Parameter Learning

Gradient Descent

- It is used for minimizing cost function.
- It is a more general algorithm and is used not only in linear regression but at all over the different places in Machine Learning.
- Eg:- Have some function $J(\theta_0, \theta_1)$
  Want $\min\limits_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

  Outline →
  - Start with some θ_0, θ_1 (random).
  - Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum (local).

- Gradient Descent Algorithm :-

  repeat until convergence {

  $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j=0 \ \& \ j=1)$$

  }

(Next page)

Also update θ_0 and θ_1 simultaneously :—

Simultaneous Update :—

$$\boxed{\begin{aligned} &temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ &temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ &\theta_0 := temp0 \\ &\theta_1 := temp1 \end{aligned}}$$

This is correct method.

$$\boxed{\begin{aligned} &temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ &\theta_0 := temp0 \\ &temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0 - \theta_1) \\ &\theta_1 := temp1 \end{aligned}}$$

This is incorrect method, as you can see it uses updated θ_0 for cost function for getting temp1. This method may also work fine but it should be avoided as this some other algorithm, not gradient descent. It can also cause some weird behaviour.

- α is learning rate. It ~~tells~~ determines how much the θ_0 or θ_1 or θ_n should be changed.

- $:=$ is assignment operator.

- Above algorithm (gradient descent) can be applied on cost function with any (n) no. of parameters by slighly modifying above formulae.

- $\frac{\partial}{\partial \theta_j}$ ~~to~~ is symbol of partial derivative with respect ~~to~~ to θ_j.

- $j = 0, 1 \ldots n$. • j represents the feature index number.

- If α is too small, gradient descent can be ~~sl~~ slow.

- If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

- If parameter θ_1 is at local minimum already, then gradient descent will leave it unchanged. Because we will have 0 slope, so ~~to~~ derivative will be ~~also~~ 0, hence parameter will be unchanged.

- Gradient descent converge to a local ~~min~~ minimum even with the learning rate α fixed.

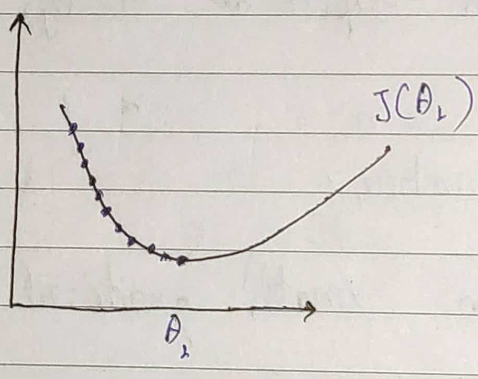
Because,

As we approach a local minimum, gradient descent will automatically take smaller steps.

So, no need to decrease α over time.

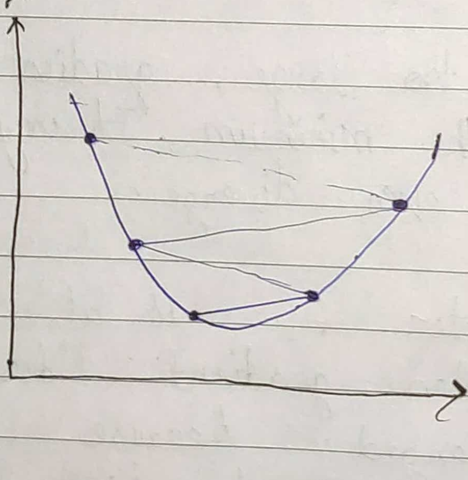
Examples :-

α too small :-



Small steps, slow gradient descent.

α too large :-



Fails to converge or even diverge

Applying Gradient Descent to Our Cost Function:-

Gradient Descent Algorithm

repeat until convergence {

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i\right)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i\right)^2$$

$$\Rightarrow \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x_i - y_i\right)^2$$

Now,

For :-

$\theta_0 \rightarrow j = 0$: $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i\right)$

$\theta_1 \rightarrow j = 1$: $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i\right) \cdot x_i$

So, our case:-

Gradient Descent:-

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i \right)}_{\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)}$$

$$\theta_1 := \theta_1 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i \right) \cdot x_i}_{\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)}$$

}

Also ~~upda~~ update θ_0 & θ_1 simultaneously.

Note:- Cost function for linear regression is always going to be a bow-shaped function.

Technical term for this is convex function.

Convex function means a bowl-shaped function, so this function doesn't have any local optima except for one global optimum.

Note :- Above implementation is also called Batch Gradient Descent.

It basically means that it uses all training examples in each step.

Other implementations of gradient descent which use subsets of data are also possible.