Gradient descent

Fraida Fund

Contents

In this lecture

- Runtime of OLS solution for multiple/LBF regression
- Solution using gradient descent

Solution using gradient descent

Why gradient descent?

We had

$$\mathbf{w}^* = \left(\Phi^T \Phi\right)^{-1} \Phi^T \mathbf{y}$$

where Φ is an $n \times d$ matrix. If $n \geq d$ then it is (usually) full rank and a unique solution exists.

What if n, d are large?

Runtime of a “naive” solution using “standard” matrix multiplication:

- $O(dn^2)$ to multiply $\Phi^T \Phi$
- $O(dn)$ to muplity $\Phi^T y$
- $O(d^3)$ to compute the inverse of $\Phi^T \Phi$

Since n is generally much larger than d, the first term dominates and the runtime is $O(dn^2)$. Can we do better?

(Note: in practice, we would not necessarily use the “naive” way.)

Gradients and optimization

Gradient has *two* important properties for optimization:

At a minima (or maxima, or saddle point),

$$\nabla L(\mathbf{w}) = 0$$

At other points, $\nabla L(\mathbf{w})$ points towards direction of maximum (infinitesimal) *increase*.

Gradient descent idea

To move towards minimum of a (smooth, convex) function, use first order approximation:

Start from some initial point, then iteratively

- compute gradient at current point, and
- add some fraction of the negative gradient to the current point

Visual example: least square solution 3D plot

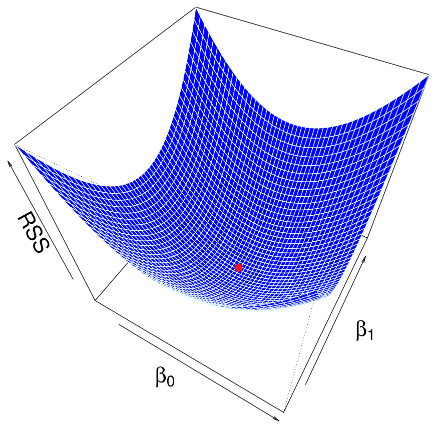

Figure 1: Regression parameters - 3D plot.

Standard ("batch") gradient descent

For each step t along the error curve:

$$\begin{aligned} \mathbf{w}^{t+1} &= \mathbf{w}^t - \alpha \nabla L(\mathbf{w}^t) \\ &= \mathbf{w}^t - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla L_i(\mathbf{w}^t, \mathbf{x}_i, y_i) \end{aligned}$$

Repeat until stopping criterion is met.

To update $\mathbf{w}$, must compute n loss functions and gradients - each iteration is $O(nd)$. We need multiple iterations, but as long as we need fewer than n iterations, it's more efficient than the previous approach.

However, if n is large, it may still be expensive!

Stochastic gradient descent

Idea:

At each step, compute estimate of gradient using only one randomly selected sample, and move in the direction it indicates.

Many of the steps will be in the wrong direction, but progress towards minimum occurs *on average*, as long as the steps are small.

Each iteration is now only $O(d)$, but we may need more iterations than for gradient descent. However, in many cases we still come out ahead (especially if n is large!).

See supplementary notes for an analysis of the number of iterations needed.

Mini-batch (also "stochastic") gradient descent (1)

Idea:

At each step, select a small subset of training data ("mini-batch"), and evaluate gradient on that mini-batch.

Then move in the direction it indicates.

Mini-batch (also "stochastic") gradient descent (2)

For each step t along the error curve:

- Select random mini-batch $I_t \subset 1, \ldots, n$
- Compute gradient approximation:

$$g^t = \frac{1}{|I_t|} \sum_{i \in I_t} \nabla L(\mathbf{x}_i, y_i, \mathbf{w}^t)$$

- Update parameters: $\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha^t g^t$

Next: more on linear regression