16-745
Optimal Control
and
Reinforcement Learning
2026

Chris Atkeson, 1/12/26

Be Zac!

I am supposed to teach what Zac taught.

Zac's slides and lecture recordings are available ....

I expect you to at least look at the slides, and scan the lectures to cover anything you don't already know. I listen to the lectures at 1.5X.

Class web page

https://www.cs.cmu.edu/~cga/dynopt/

Go over this page now …

Starting section on Function Optimization

Optimal control vs. optimization

This is a course about generating (control) or (reinforcement) learning dynamic behavior (things that unfold over time, things we do with our body, vehicles, industrial processes, ...)

This is not a course on optimization. It is a course about using optimization to generate dynamic behavior.

However, we will review function optimization.

Assume this subroutine is available to you: optfun(function, constraints, hints)

Please scan https://en.wikipedia.org/wiki/Mathematical_optimization to see what this subroutine can pick from to do the function optimization. We will not talk about much of this list, such as combinatorial optimization, integer optimization, multiple agents, games, ...

Optimists vs. Pessimists

Maximize reward (psychology, neuroscience, machine learning)

Minimize cost (engineering)

Doesn’t matter

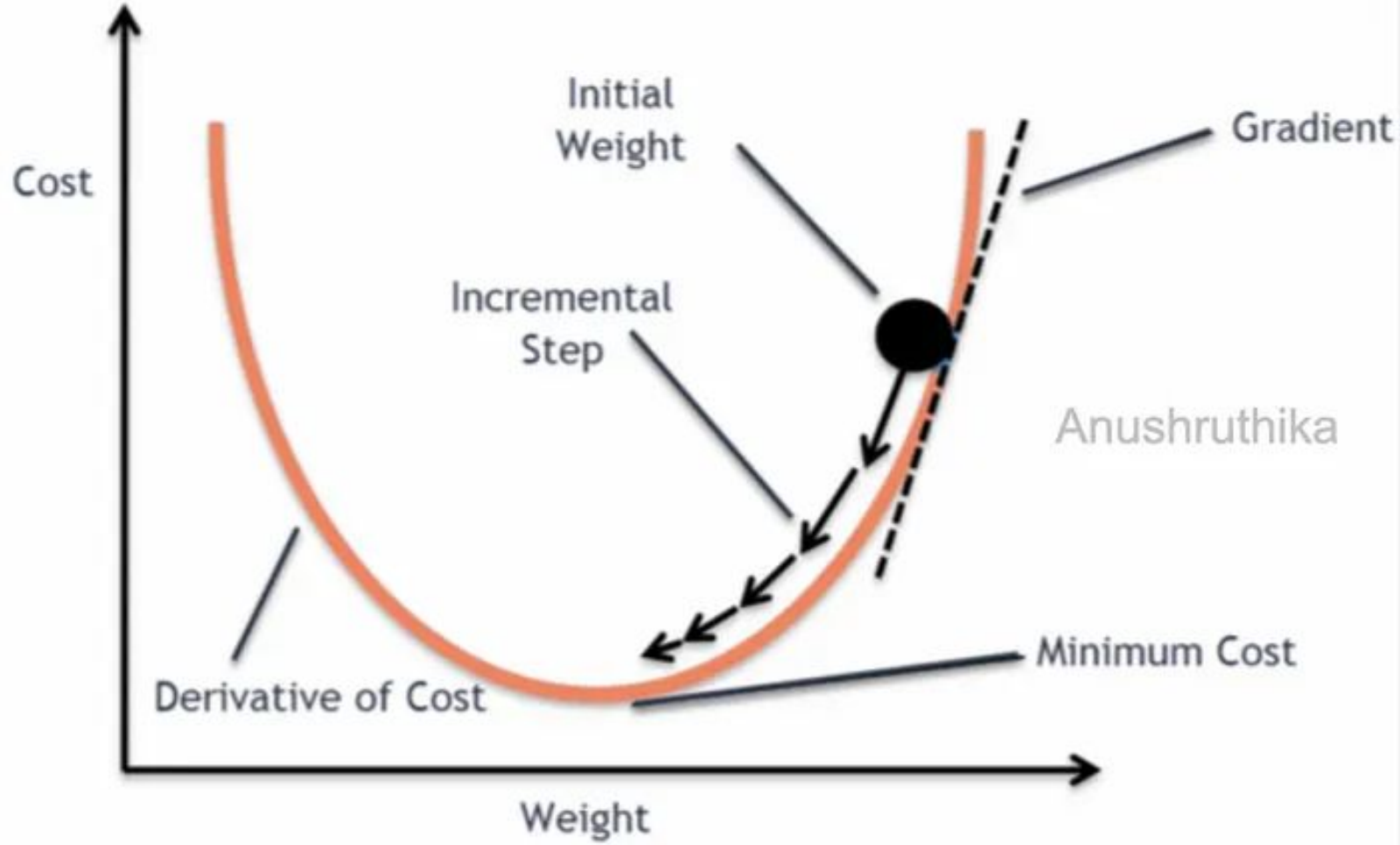
I will typically minimize a cost function (optimization criterion).

Optimization (Engineering) vs. Search (AI)
Optimal Control (Engineering) vs. Reinforcement Learning (ML)

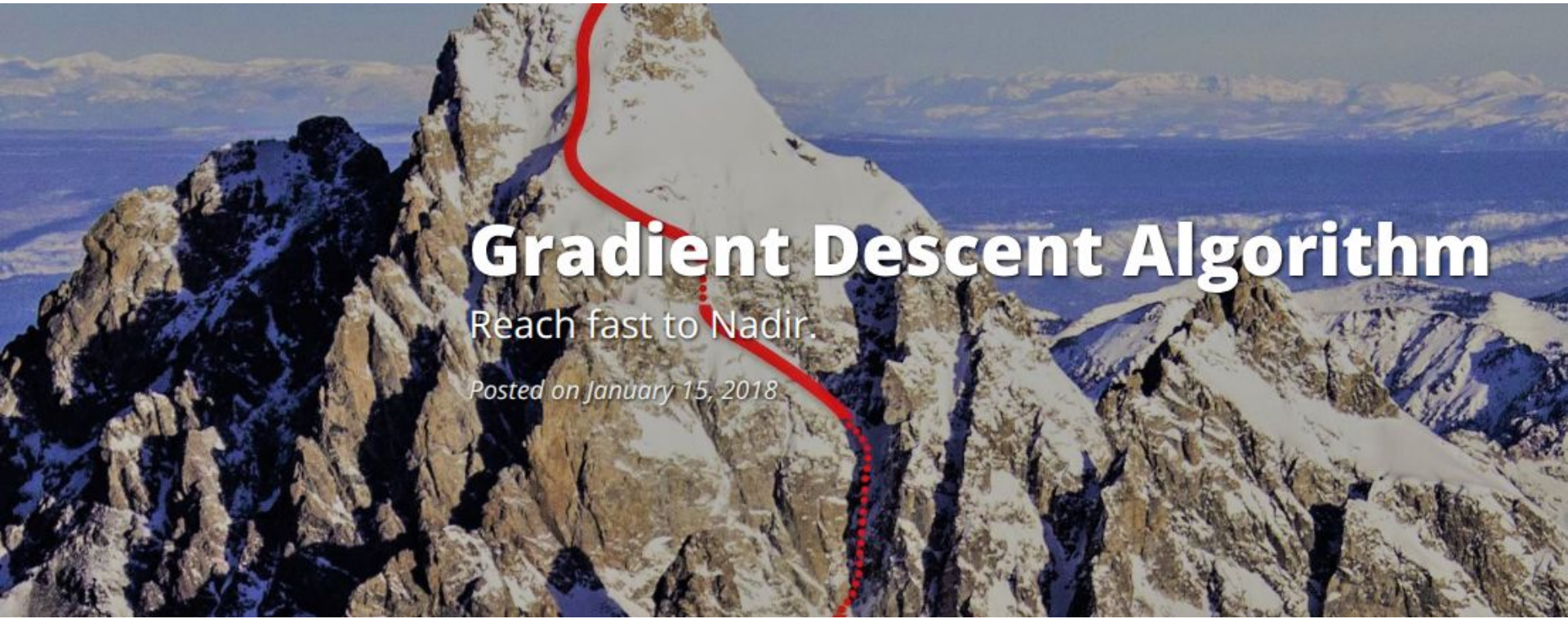
- How do you find local minima?
- How do you find the (or a) global minimum?

- How do you find local minima?
  - Make random guesses and iterate. New guesses are close to previous good guesses. Or do a more organized search pattern. Or imitate evolution.
-

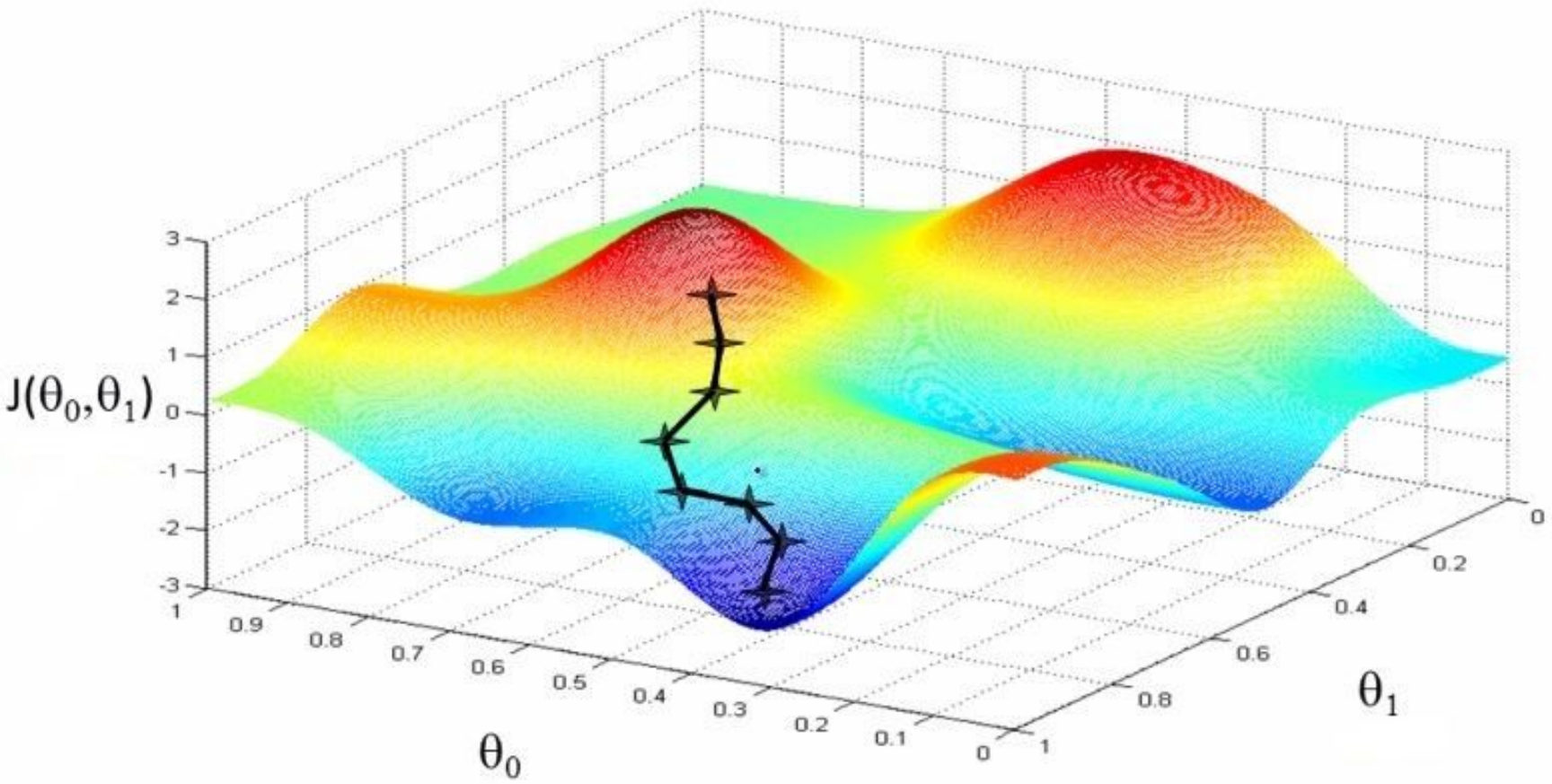
- How do you find local minima?
  - Figure out which way is downhill and move a little bit in that direction (step size α (alpha) or ε (epsilon)) (This is gradient descent: update = - step_size * gradient)
    - Line search



https://medium.com/@anushruthikae/optimizing-machine-learning-a-deep-dive-into-cost-functions-and-gradient-descent-7694ca75cc32

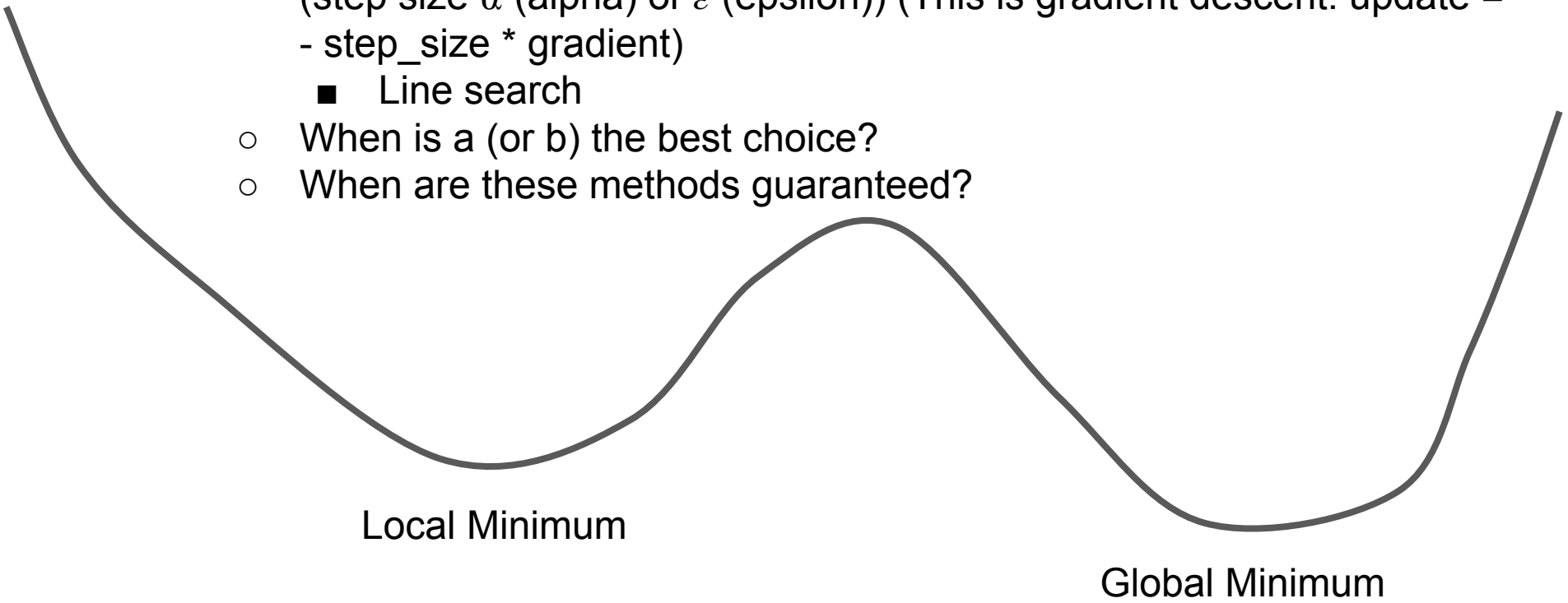


https://ruthwik.github.io/machinelearning/2018-01-15-gradient-descent/

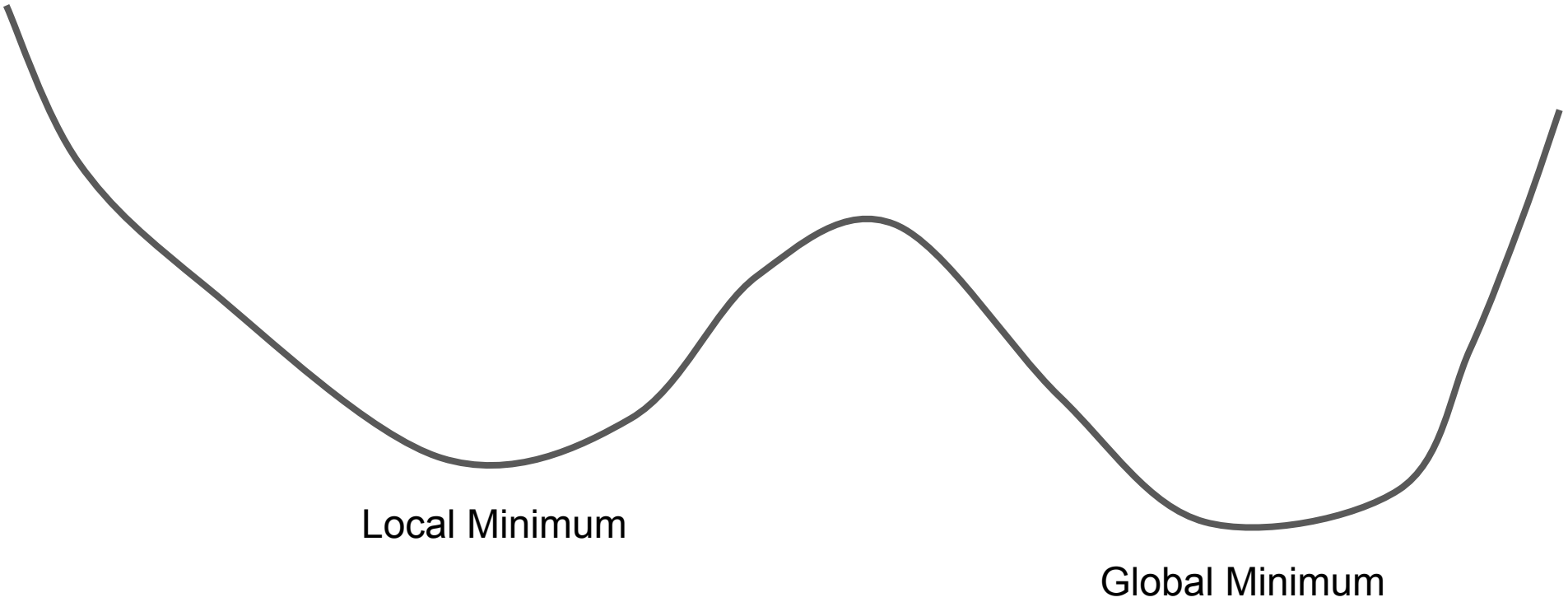


https://ruthwik.github.io/machinelearning/2018-01-15-gradient-descent/

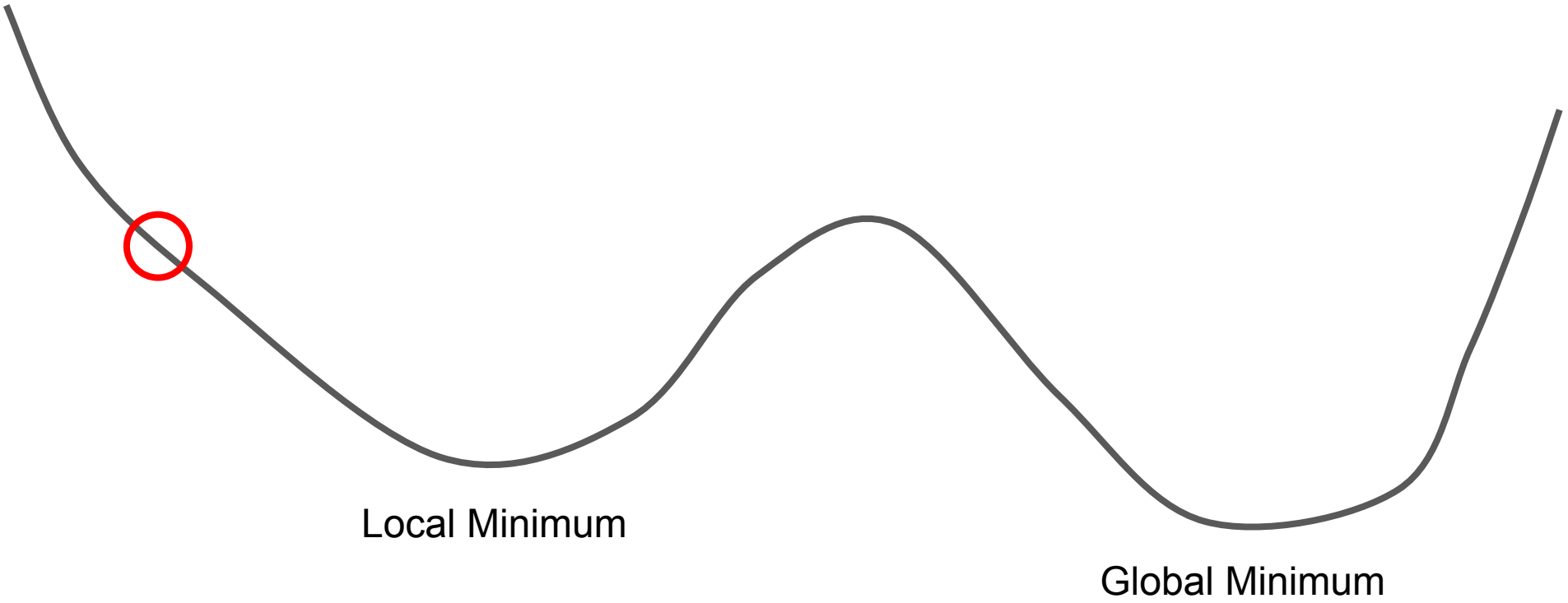
- How do you find local minima?
  - Make random guesses and iterate. New guesses are close to previous good guesses. Or do a more organized search pattern. Or imitate evolution.
  - Figure out which way is downhill and move a little bit in that direction (step size α (alpha) or ε (epsilon)) (This is gradient descent: update = - step_size * gradient)
    - Line search
  - When is a (or b) the best choice?
  - When are these methods guaranteed?



- 2. How do you find the (or a) global minimum?
  - Restart search in random places
  - Simulated annealing
  - When are these methods guaranteed?



1. How do you find local minima?
2. How do you find the (or a) global minimum?



Gradient Methods - Taking derivatives seriously

df(x)/dx = limit_as_e_goes_to_zero_of (f(x+e) - f(x))/e

Analytic functions: A Taylor series tells you what the function is nearby.

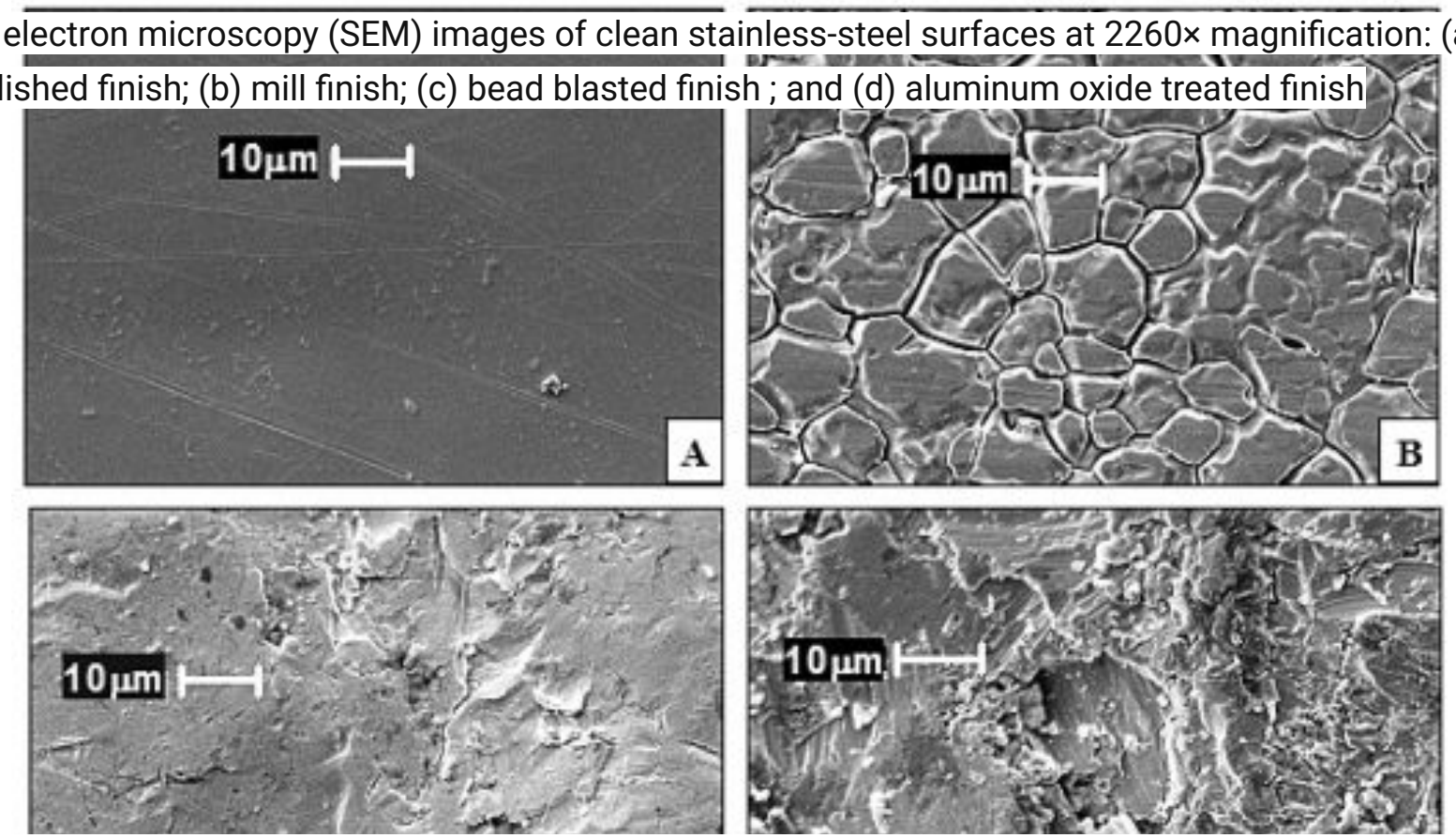
$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots$$

Numerical estimate of a derivative:

df(x)/dx = for_some_finite_e (f(x+e) - f(x-e))/(2*e) N^2 cost

You chose e to match the phenomena you are interested in. In time series (like stock prices), we might care about sample to sample variations (milliseconds or seconds), or what happens in a minute, hour, day, week, month, year, decade, or century.

Scanning electron microscopy (SEM) images of clean stainless-steel surfaces at 2260× magnification: (a) electropolished finish; (b) mill finish; (c) bead blasted finish ; and (d) aluminum oxide treated finish



In the real world derivatives taken in an infinitesimally small region are not very predictive

At least two points of view/emphases

Zac (Space): Models are known and accurate (model structures are correct and model parameters are fairly well known/estimated/learned); energy remains constant (intermittent actuation, no friction, damping, or perturbations). ➔ derivatives are available, accurate, and useful; gradient-based optimization, more emphasis on trajectories rather than policies; simulations should conserve energy and accurate simulation matters.

Chris (Practical Robotics): Models can be accurate, but sometimes are not; models are often “learned” using generic model structures (bias); models of the task or environment are generally not easily or cheaply available; energy fluctuates wildly (constant actuation, impact, perturbations, friction, damping, wind, temperature, humidity, …), some sensors are noisy. ➔ gradients are not necessarily reliable; more use of non-gradient-based optimization; more focused on dynamic programming and reinforcement learning, more interested in learning rather than computing policies; more interested in fast simulation than accurate simulation.

ML = The chain rule (for example, neural network training)

A good optimization update can be computed with a first order gradient step (-step_size*gradient) or a second order step (-inverse(Hessian)*gradient).

Non-gradient methods often numerically estimate derivatives and gradients implicitly, incurring an N^2 cost in terms of function evaluations.

Symbolic computation of derivatives and gradients using the chain rule greatly reduces computational cost. Given that one can take the derivative of a computer program symbolically, this is a huge advantage for gradient-based optimization methods.

Something to help you remember the chain rule