Scientific Computing

April 7, 2025

Announcements

→ Homework 5 due Fri Apr 18, 11:59pm

→ No class Fri Apr 18, Mon Apr 21 (Easter Break)

Today

→ Simulated Annealing

→ Intro to Neural Networks

Office Hours:

Mon + Fri

9:30am - 10:30am

Cudahy 307

* Spring Demo — continuous
  - staying within constraints by re-tweaking
  - playing with parameters

* Knapsack Demo — discrete

* Knapsack Demo

(1) Bad tweak function

(2) Constraint penalty

We will allow invalid solutions, but we'll penalize the score

$$score \rightarrow score \cdot \left(1 - u \cdot \frac{excess\ weight}{capacity}\right)$$

where u can be tuned statically or dynamically

% over capacity

The constraint penalty idea lets us use simulated annealing on problems with only constraints, like Sudoku.

Score = [# of row conflicts]
+ [# of column conflicts]
+ [# of square conflicts]

Goal: Minimize score, hope to get to 0

Tweaks?

Parallel Tempering

An interesting related idea:

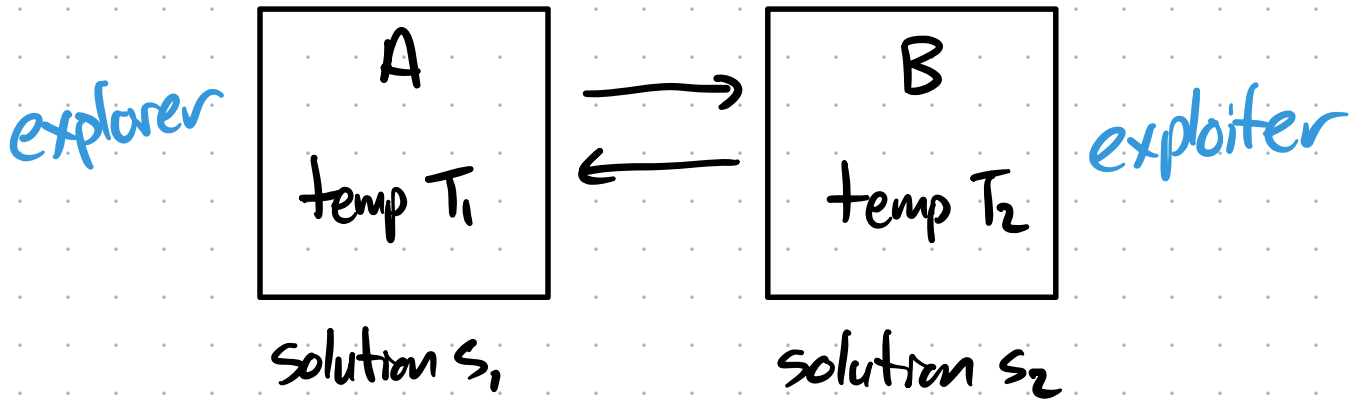
Instead of running one system that cools over time, run multiple systems each at constant, but different, temperatures that are allowed to swap solutions.

Intuition: Person A is very good at exploring

Person B is very good at exploiting

They both run for a while, until Person A says "Hey, I think I'm on a good hill, let's swap so you can exploit it."

$T_1 > T_2$



Should they swap solutions?

Let $E_i = \text{score}(s_i)$.

At any point in time, swap with probability

$$p = \min(1, e^{\Delta})$$

where

$$\Delta = (E_1 - E_2)\left(\frac{1}{T_2} - \frac{1}{T_1}\right)$$

> 0 when the explorer is winning

always > 0

$$\Delta = (E_1 - E_2)\left(\frac{1}{\tau_2} - \frac{1}{\tau_1}\right)$$

>0 when the explorer is winning

always >0

When the explorer is winning, we should definitely swap, so the exploiter can improve it.

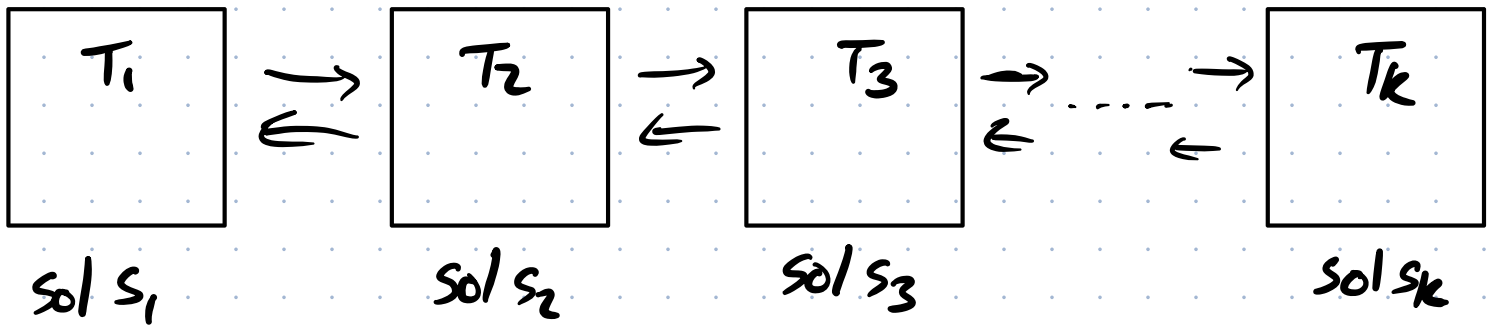
In this case, $\Delta > 0$ so $p = \min(1, e^{\Delta}) = 1$. ✓

Otherwise $E_1 - E_2 < 0$ and $p = e^{\frac{E_1 - E_2}{1/\tau_2 - 1/\tau_1}} < 1$.

The better the explorer is doing, the more likely they are to swap.

More generally:

k different explorers



$$\Delta_i = (E_i - E_{i+1})\left(\frac{1}{T_{i+1}} - \frac{1}{T_i}\right)$$

Swap s_i and s_{i+1} with prob $p_i = \min(1, e^{\Delta_i})$

Recap

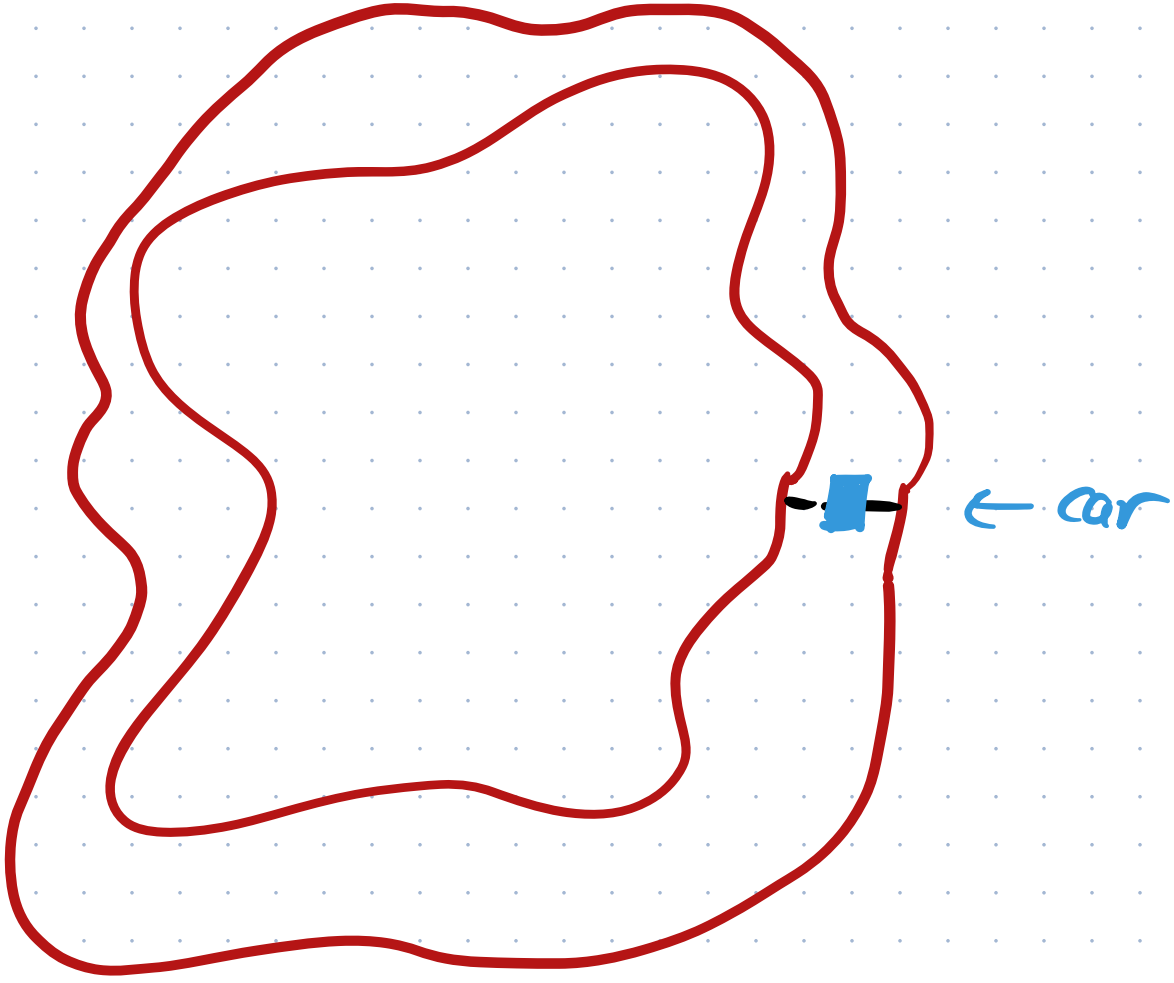
Want to use Hill Climbing or Simulated Annealing to solve a problem? You need:

* to understand the search space (or even need to rephrase the problem to have a search space)
* a scoring function that rates things in the search space as better or worse
* a "tweak" function that turns any solution into a very similar ("close") solution - and each solution should have many different tweaks possible
* a stopping condition

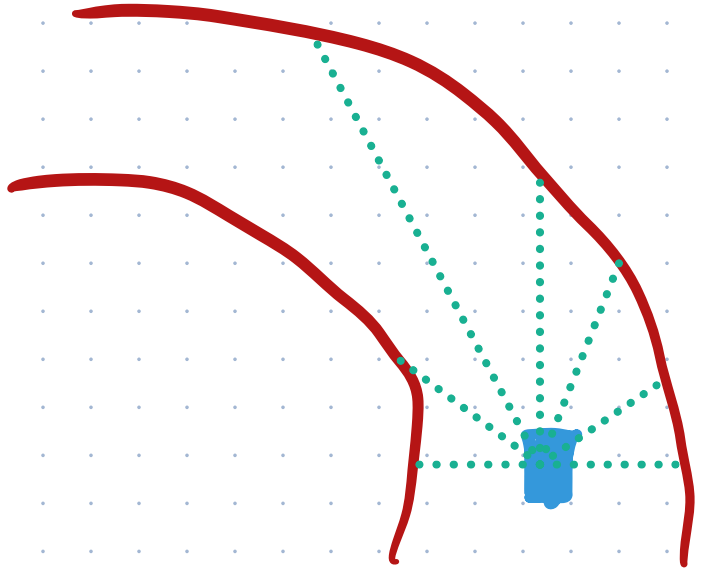
For SA: * an initial temp and a cooling schedule (how to cool and when to cool)

Crazy Example : Self-Driving Car

Problem: We want to teach a 2D car to drive around a track.



What data does the car have? Let's say it has 7 sensors that tell it the distance to the wall along that sensor direction.



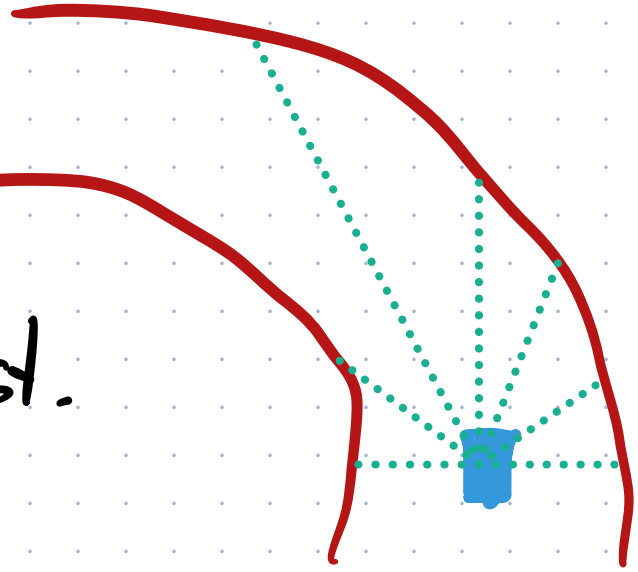
To drive, the car decides two things each step

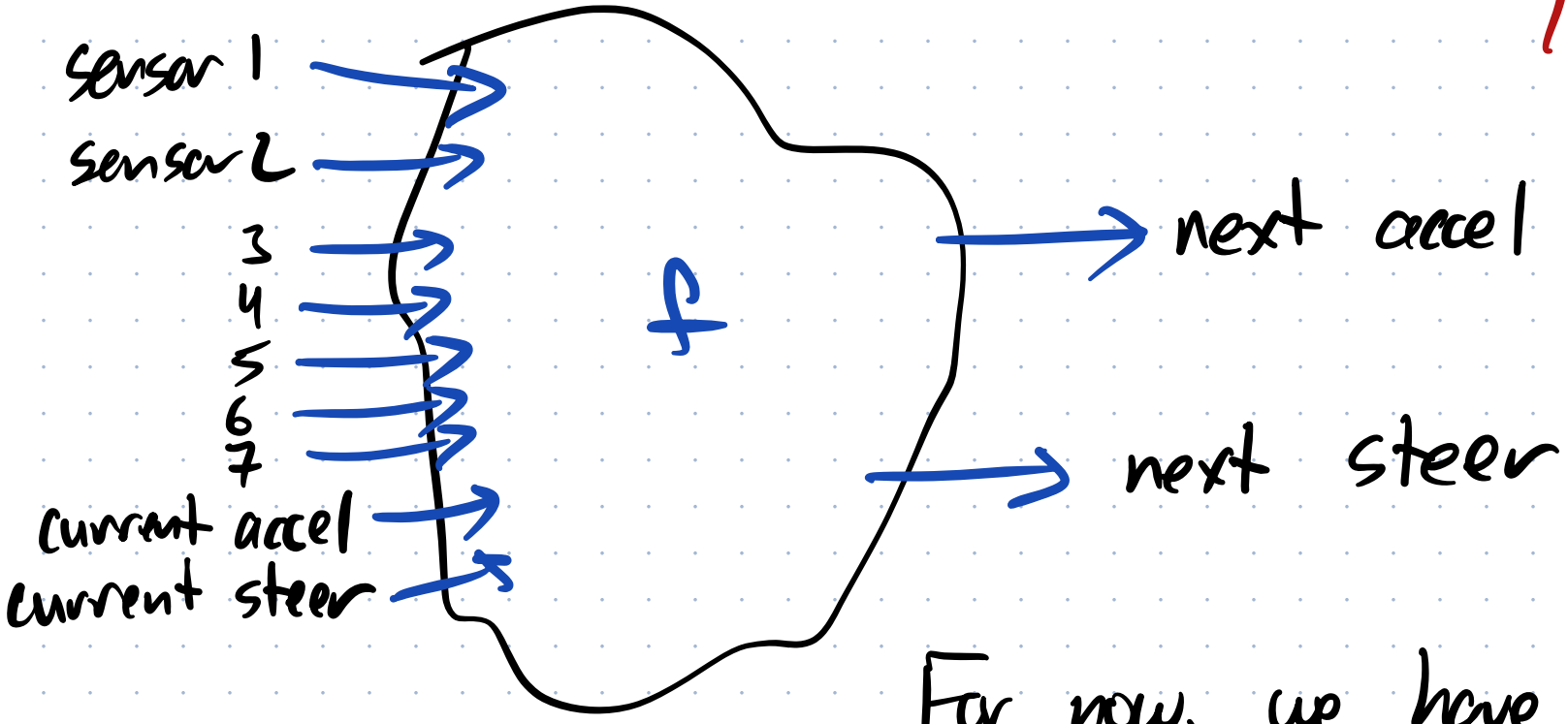
* acceleration (how much to change its speed)
* turning wheel (how much to change its steering angle)

[manual demo]

Conversion to an optimization problem:

Find the function f that makes the car drive the best.





For now, we have to pick a specific form for f.

$$f(s_1, s_2, \ldots, s_7, a_c, s_c) = (a_n, s_n)$$

One possibility: assume f is a <u>linear</u> function.

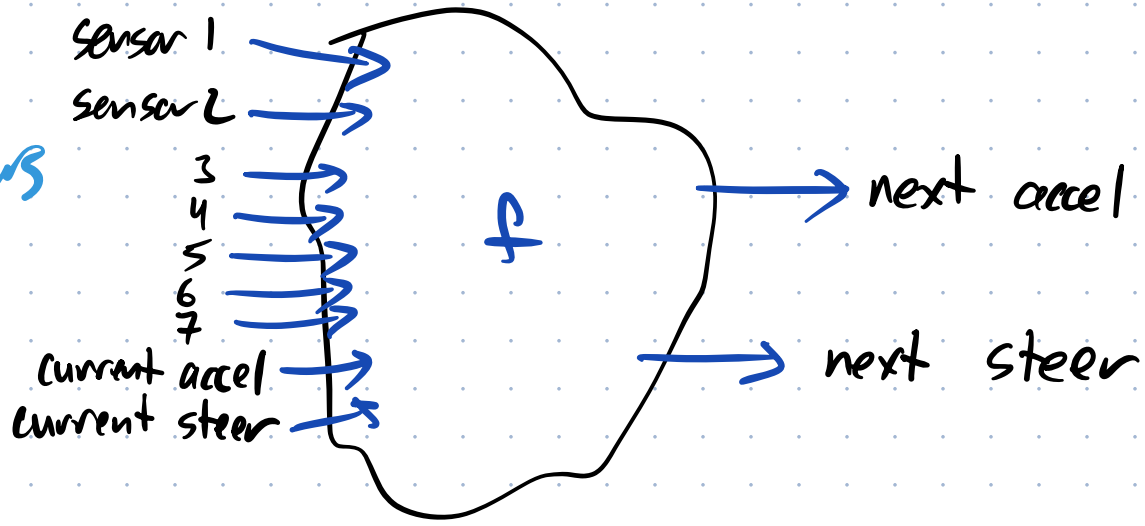
$$a_n = (?)s_1 + (?)s_2 + (?)s_3 + \cdots + (?)s_7 + (?)a_c + (?)s_c + (?)$$

$$s_n = (?)s_1 + (?)s_2 + (?)s_3 + \cdots + (?)s_7 + (?)a_c + (?)s_c + (?)$$

20 unknown coefficients that we get to choose.

Search space:

20-dimensional vectors of decimals, $\mathbb{R}^{20}$



Search space:

20-dimensional vectors of decimals, $\mathbb{R}^{20}$

tweak: change all (or some?) of the coefficients by a little bit (how much?)

Still need a scoring function! What do you want the car to do?

* Distance travelled without crashing?
* Incorporate average velocity?

Hill Climbing

Start with 20 random coefficients for f.

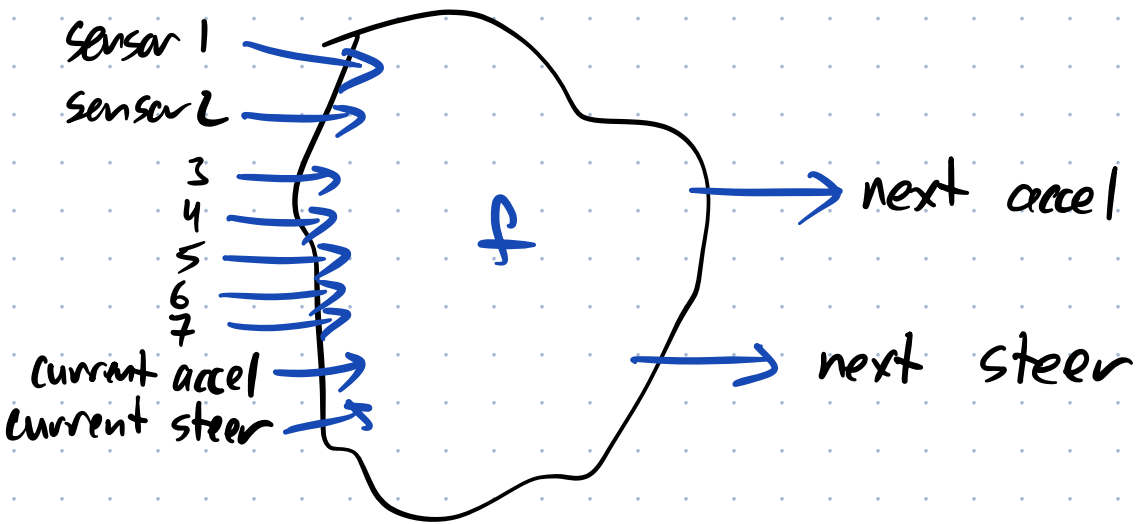
Put the car on a track and run it.

Each step, f decides how the car moves.

If the car saves better, that's your new "best". Tweak it and try again.

If the tweak is worse, throw it out.

Run for a long time!



Limitations:

* Slow scoring function!
* Maybe assuming self-driving is linear is very constraining - more complicated functions have more freedom to treat sensor input differently
* One car at a time doesn't give you a lot of diversity. Should try 100-trials steepest ascent or other metaheuristics like genetic algorithms

[demo]

Introduction to Neural Networks

Two separate questions:

(1) What is a neural network?
What does it do?

(2) How do we produce a good one for our task?

This lecture addresses (1).

Future lectures address (2).

Sources:

* Book: "Neural Networks From Scratch in Python"
* Lots of online resources and Youtube videos that I'll share as we go.

Goals:

* Understand the foundations of neural networks.
* See examples of training NNs to accomplish a task
  ↳ "training" = "find a good NN"
* Write our own Python code to create and train NNs (not using Machine Learning libraries)

Maybe:

* Learn about Python ML libraries (pytorch, tensorflow)
* Learn about specialized kinds of NNs for certain tasks.

Strong Analogy: Linear Regression

What is it?

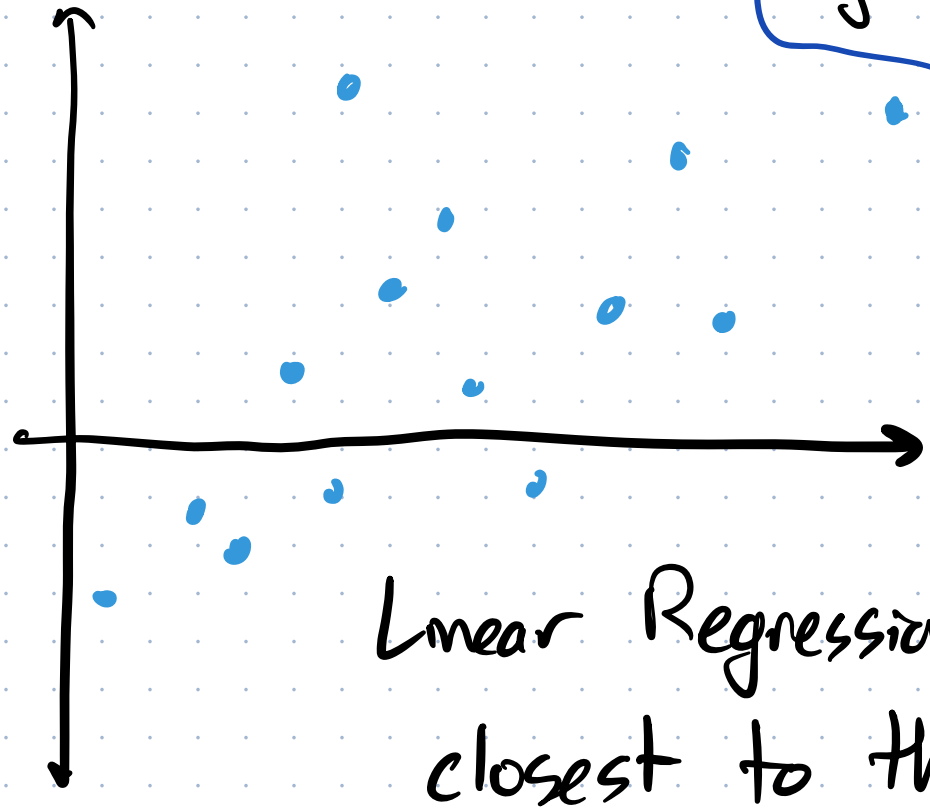
Simplest version: you have a bunch of (x,y) points and you want to find the line the best approximates them

Usually, the (x,y) points represent inputs (x) and outputs (y) of some unknown function.

x: time since Jan 1 this year
y: temperature on my outdoor thermometer

Strong Analogy: Linear Regression

x: time since Jan 1 this year
y: temperature on my outdoor thermometer



We only have sporadic readings.

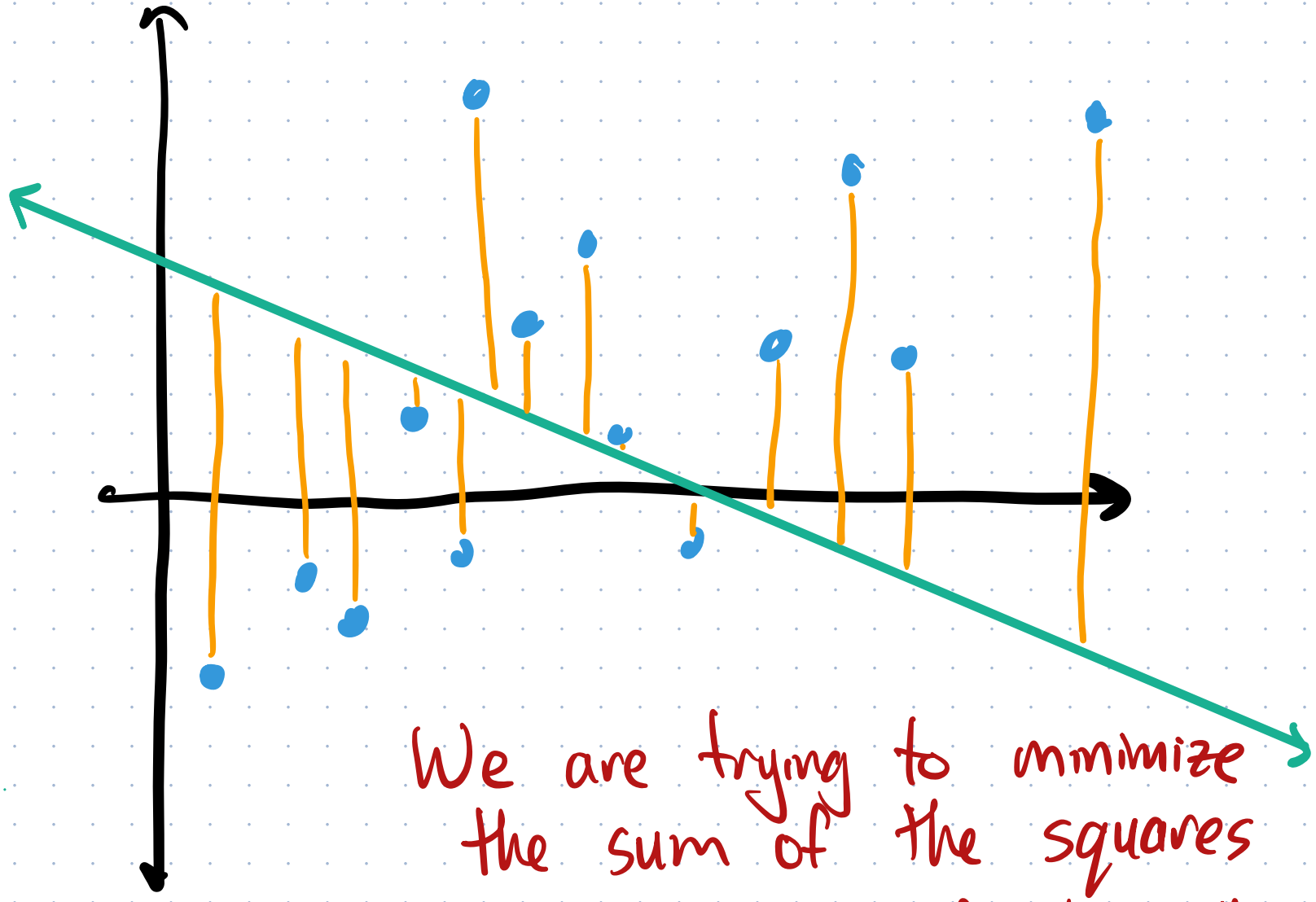
Linear Regression asks "what line is closest to these points?"

"Closest" means minimizing the sum of

$$([\text{actual y value}] - [\text{predicted y value}])^2$$

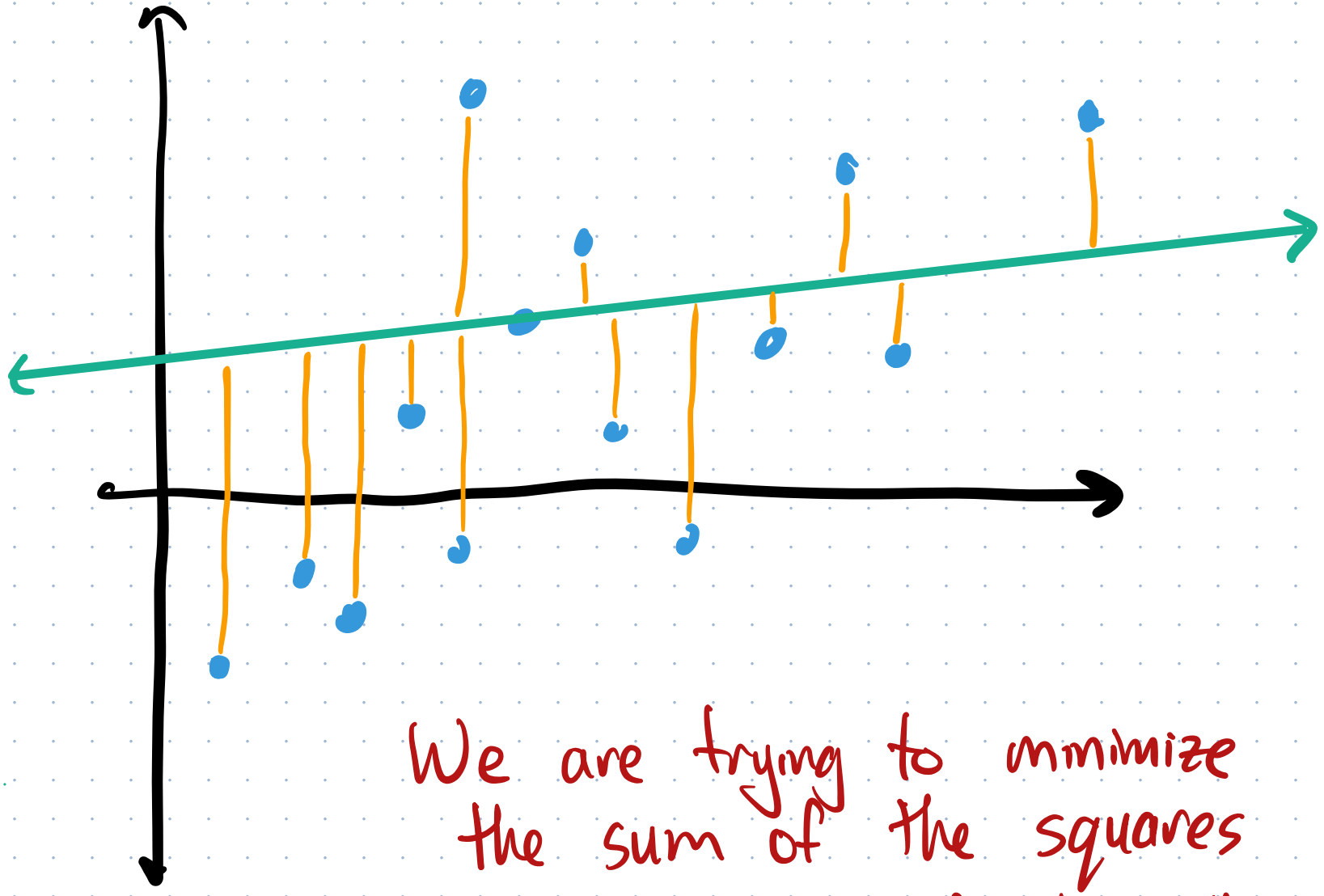
over all known points.

Strong Analogy: Linear Regression



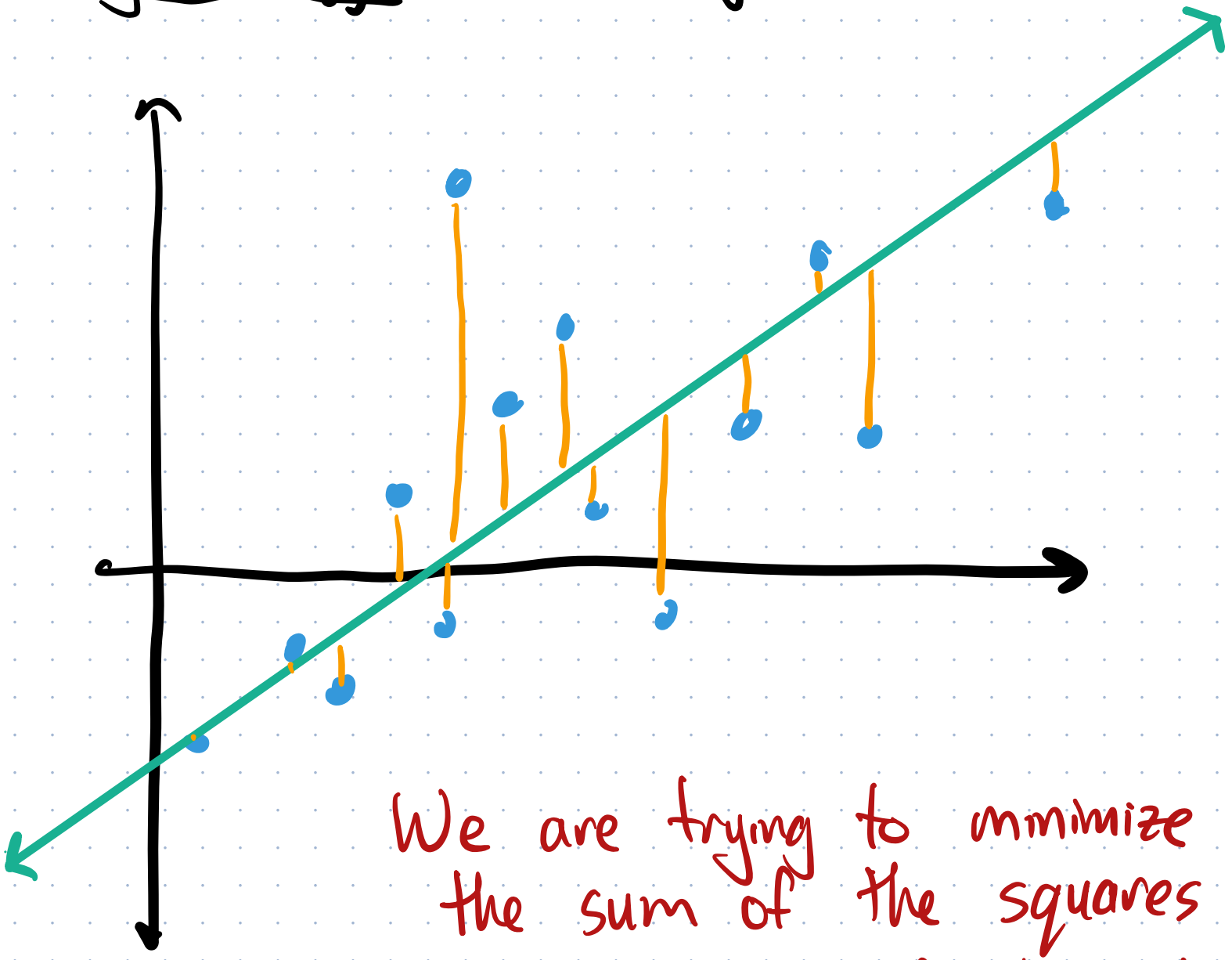
We are trying to minimize the sum of the squares of the lengths of the yellow lines

Strong Analogy: Linear Regression



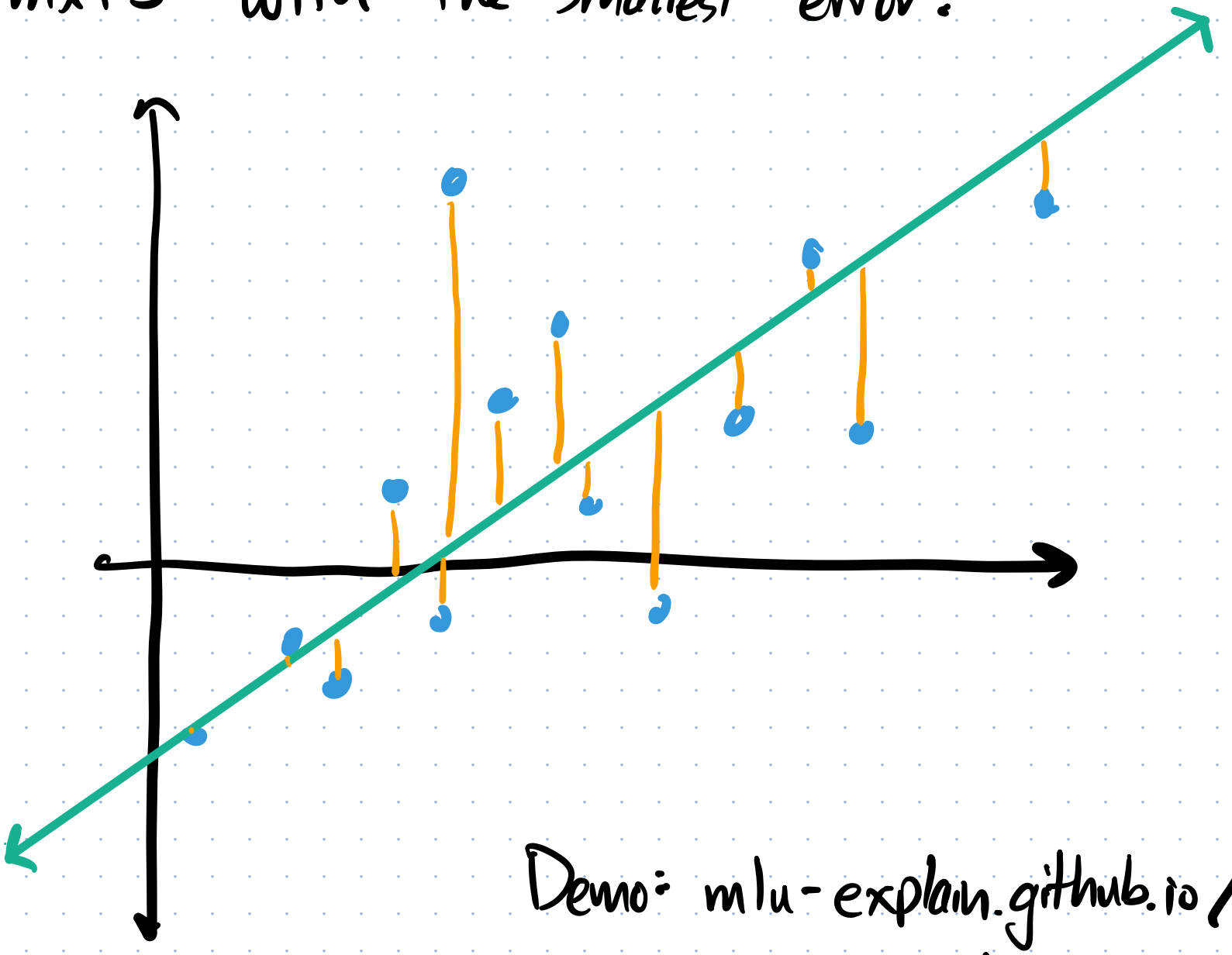
We are trying to minimize the sum of the squares of the lengths of the yellow lines

Strong Analogy: Linear Regression



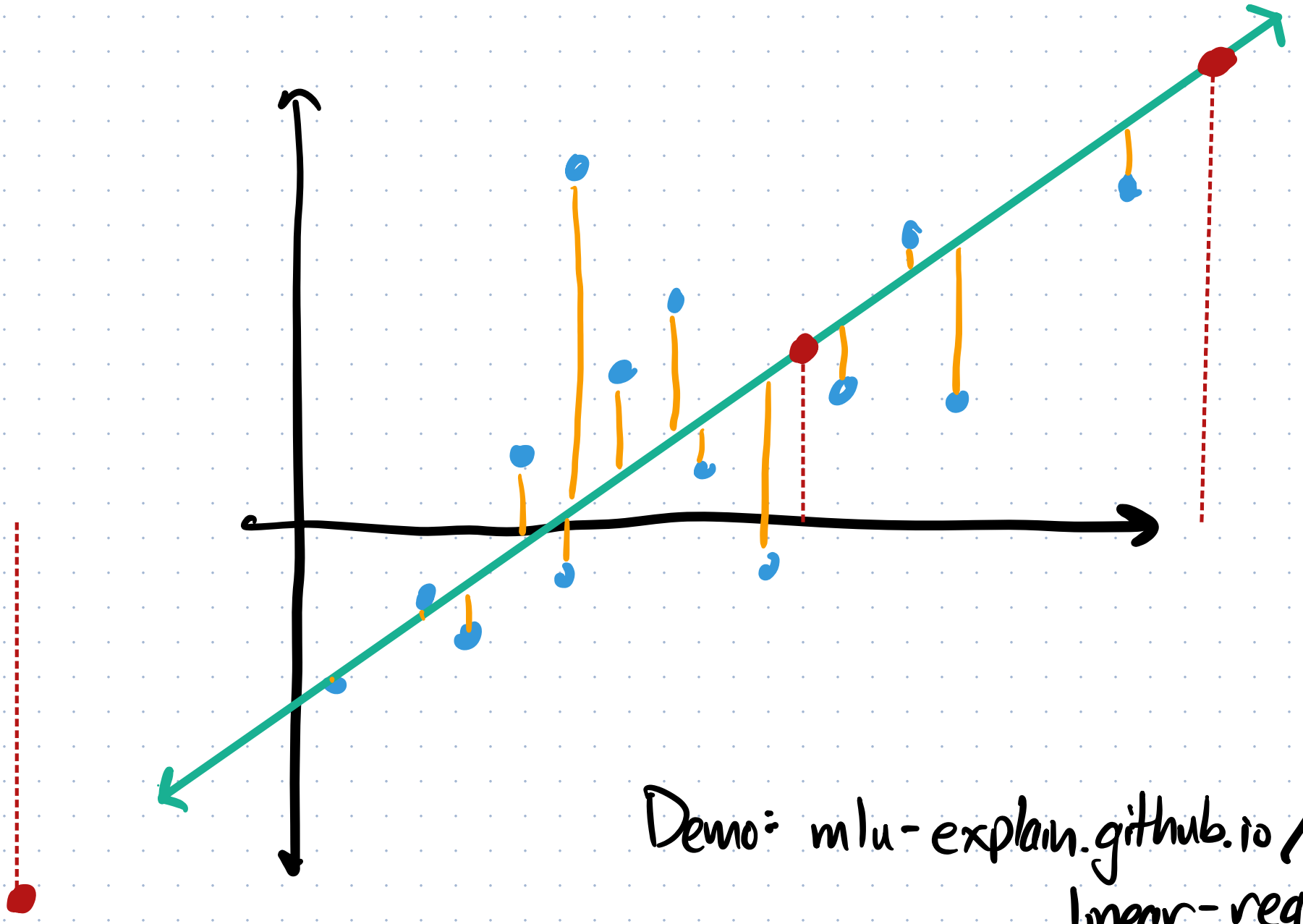
We are trying to minimize the sum of the squares of the lengths of the yellow lines

Problem: What values of m and b make the line $y = mx + b$ with the smallest error?



Demo: mlu-explain.github.io/linear-regression

Purpose: Estmate the output at new inputs.



Demo: mlu-explain.github.io/linear-regression