Machine Learning and the Big Data Regime...

goal: find a d-dim parameter vector which minimizes the loss on n training examples.

- have n training examples $(x_1, y_1), \ldots (x_n, y_n)$
- have parametric a classifier $h(x, w)$, where w is d dimensional.

$$\min \sum_i \mathrm{loss}(h(x_i, w), y_i)$$

- "Big Data Regime": How do you optimize this when n and d are large? memory? parallelization?

Can we obtain linear time algorithms?

Part 1: Least Squares

$$\min_{w} \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

How much computation time is required to to get ϵ accuracy?

- n points, d dimensions.
- "Big Data Regime": How do you optimize this when n and d are large?

Aside: think of x as a large feature representation.

Review: Direct Solution

$$\min_w \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

- solution:

  $$w = (X^\top X + \lambda I)^{-1} X^\top Y$$

  where X be the $n \times d$ matrix whose rows are x_i, and Y is an n-dim vector.
- time complexity: $O(nd^2)$ and memory $O(d^2)$

Not feasible due to both time and memory.

Review: Gradient Descent (and Conjugate GD)

$$\min_w \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

- n points, d dimensions,
- $\lambda_{\max}, \lambda_{\min}$ are eigs. of "design/data matrix"
- Computation time to get ϵ accuracy:
  - Gradient Descent (GD):

$$\frac{\lambda_{\max}}{\lambda_{\min}} nd \log 1/\epsilon$$

  - Conjugate Gradient Descent:

$$\sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} nd \log 1/\epsilon$$

- memory: $O(d)$

Better runtime and memory, but still costly.