Linear Regression

(Gradient Descent Optimization)

Dr. JASMEET SINGH

ASSISTANT PROFESSOR, CSED

TIET, PATIALA

Simple Linear Regression- Cost Function Overview

- We know, the linear function that binds the input variable x with the corresponding predicted value of (y^) is given by:

$$\hat{y} = \beta_0 + \beta_1 x$$

- The cost function (mean square error function) is given by:

$$J(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

- The cost function is a function of β_0 and β_1.

- Let's plot the cost function as function of β_1 considering β_0=0 (for the sake of simplicity i.e., 2D view). Consider the dataset shown in Table 1

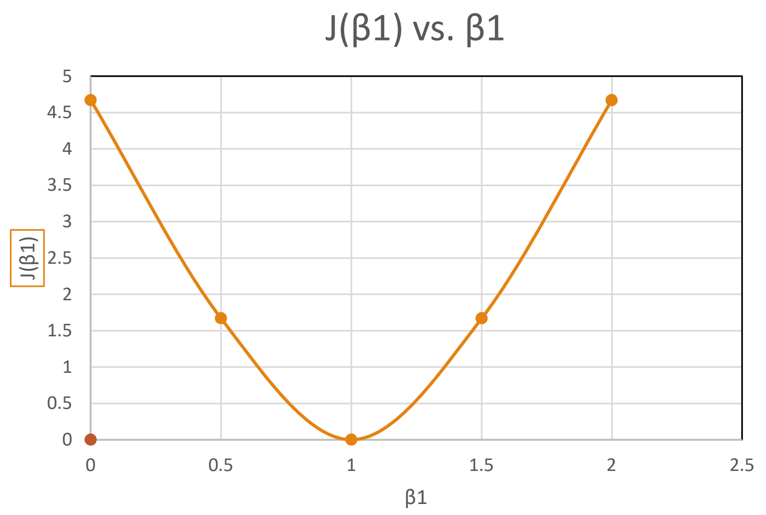
Table 1: Dataset for SLR containing 3 instances

Independent Variable (x_i)	Dependent Variable (y_i)
1	1
2	2
3	3

Plot of Cost Function of SLR

Table 2: Value of Cost Function $J(\beta_1)$ for different values of β_1 (using dataset shown in Table 1)

S.No	β_1	$J(\beta_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_1 x_i)^2$
1	1	$J(\beta_1) = \frac{1}{3}[(1-1)^2 + (2-2)^2 + (3-3)^2] = 0$
2	0.5	$J(\beta_1) = \frac{1}{3}[(1-0.5)^2 + (2-1)^2 + (3-1.5)^2] = 1.67$
3	0	$J(\beta_1) = \frac{1}{3}[(1-0)^2 + (2-0)^2 + (3-0)^2] = 4.67$
4	1.5	$J(\beta_1) = \frac{1}{3}[(1-1.5)^2 + (2-3)^2 + (3-4.5)^2] = 1.67$
5	2	$J(\beta_1) = \frac{1}{3}[(1-2)^2 + (2-4)^2 + (3-6)^2] = 4.67$

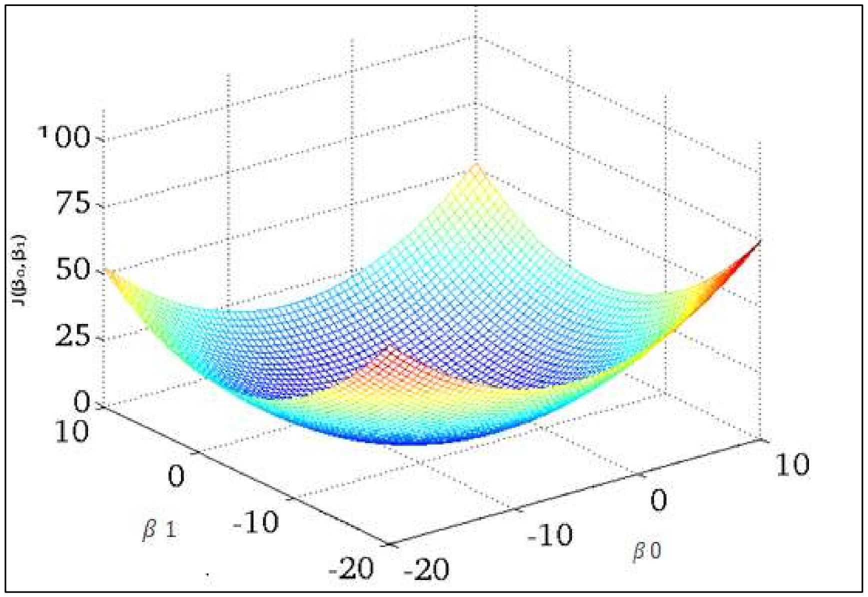


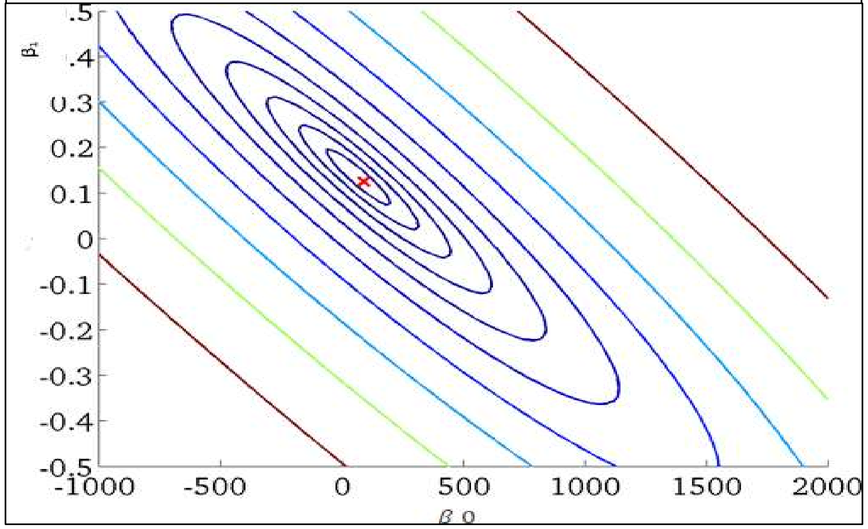
It is clear from the above function that the cost function is parabolic in shape (bowl-shaped) with one point of minimum where the mean square error is zero.

Plot of Cost Function of SLR

(Cost Function as function of β_0 and β_1)

SURFACE PLOT	CONTOUR PLOT
(BOWL SHAPED CURVE WITH ONLY ONE POINT OF MINIMUM)	(SAME COLOR LINES MEAN SAME VALUE OF COST FUNCTION AT DIFFERENT POINTS OF β_0 AND β_1)





Contour plot represents a 3-dimensional surface by **plotting** constant z slices, called **contours**, on a 2-dimensional format.

Gradient Descent Optimization- Introduction

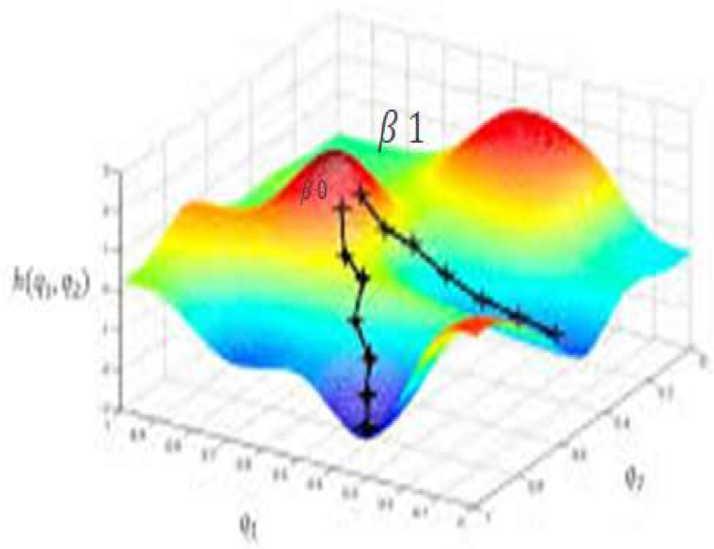
- **Gradient Descent** is an **optimization algorithm** for finding a local minimum of a differentiable function.
- **Gradient descent** is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible. i.e.,

$$argmin_{(\beta_0,\beta_1,\beta_2 \ldots\ldots,\beta_n)} J(\beta_0,\beta_1,\beta_2 \ldots \ldots .., \beta_n)$$

- It's based on minimizing a convex cost function and tweaks its parameters iteratively to minimize a given function to its local minimum.
- It considers gradient of the cost function to tune the parameters.
- Gradient can be considered as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning.
- In mathematical terms, a gradient is a partial derivative of the function with respect to its inputs.

Gradient Descent Optimization- Introduction

- The image illustrates the cost function from a top-down view and the black arrows are the steps of gradient descent algorithm.
- The algorithm will reach to different local or global minimum depending upon the initial value of β_0 and β_1.
- The gradient in this context is a vector that contains the direction of the steepest step the algorithm can take and also how long that step should be.



Steps of Gradient Descent Optimization

- In order to minimize any differentiable cost function, $J(\beta_0, \beta_1, \beta_2 \ldots \ldots .., \beta_k)$, containing parameters $\beta_0, \beta_1, \beta_2 \ldots \ldots .., \beta_k$, following steps are followed in gradient descent optimization:

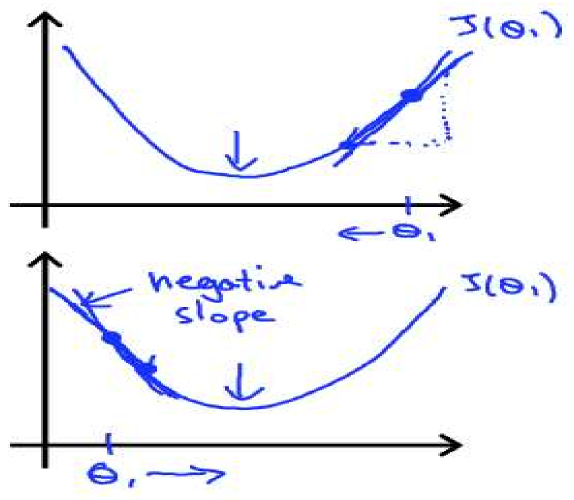
1. Initialize the parameters, $\beta_0, \beta_1, \beta_2 \ldots \ldots .., \beta_k$, to any arbitrary values. Usually, these are set to 0 initial value.
2. Update the values of parameters $\beta_0, \beta_1, \beta_2 \ldots \ldots .., \beta_k$, using the following equation (until convergence or for fixed number of iterations:

$$\beta_j = \beta_j - \alpha \frac{\partial(J(\beta))}{\partial \beta_j} \; for \; j = 0,1,2, \ldots \ldots .. k$$

- This update must be simultaneous i.e., the RHS of the above equations must be stored in temporary variables for each value of j and then simultaneously assigned.
- Here α is called the fixed step size that controls the step size and $\frac{\partial(J(\beta))}{\partial \beta_j}$ is called the gradient of cost function.
- Convergence of β_j's means that there is no change in value of β_j which will happen only when $\frac{\partial(J(\beta))}{\partial \beta_j} = 0$

Gradient Descent Optimization- Intuition

- The intuition behind gradient descent optimization is that it may start from any arbitrary point it may converge at some local or global minimum.
- For instance, consider cost function with only one parameter (θ_1). The shape of cost function is shown in the images 1 and 2.
- If we start from θ_1 as shown in figure 1, then gradient $\frac{\partial J(\theta_1)}{\partial \theta_1}$ is positive. Therefore, $\theta_1 = \theta_1 - positive\ quantity$. So, it will slowly move towards the minimum point.
- If we start from θ_1 as shown in figure 2, then gradient $\frac{\partial J(\theta_1)}{\partial \theta_1}$ is negative. Therefore, $\theta_1 = \theta_1 + positive\ quantity$. So, it will again slowly move towards the minimum point.



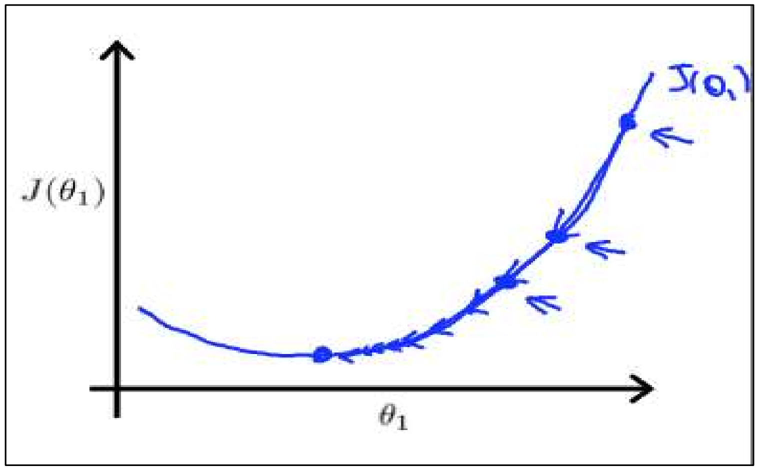
Learning Rate in Gradient Descent

Why is learning rate fixed?

- Gradient descent algorithm, can converge to a local minimum, even with fixed learning rate.

$$\theta_1 = \theta_1 - \alpha \frac{\partial(J(\theta))}{\partial\theta_1}$$

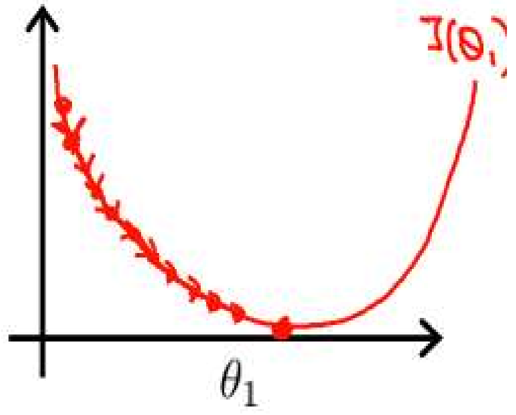
- As we approach a local minimum, gradient descent will automatically take smaller steps.
- So, no need to decrease learning rate over time.



Learning Rate in Gradient Descent Contd…

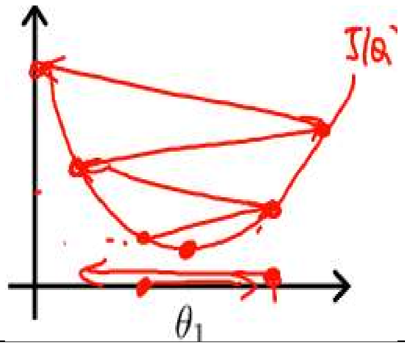
What if learning rate is too small?

- If learning rate is small, then gradient descent will take a lot of time to converge (as shown in the figure).



What if learning rate is too large?

- If learning rate is too large, then gradient descent can overshoot the minimum.
- It may fail to converge or even diverge (as shown in figure below).



Gradient Descent Optimization for Multiple Linear Regression (MLR)

- A multiple linear regression model with k independent predictor variables $x_1, x_2 \ldots, x_k$ predicts the output variable as:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots \ldots \ldots \ldots .. + \beta_k x_k$$

- The cost function (mean square error function) is given by:

$$J = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3} - \cdots \ldots \ldots \ldots .. - \beta_k x_{ik})^2$$

Gradient of the cost function with respect to input parameters is given by:

$$\frac{\partial J}{\partial \beta_0} = \frac{1}{n} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots \ldots \ldots \ldots .. + \beta_k x_{ik} - y_i)$$

Gradient Descent Optimization for MLR

Similarly, $\frac{\partial J}{\partial \beta_1} = \frac{1}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots \ldots \ldots \ldots .. + \beta_k x_{ik} - y_i) \times x_{i1}$

$$\frac{\partial J}{\partial \beta_2} = \frac{1}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots \ldots \ldots \ldots .. + \beta_k x_{ik} - y_i) \times x_{i2}$$

$$\frac{\partial J}{\partial \beta_3} = \frac{1}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots \ldots \ldots \ldots .. + \beta_k x_{ik} - y_i) \times x_{i3}$$

$$\vdots$$
$$\vdots$$
$$\vdots$$

In general, $\boldsymbol{\frac{\partial J}{\partial \beta_j} = \frac{1}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots \ldots \ldots \ldots .. + \beta_k x_{ik} - y_i) \times x_{ij}}$

Gradient Descent Optimization for MLR

- The gradient descent optimization for Multiple Linear Regression is summarized as below:

1. Initialize $\beta_0 = 0$, $\beta_1 = 0$, $\beta_2 = 0$,……………………………. $\beta_k = 0$
2. Update parameters until convergence or for fixed number of iterations using following equation:

$$\beta_j = \beta_j - \frac{\alpha}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots \ldots \ldots \ldots .. + \beta_k x_{ik} - y_i) \times x_{ij}$$

For j=0,1,2,3……………..k

Where x_{i0}=1 and k are the total number of iterations

Gradient Descent Optimization for MLR-Example

Consider the following dataset that shows the Stock Index Price as function of Interest Rate and Unemployment rate. For the given dataset show first iteration of gradient descent optimization for linear regression. Initialize β_0 =0 , $\beta_1 = 0,$, $\beta_2 = 0$ and consider learning rate as 0.01

Interest Rate (x_{i1})	Unemployment rate (x_{i2})	Stock Index Price (y_i)
2.75	5.3	1464
2.5	5.3	1394
2.25	5.5	1159
2	5.7	1130
2	5.9	1075
2	6	1047
1.75	5.9	965
1.75	6.1	719

Gradient Descent Optimization for MLR-Example

Initially, $\beta_0 = 0, \beta_1 = 0, , \beta_2 = 0$

Iteration I:

$$temp0 \coloneqq \beta_0 - \frac{\alpha}{n}\left(\sum_{i=1}^{n}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} - y_i)\right)$$

$$temp0 \coloneqq 0 - \frac{0.01}{8}(8 \times 0 + 17 \times 0 + 45.7 \times 0 - 8953) = 11.19$$

$$temp1 \coloneqq \beta_1 - \frac{\alpha}{n}(\sum_{i=1}^{n}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} - y_i) \times x_{i1})$$

$$temp1 \coloneqq 0 - \frac{0.01}{8}(17 \times 0 + 37 \times 0 + 96.4 \times 0 - 19569.75) = 24.46$$

$$temp2 \coloneqq \beta_2 - \frac{\alpha}{n}(\sum_{i=1}^{n}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} - y_i) \times x_{i2})$$

$$temp2 \coloneqq 0 - \frac{0.01}{8}(45.7 \times 0 + 96.4 \times 0 + 261.75 \times 0 - 50666.8) = 63.33$$

$\beta_0 = 11.19, \beta_1 = 24.46, , \beta_2 = 63.33$

S.No	(x_{i1})	(x_{i2})	(y_i)	$x_{i1}x_{i2}$	$x_{i1}y_i$	$x_{i2}y_i$	$(x_{i1})^2$	$(xi2_i)^2$
1	2.75	5.3	1464	14.575	4026	7759.2	7.5625	28.09
2	2.5	5.3	1394	13.25	3485	7388.2	6.25	28.09
3	2.25	5.5	1159	12.375	2607.75	6374.5	5.0625	30.25
4	2	5.7	1130	11.4	2260	6441	4	32.49
5	2	5.9	1075	11.8	2150	6342.5	4	34.81
6	2	6	1047	12	2094	6282	4	36
7	1.75	5.9	965	10.325	1688.75	5693.5	3.0625	34.81
8	1.75	6.1	719	10.675	1258.25	4385.9	3.0625	37.21
Total	17	45.7	8953	96.4	19569.75	50666.8	37	261.75