Lecture 19 - 17/11/24

Gradient Descent

- It is an optimization algorithm
- Optimization = Search, finding best solution from all the feasible ones!
- Minimizing or Maximizing an objective function wrt some vars.

- Analytical methods → finding a closed-form solution!
  - Gives exact solutions
  - Time-consuming
  - Closed-form solutions don't always exist!
- Numerical Methods:
  - Approx. solutions with allowable tolerance
  - Might be faster!
  - Possible in most cases
  - Better suited where there are a lot of features.

Gradient Descent

- It is a general opt. algo. to find minimum.

Resume:

- Optimization method
- General Purpose
- Iterative
- Uses first derivatives

Gradient Vector: Vector of partial derivatives of a function wrt its params:

$$\text{Gradient} \rightarrow \nabla f = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_m} \end{bmatrix}$$

It points in the direction of greatest ascent of f.

In order to descent the ∇ we use negative of ∇f

3 Steps:

- Set a learning rate: α
- Randomly initialize parameters
- Repeat until convergence

substitute with value θ^t

$$\theta^{(t+1)} = \theta^{(t)} - \boxed{\alpha} \boxed{\frac{\partial f}{\partial \theta}} \Bigg|_{\theta^t}$$

learning rate

partial derivative wrt to θ (var i.e. x)

Note! The − is due to the fact that we want opposite of gradient!!!

Example!

$$f(\theta) = (\theta - 5)^2$$

$$\boxed{\frac{\partial f}{\partial \theta}} = 2(\theta - 5) \rightarrow$$

$\alpha = 0.1$ chosen by us!

$\theta^{(0)} = 1$ randomly chosen initial val

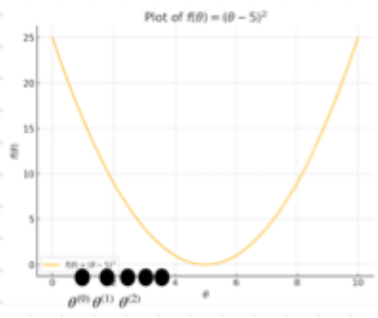
$$\theta^{(1)} = \boxed{1} - \boxed{0.1} \cdot \boxed{2(\boxed{1} - 5)} = 1 + 0.8 = \boxed{1.8}$$

$$\theta^{(2)} = 1.8 - 0.1 \cdot 2(1.8 - 5) = 1.8 + 0.64 = \boxed{2.44}$$

$\vdots$

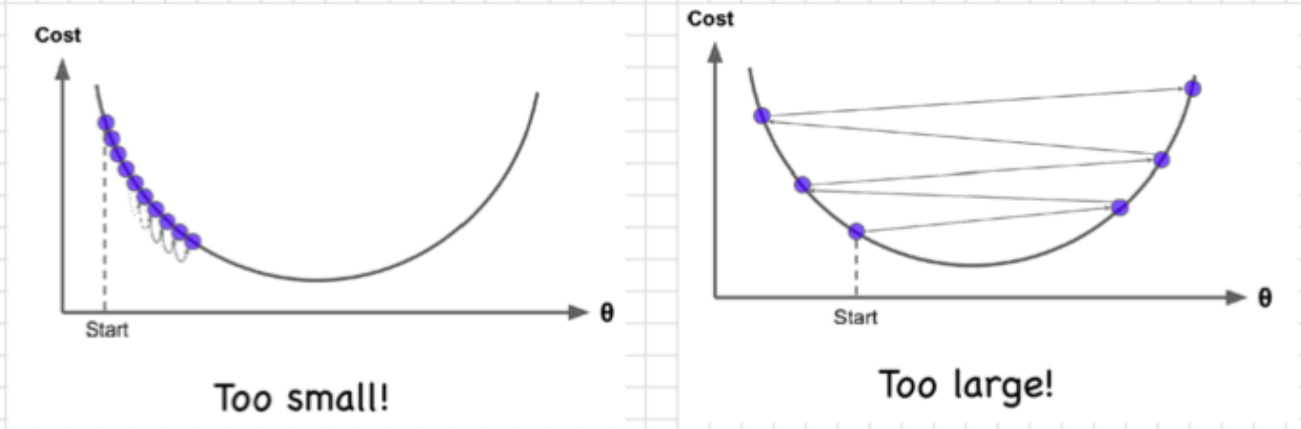
When to STOP? When we reach Convergence!!

(non mandatory gradient = 0)

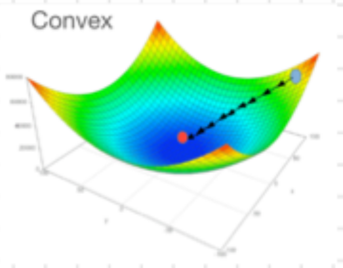


Learning Rate

- GD is sensitive to the learning rate!!



- The GD finds the global optimum if the function is:
  - differentiable
  - convex



Multivariated function

* In the example above we saw an application on just one variable θ
* The gradient = ∇ of a multivariate f is:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \cdots \\ \frac{\partial f}{\partial \theta_m} \end{bmatrix} \rightarrow \text{vector of its partial derivatives}$$

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla f \Big|_{\theta^t}$$

Train models

* Use GD to train ML models.
* Find optimal parameters of the model.

$f(\theta)$ = cost/loss function.

θ = vector of model's parameters.

- differentiable
- convex

Linear Reg. example

$$\boxed{\hat{y} = \beta_0 + \beta_1 x} \qquad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \rightarrow \text{Cost function}$$

$$\min_{\beta}[J(\beta)] \rightarrow \text{optimization objective!}$$

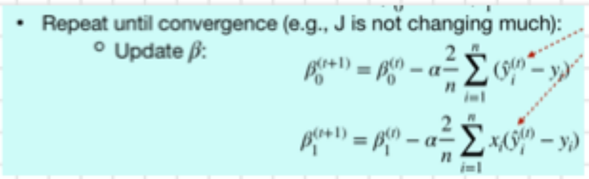
Example for Linear Regression:

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\boxed{\hat{y}_i} - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}[(\beta_0 + \beta_1 x_i) - y_i]^2$$

$$\frac{\partial J}{\partial \beta_0} = \frac{1}{n}\sum_{i=1}^{n} 2(\beta_0 + \beta_1 x_i - y_i) \cancel{+0} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$$

$$\frac{\partial J}{\partial \beta_1} = \frac{1}{n}\sum_{i=1}^{n} 2(\beta_0 + \beta_1 x_i - y_i) + x_i = \frac{2}{n}\sum_{i=1}^{n} x_i(\hat{y}_i - y_i)$$

$\alpha = 0.01 \quad \beta_0^{(0)} = 0, \ \beta_1^{(0)} = 0$



- Repeat until convergence (e.g., J is not changing much):
  - Update β:

$$\beta_0^{(t+1)} = \beta_0^{(t)} - \alpha\frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i^{(t)} - y_i)$$

$$\beta_1^{(t+1)} = \beta_1^{(t)} - \alpha\frac{2}{n}\sum_{i=1}^{n}x_i(\hat{y}_i^{(t)} - y_i)$$

Variations of Gradient Descent

Batch

* All training data is used to compute gradient:
  - Gradient of cost function
  - Used in every step
* Very slow in training
* Can cause memory problem
* Directly move towards optimum
* Works well only for convex

$$\beta^{(t+1)} = \beta^{(t)} - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla L_{\beta(t)}(\hat{y}_i^{(t)}, y_i)$$

↳ n = # of all points in DF.

Stochastic

* Considers just one datapoint at a time
* Repeat until convergence:
  * Repeat for a given # of iter:
    - Take a random sample
    - Feed it to formula to update params.
* By convention repeat n = size of DF
* Much faster
* Escapes Local Minima
* May never reach min

$$\beta^{(t+1)} = \beta^{(t)} - \alpha \nabla L_{\beta(t)}(\hat{y}_i^{(t)}, y_i)$$

only 1 point

Mini-Batch

* Computes gradients on small random sets
* May escape local minima
* Less fluctuation

$$\beta^{(t+1)} = \beta^{(t)} - \alpha \frac{1}{n} \sum_{i \in S} \nabla L_{\beta(t)}(\hat{y}_i^{(t)}, y_i)$$

S is a random set of samples