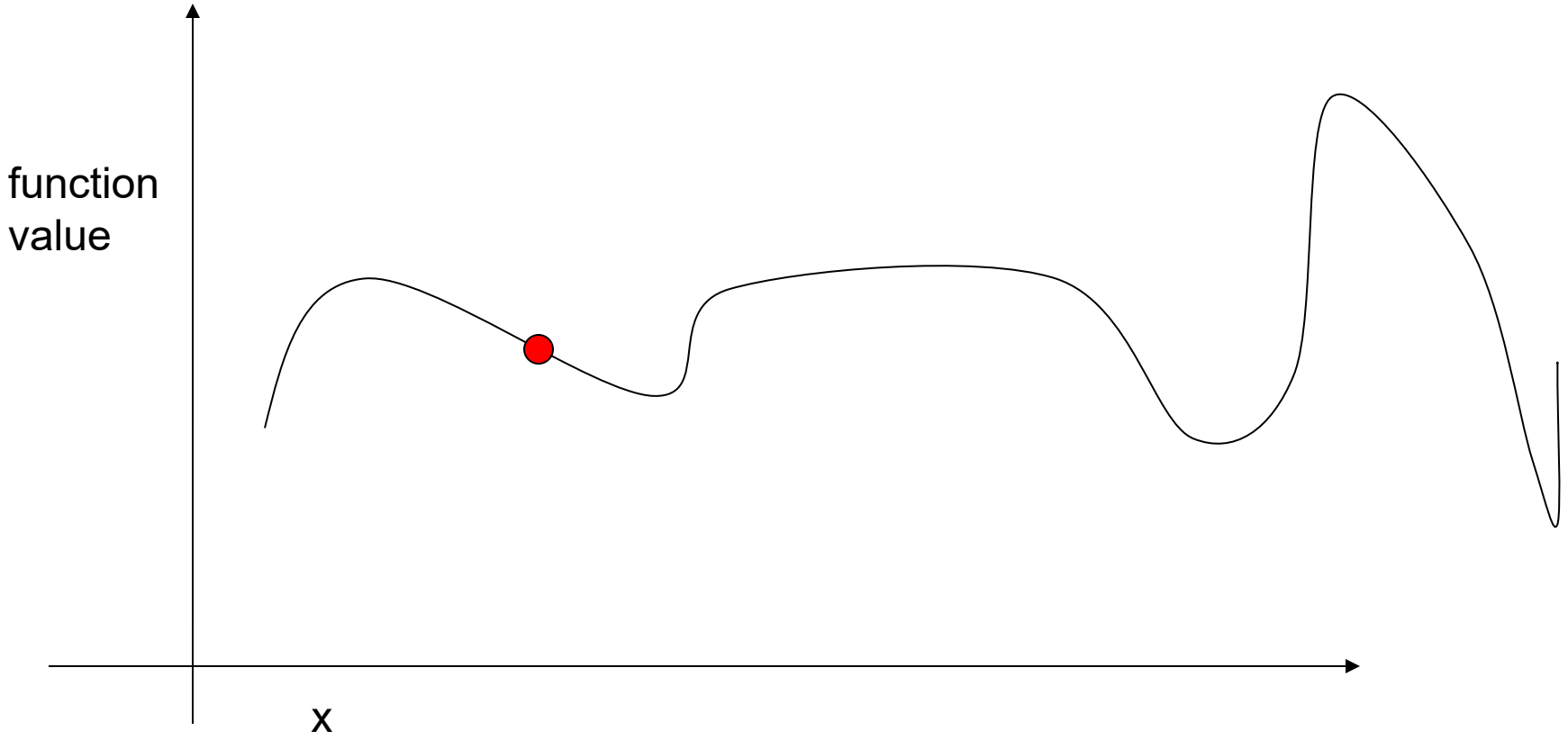
(2) $\min_{w} f(w) = (w-5)^2$ ⟹ (a) Plot



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

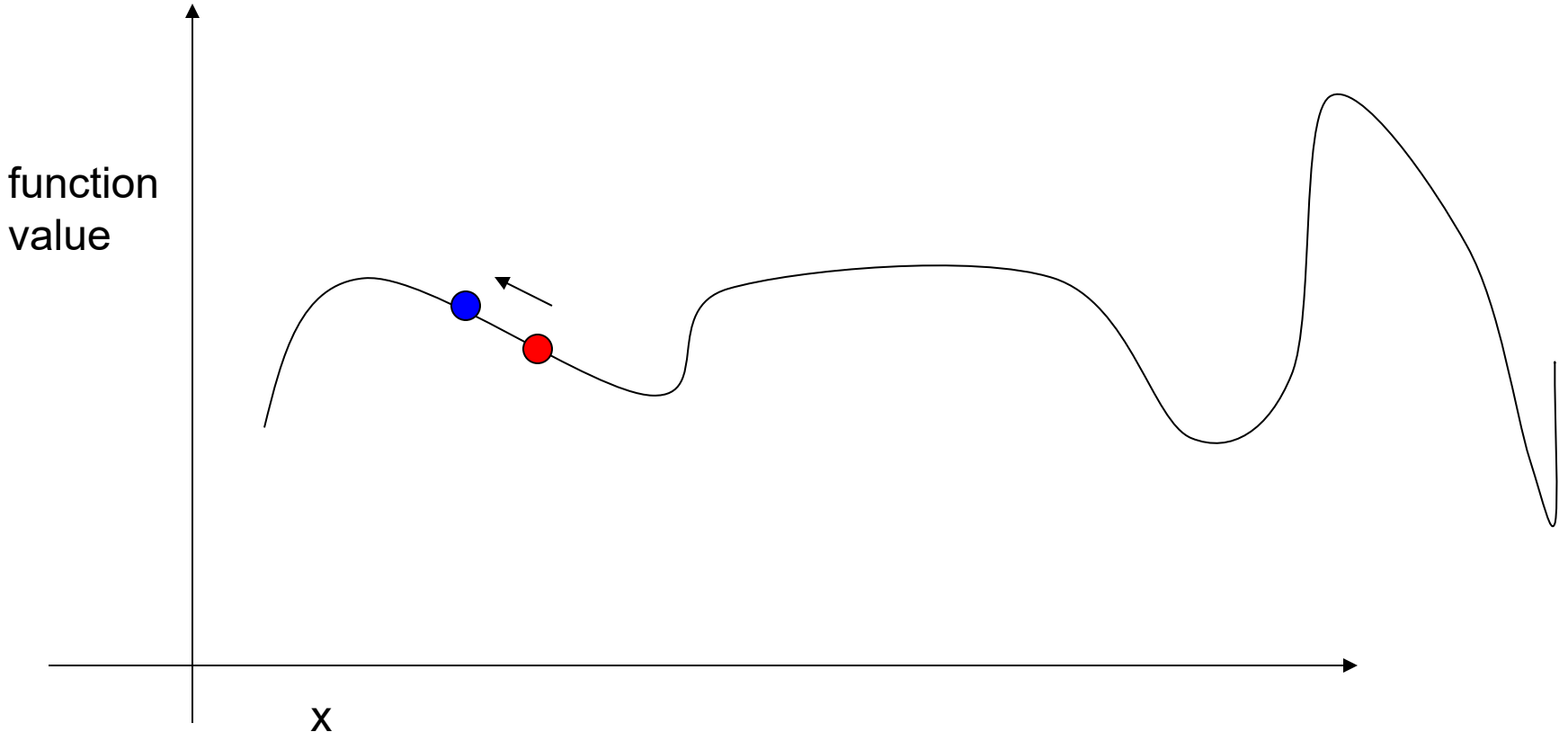
Gradient **Ascent**

- Random Starting Point



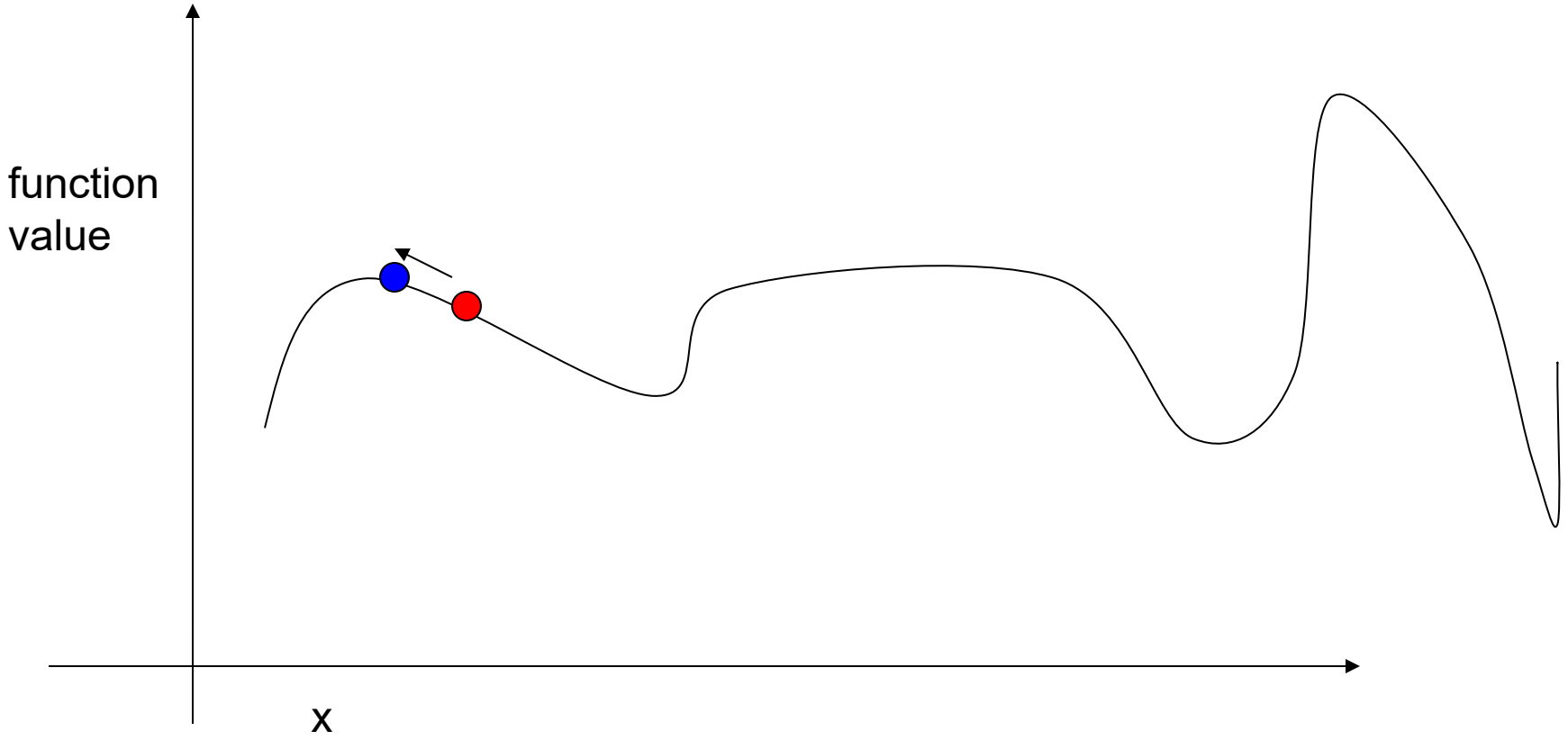
Gradient Ascent

- Take step in direction of largest increase (obvious in 1D, must be computed in higher dimensions)



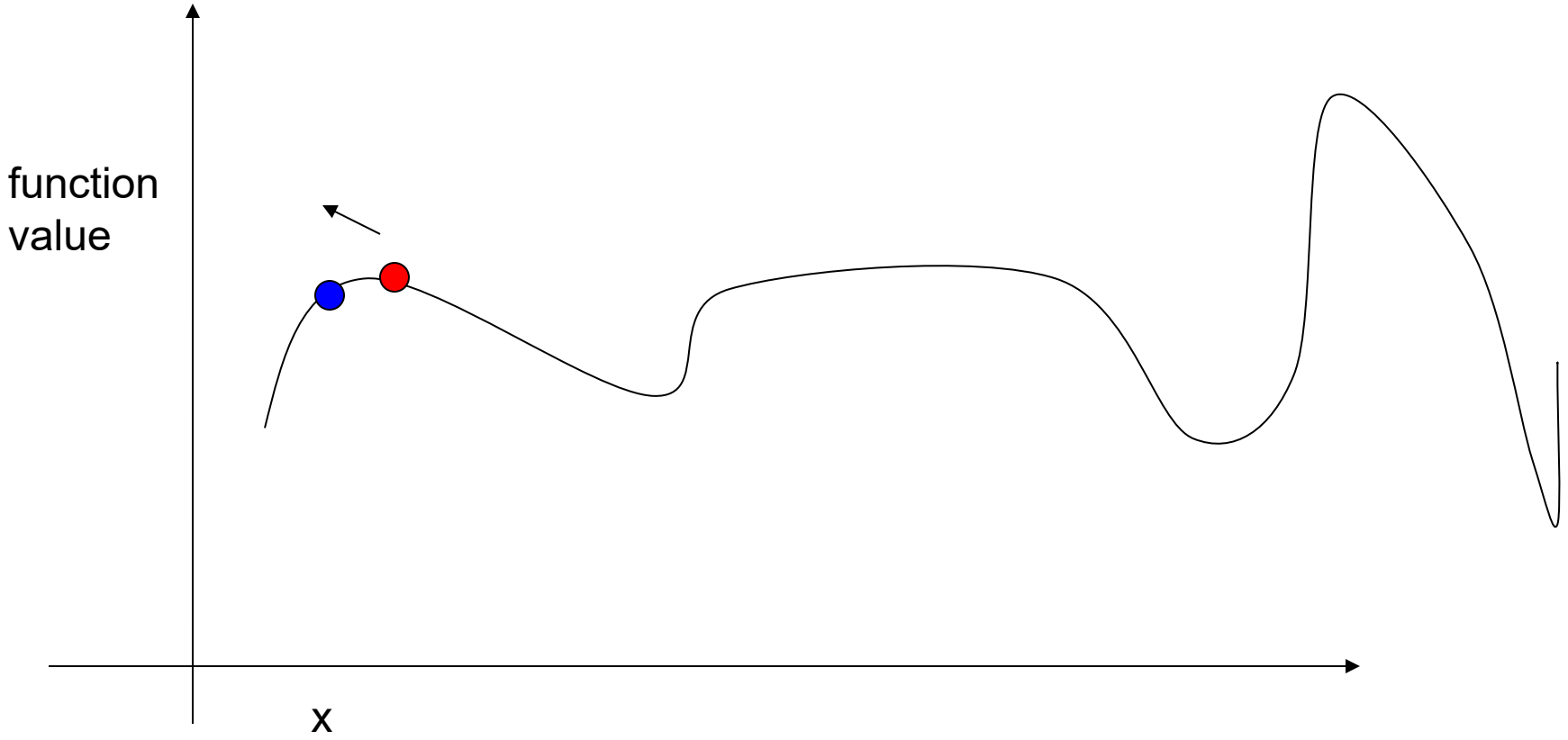
Gradient Ascent

- Repeat



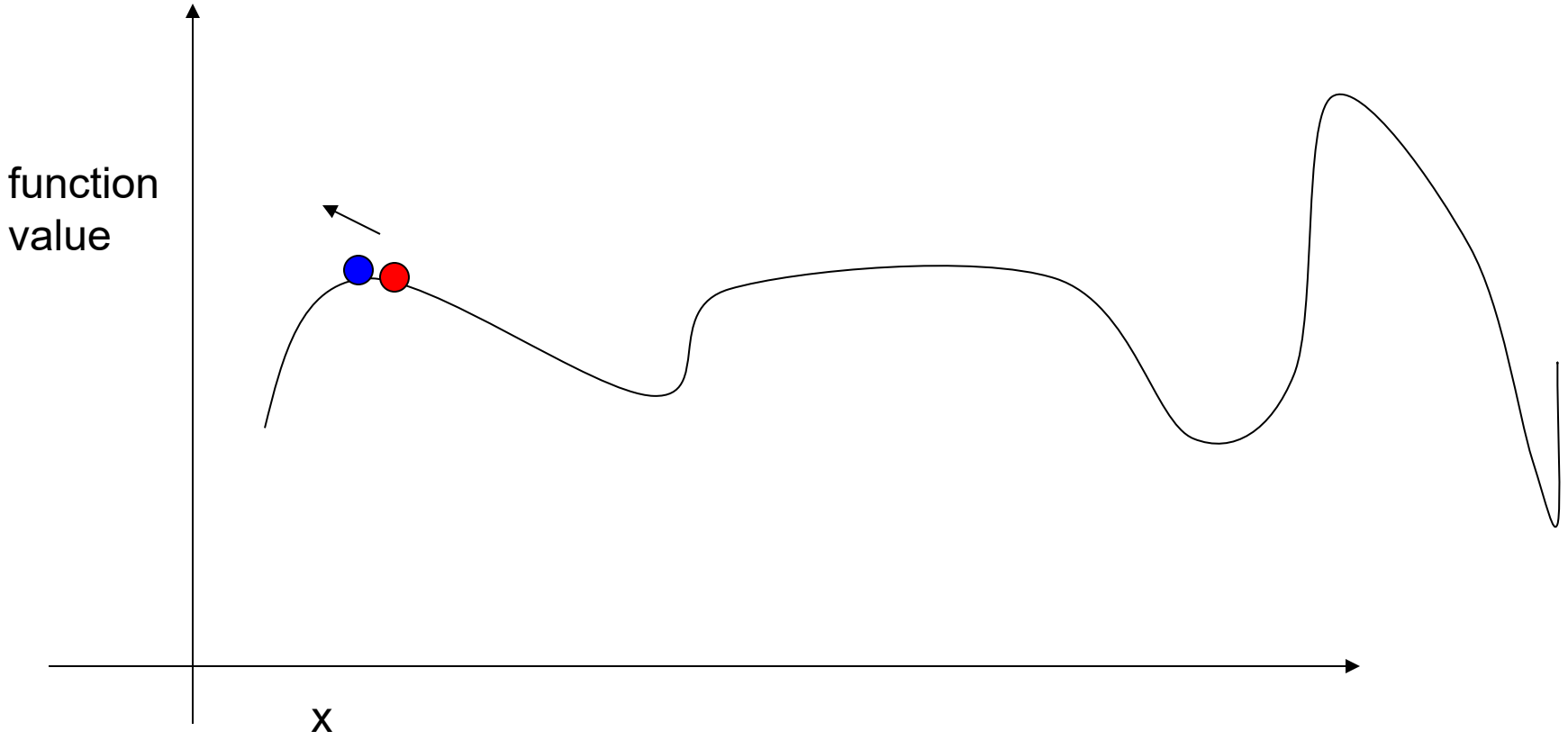
Gradient Ascent

- Next step is actually lower, so stop



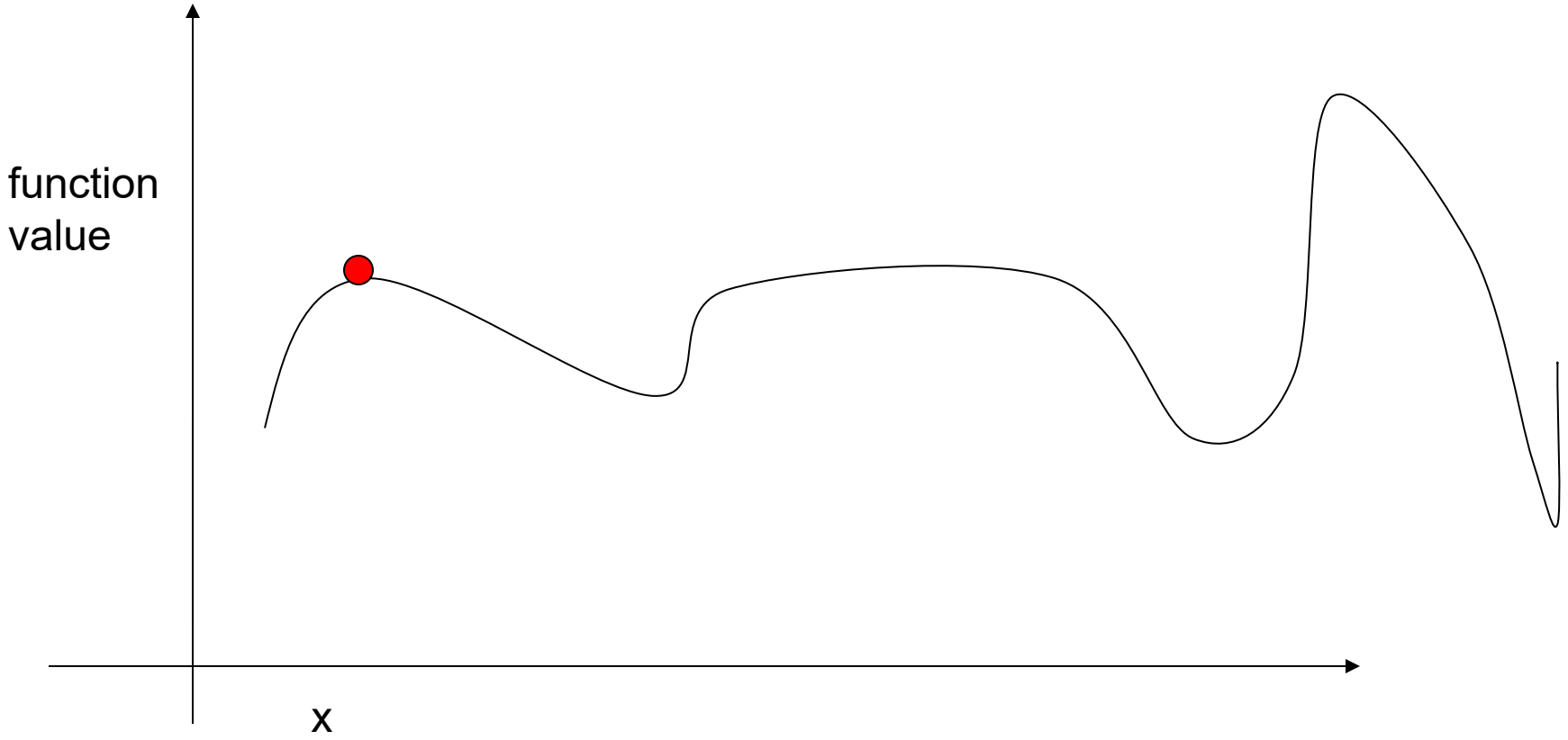
Gradient Ascent

- Could reduce step size to “hone in”



Gradient Ascent

- Converge to (local) maximum



The Finite Difference Method

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

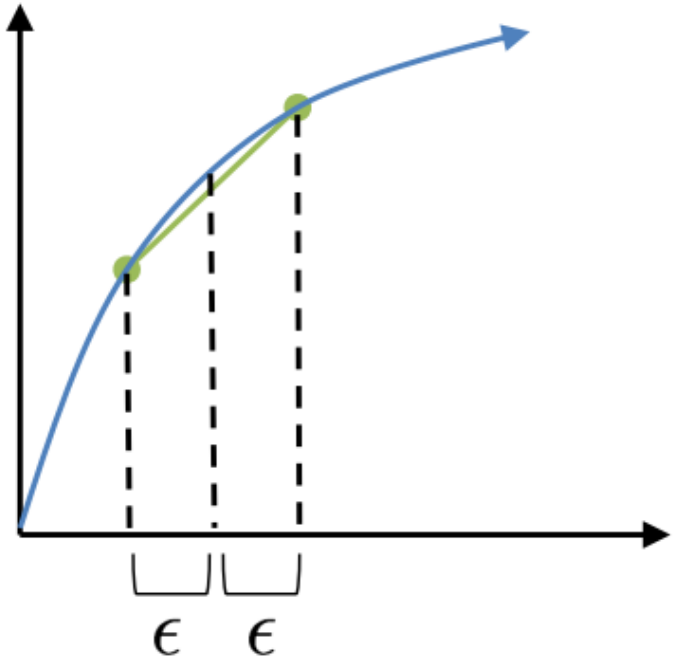
The *centered* finite difference approximation is:

$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{(J(\boldsymbol{\theta} + \epsilon \cdot \boldsymbol{d}_i) - J(\boldsymbol{\theta} - \epsilon \cdot \boldsymbol{d}_i))}{2\epsilon}$$

where $\boldsymbol{d}_i$ is a 1-hot vector consisting of all zeros except for the ith entry of $\boldsymbol{d}_i$, which has value 1.

Notes:

- Suffers from issues of floating point precision, in practice
- Typically only appropriate to use on small examples with an appropriately chosen epsilon



73

Ex: Gradient Descent on Least Squares

- Criterion to minimize

$$f(\mathrm{x}) = \frac{1}{2} || A\mathrm{x} - \mathrm{b} ||^2$$

Evaluation

  - Least squares regression $E_D(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathrm{w}^T\phi(x_n)\}^2$

- The gradient is

$$\nabla_x f(\boldsymbol{x}) = A^T(A\boldsymbol{x} - b) = A^T A\boldsymbol{x} - A^T\boldsymbol{b}$$

Optimization

- Gradient Descent algorithm is

1. Set step size ε, tolerance δ to small, positive nos.
2. *while* $|| A^T A\boldsymbol{x} - A^T\boldsymbol{b} ||_2 > \delta$ do

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \varepsilon(A^T A\boldsymbol{x} - A^T\boldsymbol{b})$$

3. *end while*

Calculus in Optimization

- Suppose we have function $y=f(x)$, x, y real nos.
  - Derivative of function denoted: $f'(x)$ or as dy/dx
    - Derivative $f'(x)$ gives the slope of $f(x)$ at point x
    - It specifies how to scale a small change in input to obtain a corresponding change in the output:

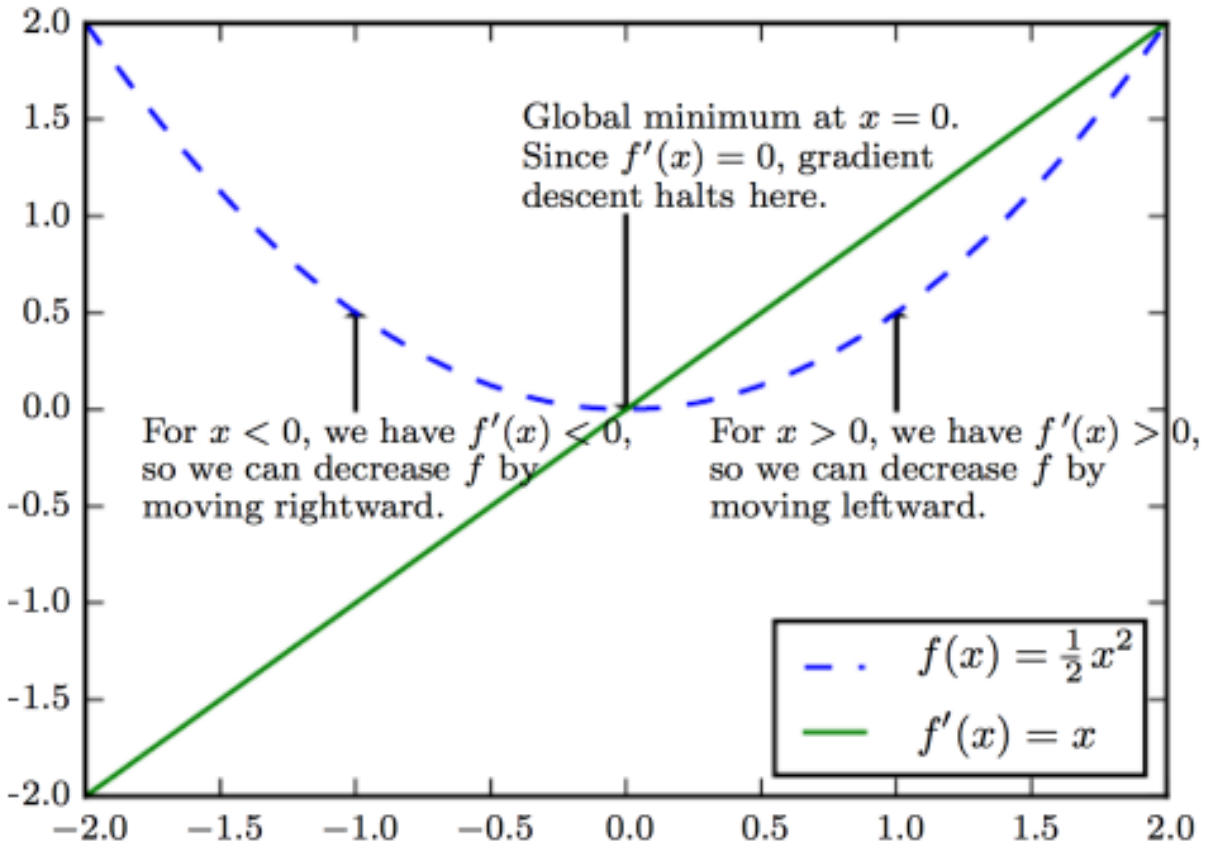
$$f(x+\varepsilon) \approx f(x) + \varepsilon f'(x)$$

  - It tells how you make a small change in input to make a small improvement in y
  - We know that $f(x - \varepsilon \text{ sign } (f'(x)))$ is less than $f(x)$ for small ε. Thus we can reduce $f(x)$ by moving x in small steps with opposite sign of derivative
    - This technique is called *gradient descent* (Cauchy 1847)

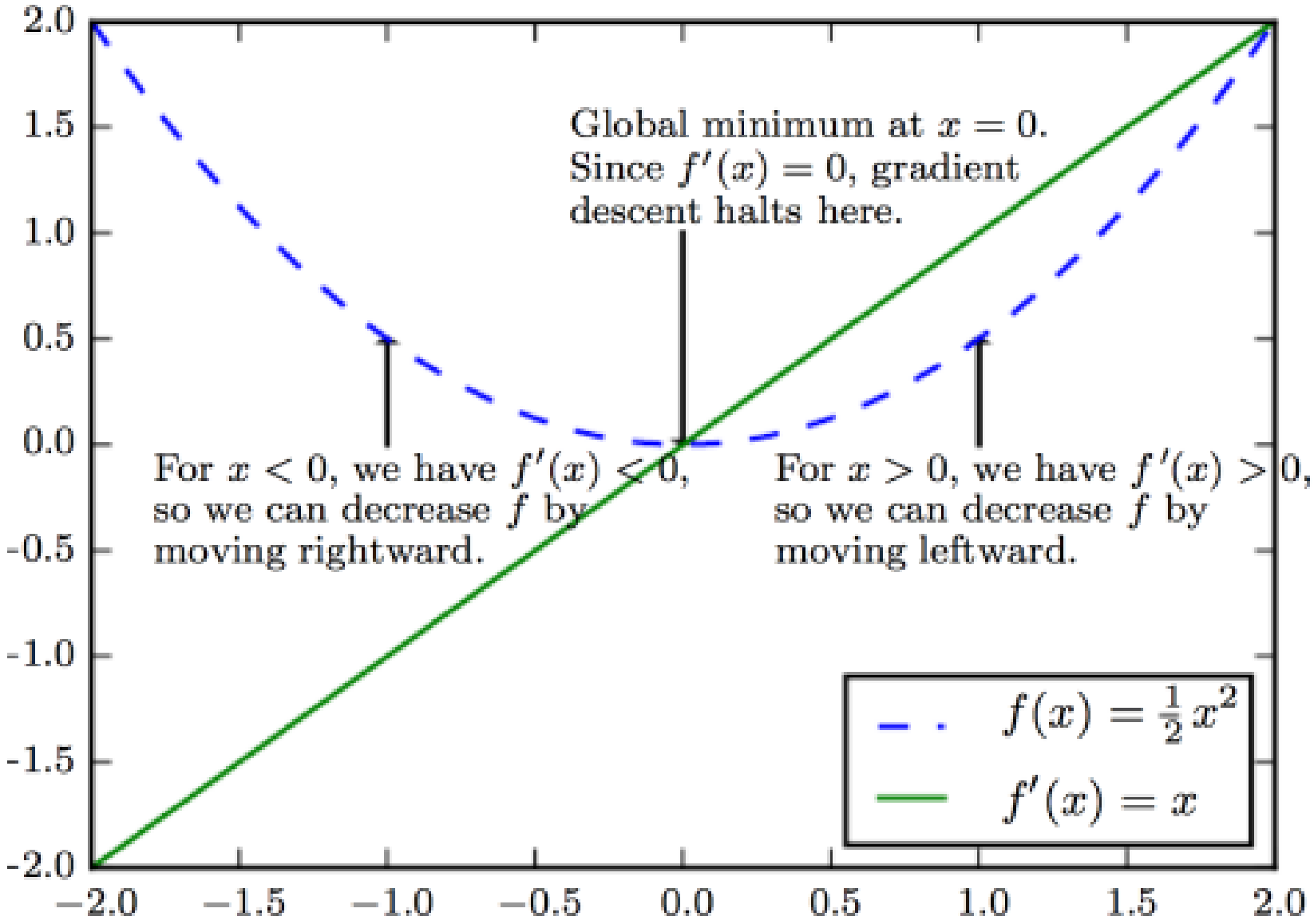
signum(v)

Gradient Descent Illustrated

- For $x>0$, $f(x)$ increases with x and $f'(x)>0$
- For $x<0$, $f(x)$ is decreases with x and $f'(x)<0$
- Use $f'(x)$ to follow function downhill
- Reduce $f(x)$ by going in direction opposite sign of derivative $f'(x)$

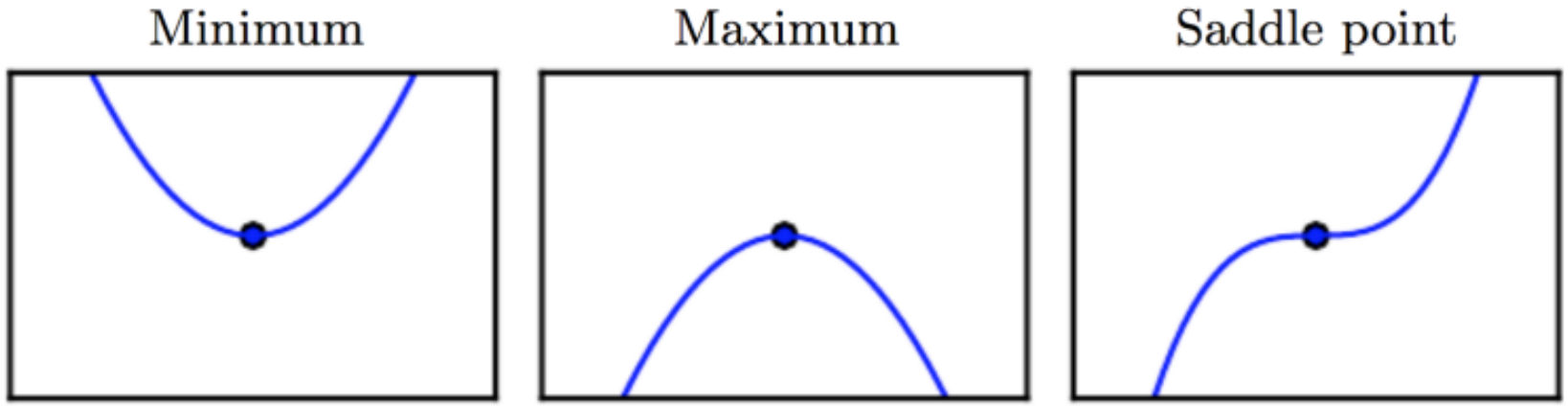


Gradient Descent Illustrated



Stationary points, Local Optima

- When $f'(x)=0$ derivative provides no information about direction of move
- Points where $f'(x)=0$ are known as *stationary* or critical points
  - Local minimum/maximum: a point where $f(x)$ lower/ higher than all its neighbors
  - Saddle Points: neither maxima nor minima



Functions with Multiple Inputs

- Need partial derivatives
- $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ measures how f changes as only variable x_i increases at point $\boldsymbol{x}$
- Gradient generalizes notion of derivative where derivative is wrt a vector
- Gradient is vector containing all of the partial derivatives denoted $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
  - Element i of the gradient is the partial derivative of f wrt x_i
  - Critical points are where every element of the gradient is equal to zero

w.r.t. (with respect to)

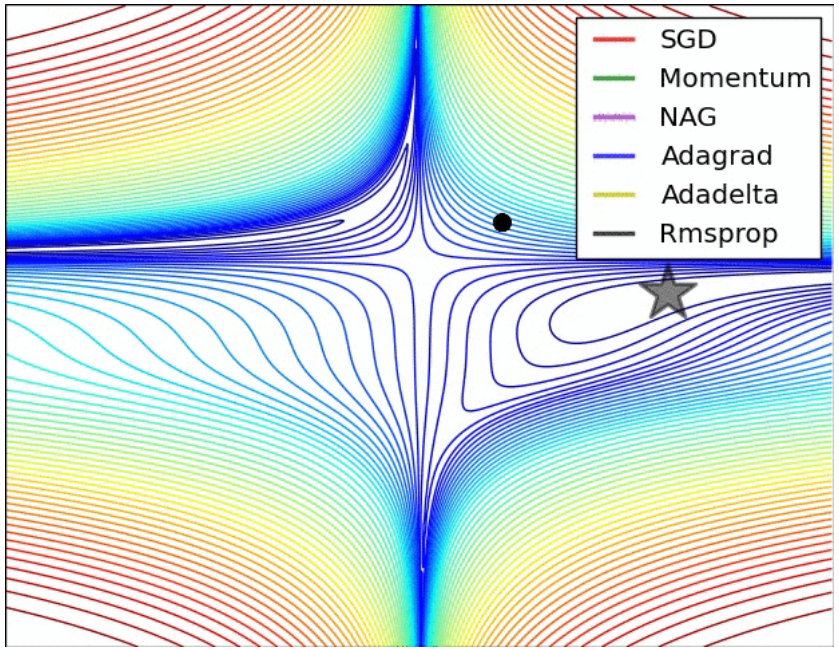
Method of Gradient Descent

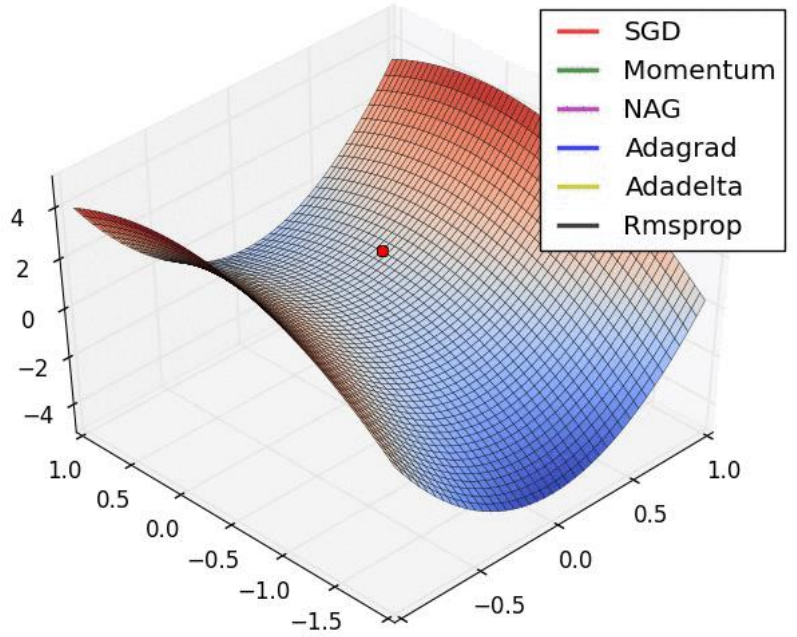
- The gradient points directly uphill, and the negative gradient points directly downhill
- Thus we can decrease f by moving in the direction of the negative gradient
  - This is known as the method of steepest descent or gradient descent
- Steepest descent proposes a new point

$$\boldsymbol{x}' = \boldsymbol{x} - \varepsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

  - where ε is the learning rate, a positive scalar. Set to a small constant.

Race of the Optimizers!

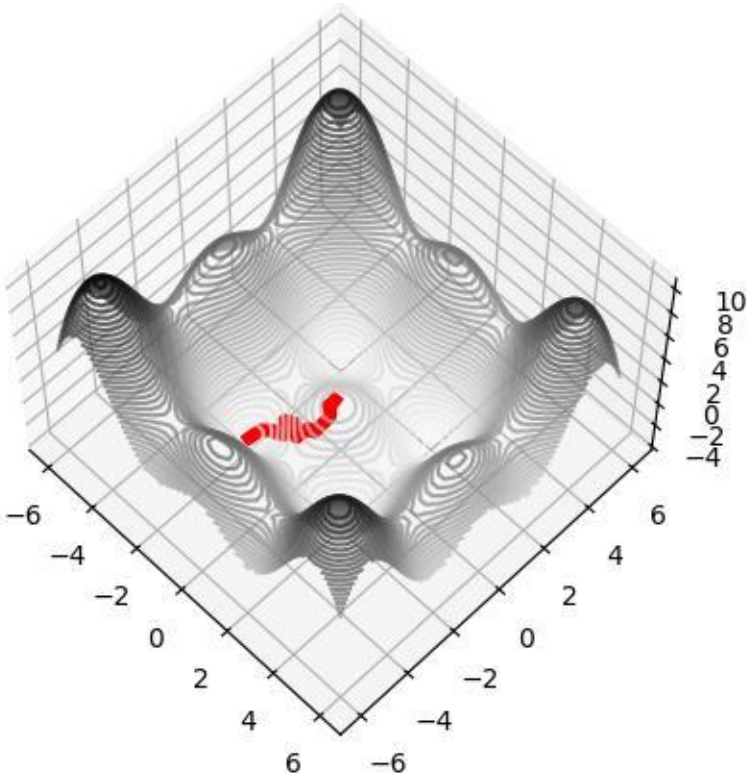


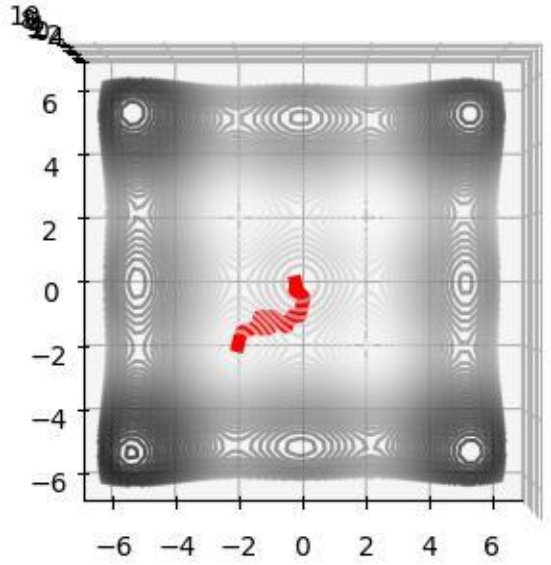


http://cs231n.github.io/neural-networks-3/#hyper

Stochastic Hill-Climbing

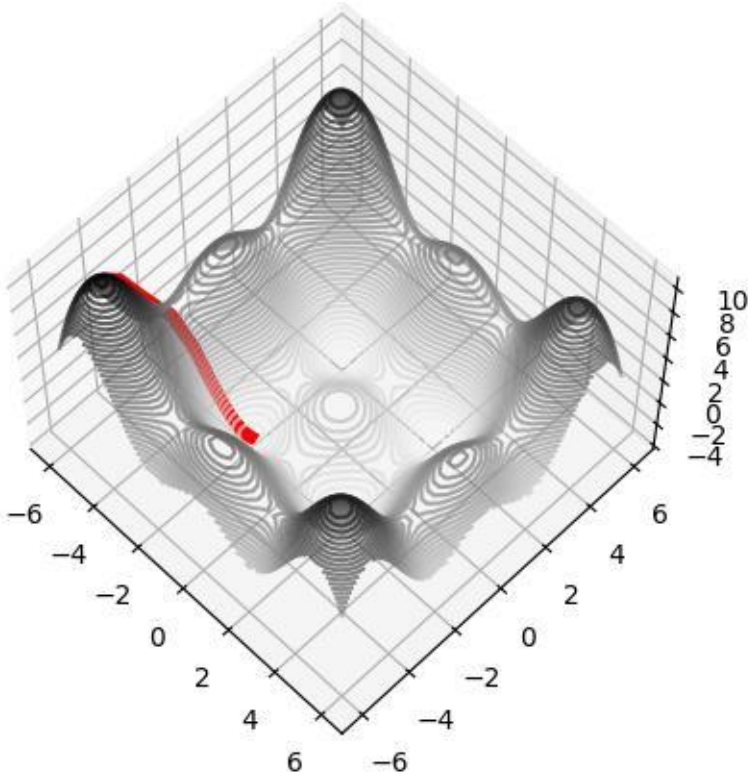
Negative Alpine Function $f(x) = -\sum_i (x_i \sin(x_i) + 0.1\, x_i)$

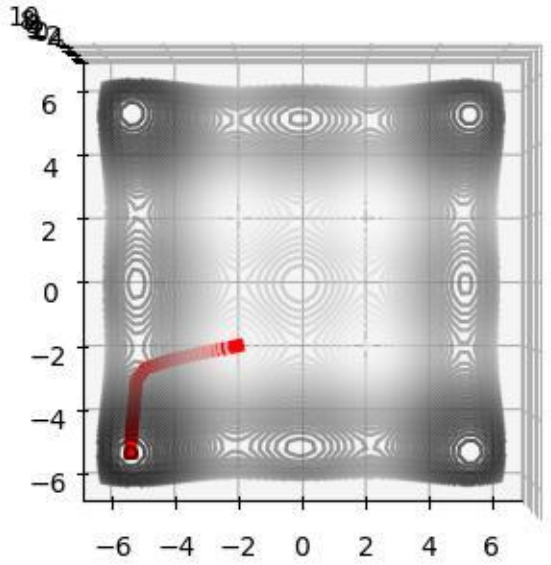




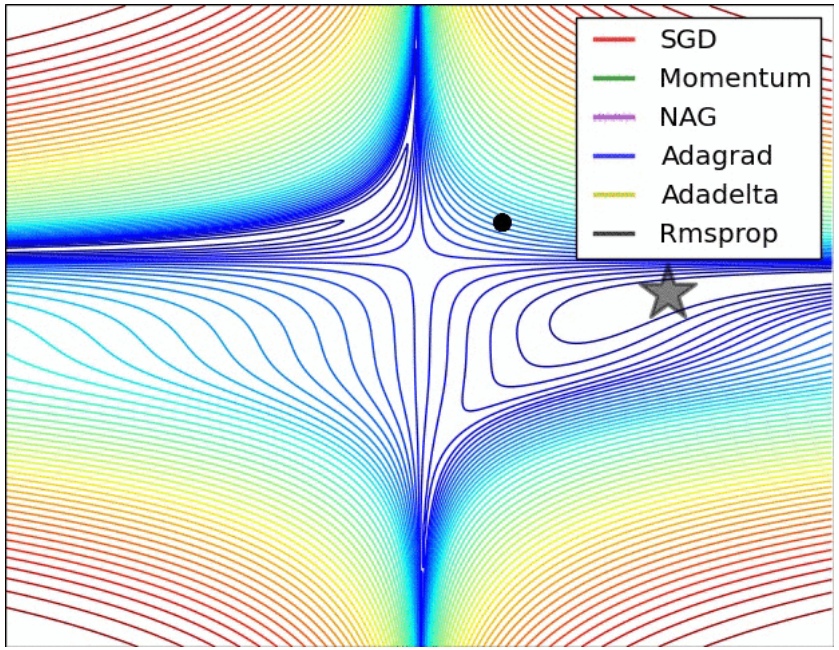
Stochastic Gradient Ascent

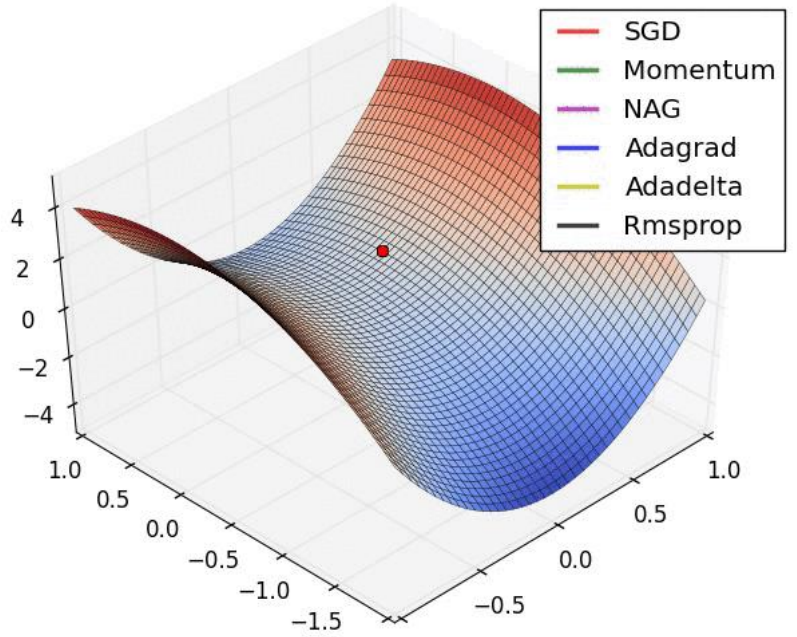
Negative Alpine Function $f(x) = -\sum_i (x_i \sin(x_i) + 0.1\, x_i)$





Race of the Optimizers!

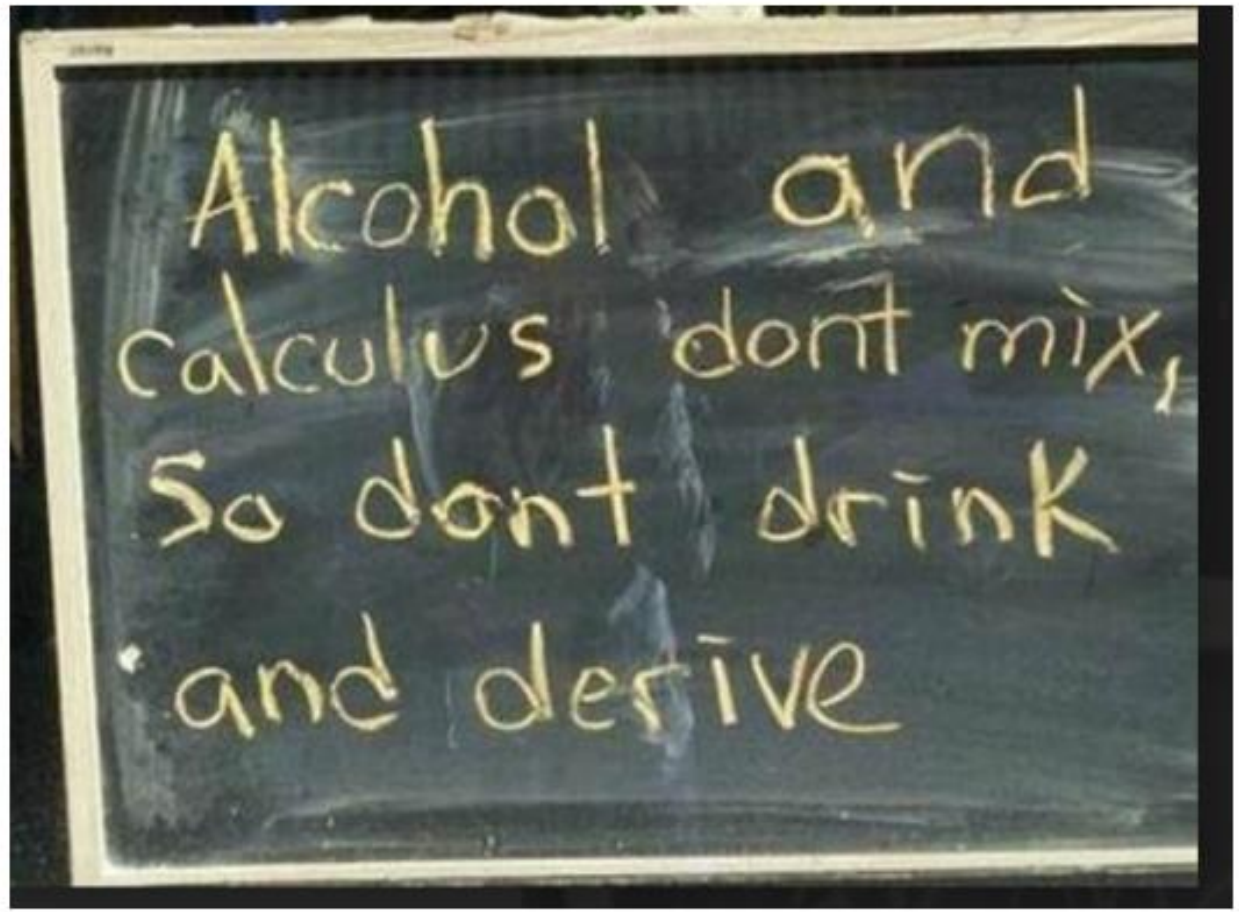




http://cs231n.github.io/neural-networks-3/#hyper

Gradient

- Essential role of calculus



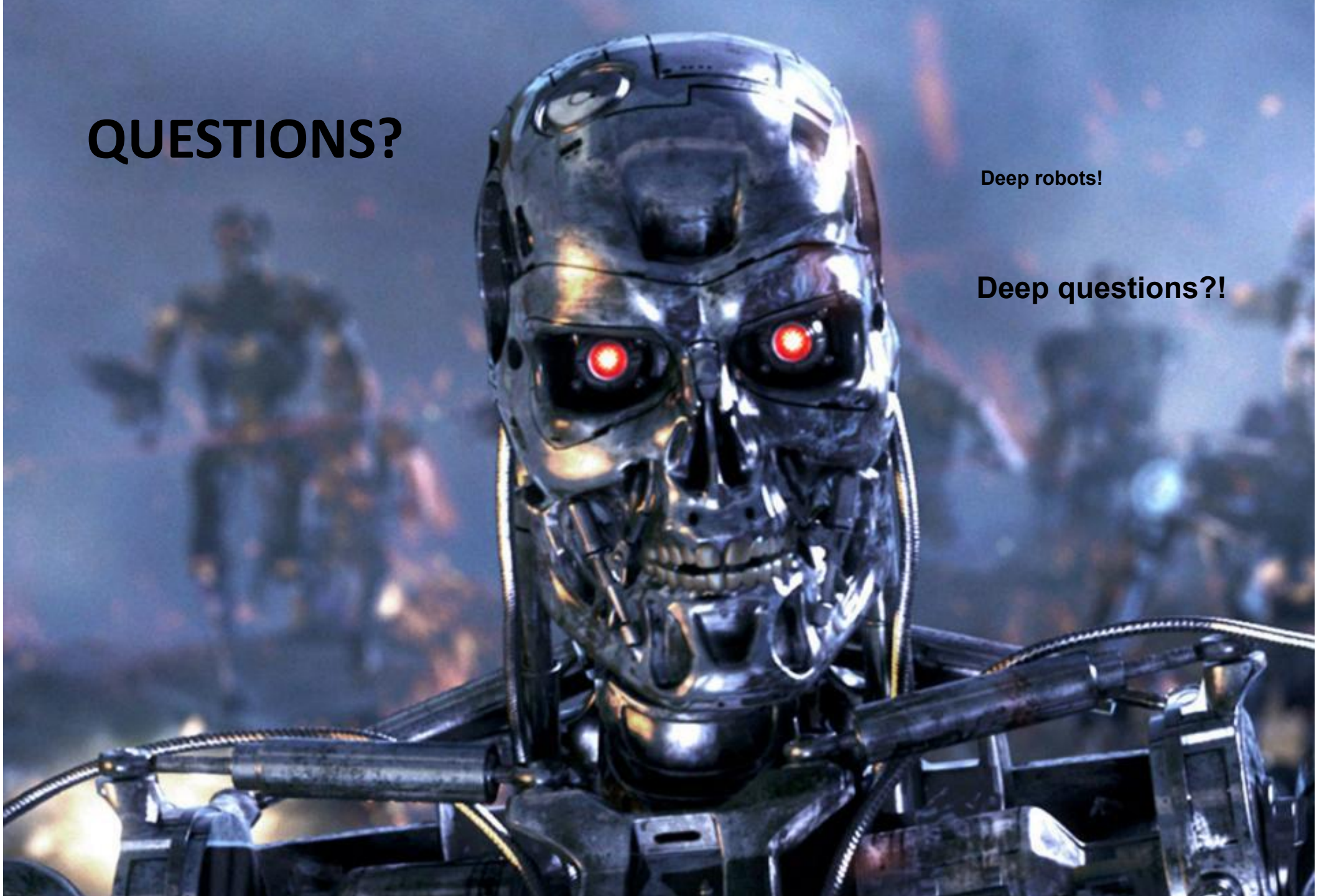
You want to build a simple univariate regression model for predicting profits y for a food truck. Furthermore, you decide to restrict yourself to a linear hypothesis space and construct a model that adheres to the following form:

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

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)


QUESTIONS?
Deep robots!
Deep questions?!