Module 5 – Gradient Descent

DSC 40A, Summer 2023

Agenda

- Brief recap of Module 4.
- Gradient descent fundamentals.

Empirical risk minimization

The recipe

Suppose we're given a dataset, $y_1, y_2, ..., y_n$ and want to determine the best future prediction h^*.

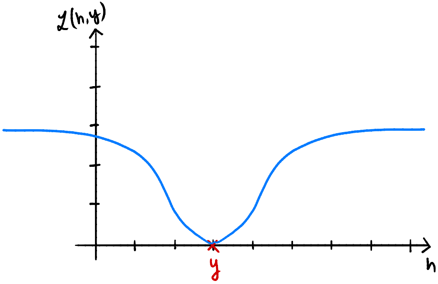
1. Choose a loss function $L(h, y)$ that measures how far our prediction h is from the "right answer" y.
   - Absolute loss, $L_{abs}(h, y) = |y - h|$.
   - Squared loss, $L_{sq}(h, y) = (y - h)^2$.
2. Find h^* by minimizing the average of our chosen loss function over the entire dataset.
   - "Empirical risk" is just another name for average loss.

$$R(h) = \frac{1}{n} \sum_{i=1}^{n} L(h, y)$$

A very insensitive loss

- Last time, we introduced a new loss function, L_{ucsd}, with the property that it (roughly) penalizes all bad predictions the same.
  - A prediction that is off by 50 has approximately the same loss as a prediction that is of by 500.

  - The effect: L_{ucsd} is not as sensitive to outliers.



A very insensitive loss

- The formula for L_{ucsd} is as follows (no need to memorize):

$$L_{ucsd}(h, y) = 1 - e^{-(y-h)^2/\sigma^2}$$

  - The shape (and formula) come from an upside-down bell curve.

- L_{ucsd} contains a **scale parameter**, σ. ← controls width
  - Nothing to do with variance or standard deviation.
  - Accounts for the fact that different datasets have different thresholds for what counts as an outlier.
  - Like a knob that you get to turn – the larger σ is, the more sensitive L_{ucsd} is to outliers (and the more smooth R_{ucsd} is).

Minimizing R_{ucsd}

- The corresponding empirical risk, R_{ucsd}, is

$$R_{ucsd}(h) = \frac{1}{n} \sum_{i=1}^{n} \left[1 - e^{-(y_i - h)^2 / \sigma^2}\right]$$

- R_{ucsd} is **differentiable**.
- To minimize: take derivative, set to zero, solve.

Step 2: Setting to zero and solving

- We found:

$$\frac{d}{dh} R_{ucsd}(h) = \frac{2}{n\sigma^2} \sum_{i=1}^{n} (h - y_i) \cdot e^{-(h-y_i)^2/\sigma^2}$$

- Now we just set to zero and solve for h:

$$0 = \frac{2}{n\sigma^2} \sum_{i=1}^{n} (h - y_i) \cdot e^{-(h-y_i)^2/\sigma^2}$$

- We **can** calculate derivative, but we **can't** solve for h; we're stuck again.

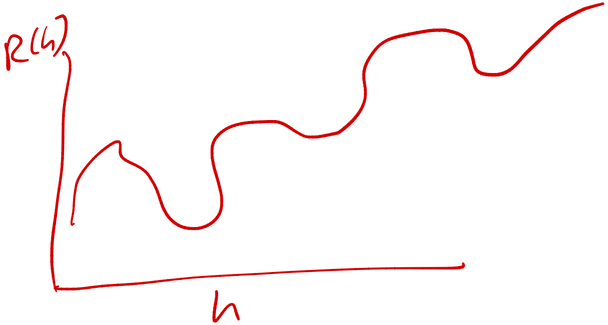
Gradient descent fundamentals

other reasons to use

- computational complexity

The general problem

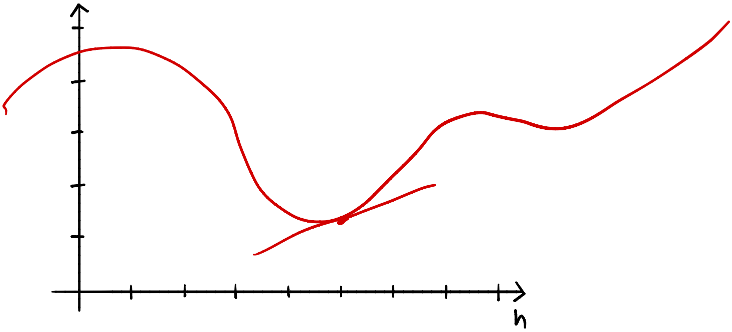
- **Given:** a differentiable function $R(h)$.
- **Goal:** find the input h^* that minimizes $R(h)$.



Meaning of the derivative

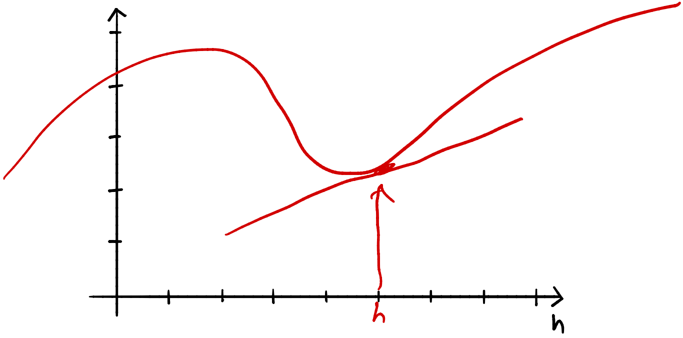
- We're trying to minimize a **differentiable** function $R(h)$. Is calculating the derivative helpful?

- $\frac{dR}{dh}(h)$ is a function; it gives the **slope** at h.



Key idea behind gradient descent

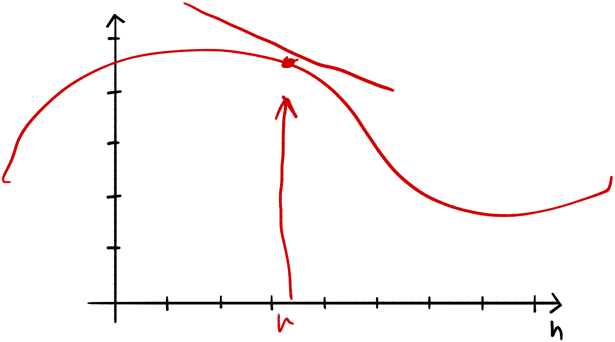
- If the slope of R at h is **positive** then moving to the **left** decreases the value of R.
- i.e., we should **decrease** h.



Key idea behind gradient descent

- If the slope of R at h is **negative** then moving to the **right** decreases the value of R.

- i.e., we should **increase** h.



Key idea behind gradient descent

- Pick a starting place, h_0. Where do we go next?

  initial guess

- Slope at h_0 negative? Then increase h_0.
- Slope at h_0 positive? Then decrease h_0.

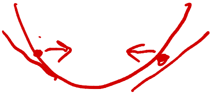
Key idea behind gradient descent

- Pick a starting place, h_0. Where do we go next?
- Slope at h_0 negative? Then increase h_0.
- Slope at h_0 positive? Then decrease h_0.
- Something like this will work:

$$h_1 = h_0 - \frac{dR}{dh}(h_0)$$

init

slope of old prediction



Gradient Descent

- Pick α to be a positive number. It is the **learning rate**, also known as the **step size**.

- Pick a starting prediction, h_0.

- On step i, perform update $h_i = h_{i-1} - \alpha \cdot \frac{dR}{dh}(h_{i-1})$

many approaches:
- momentum
- adaptive

- Repeat until convergence (when h doesn't change much). Alternative criteria: magnitude of gradient is close to zero; validation error stops improving.

bigger α yields bigger steps

Gradient Descent

→ type: callable

```
def gradient_descent(derivative, h, alpha, tol=1e-12):
    """Minimize using gradient descent."""
    while True:
        h_next = h - alpha * derivative(h)
        if abs(h_next - h) < tol:
            break
        h = h_next
    return h
```

Note: it's called gradient descent because the gradient is the generalization of the derivative for multivariable functions.

Example: Minimizing mean squared error

- Recall the mean squared error and its derivative:

$$R_{sq}(h) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h)^2 \qquad \frac{dR_{sq}}{dh}(h) = \frac{2}{n}\sum_{i=1}^{n}(h - y_i)$$

Discussion Question

Consider the dataset $-4, -2, 2, 4$. Pick $h_0 = 4$ and $\alpha = \frac{1}{4}$. Find h_1.

a) -1
b) 0
c) 1
d) 2

Solution

$$R_{sq}(h) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h)^2 \qquad \frac{dR_{sq}}{dh}(h) = \frac{2}{n}\sum_{i=1}^{n}(h - y_i)$$

Consider the dataset $-4, -2, 2, 4$. Pick $h_0 = 4$ and $\alpha = \frac{1}{4}$. Find h_1.

$h_i = h_{i-1} - \alpha \frac{dR}{dh}(h_{i-1})$ $n = 4$

$\frac{2}{4}\left((4+4) + (4+2) + (4-2) + (4-4)\right)$

$\frac{2}{4}(8 + 6 + 2 + 0)$

$\frac{2}{4}(16)$

$\frac{32}{4} = 8$

$4 - \frac{1}{4}\cdot 8 = 2$

Summary

- Gradient descent is a general tool used to minimize differentiable functions.
  - We will usually use it to minimize empirical risk, but it can minimize other functions, too.

- Gradient descent progressively updates our guess for h^* according to the update rule

$$h_i = h_{i-1} - \alpha \cdot \left(\frac{dR}{dh}(h_{i-1})\right).$$

- **Next Time:** We'll demonstrate gradient descent in a Jupyter notebook. We'll learn when this procedure works well and when it doesn't.