TASK - 6

REFERRAL CODE: SIRSS2105

SHUBHAM SALOKHE

1. Calculate/ derive the gradients used to update the parameters in cost function optimization for simple linear regression.

Referred from Andrew N G

A linear regression model attempts to explain the relationship between a dependent (output variables) variable and one or more independent (Predictor variable) variables using a straight line.
This straight line is represented using the following formula:

Straight line equation:

Hypothesis: $\hat{\boldsymbol{y}} = \boldsymbol{h_\theta(x)} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x}$

Were,

$\theta_{i's}$ = Parameters

y : Dependent variable,

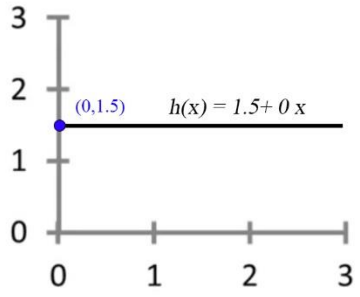
x : Independent variable,

θ_1: Slope of the line This straight line is represented using the following formula

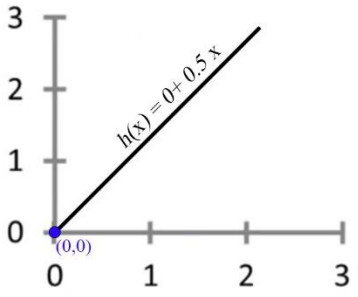
θ_0: y intercept (The value of y is $\theta_0$when the value of x is 0)

How to choose $\theta_{i's}$?

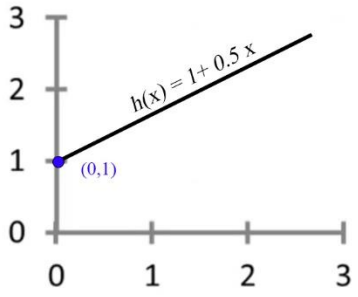
$$h_\theta(x) = \theta_0 + \theta_1 x$$



$\theta_0 = 1.5$
$\theta_1 = 0$



$\theta_0 = 0$
$\theta_1 = 0.5$

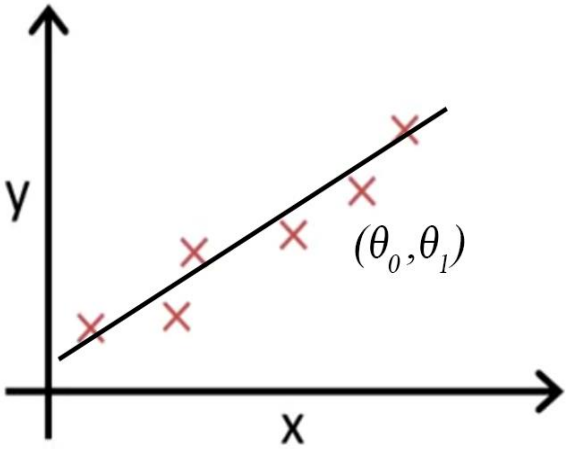


$\theta_0 = 1$
$\theta_1 = 0.5$

Cost function:

The cost is the error in our predicted value. We will use the **Mean Squared Error** function to calculate the cost.

The cost function is the technique of evaluating "the performance of our algorithm/model". It takes both predicted outputs by the model and actual outputs and calculates how much wrong the model was in its prediction. It outputs a higher number if our predictions differ a lot from the actual values. We will use Mean Squared Error(MSE) to calculate the error in our model.



$(x^{(i)}y^{(i)})$

Choose $\boldsymbol{\theta_0, \theta_1}$ so that $\boldsymbol{h_\theta(x)}$ is close to $\boldsymbol{y}$ for our training examples $(\boldsymbol{x}, \boldsymbol{y})$

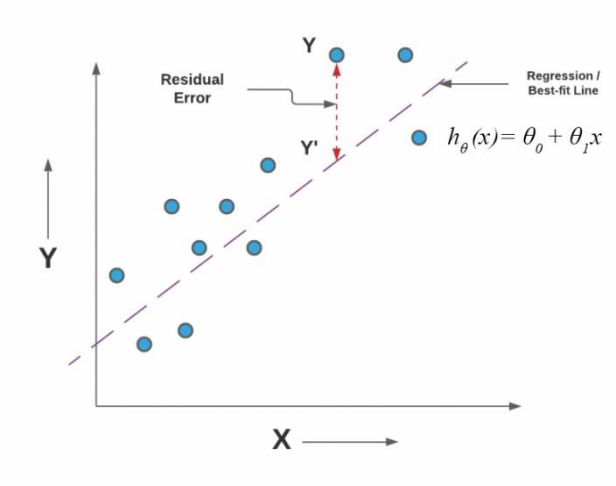
$$\min_{(\theta_0,\theta_1)} \frac{1}{2m}\sum_{i=0}^{n} h_\theta(x^{(i)}) - y^{(i)})^2$$

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

Cost function = $J(\theta_0, \theta_1)$

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=0}^{n} h_\theta(x^{(i)}) - y^{(i)})^2$$

$\min_{(\theta_0,\theta_1)} J(\theta_0, \theta_1)$ (Squared Error Function)



Hypothesis:

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

Parameters:

$$(\theta_0, \theta_1)$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=0}^{n} h_\theta(x^{(i)}) - y^{(i)})^2$$

Goal:

Our goal is to minimize the cost as much as possible in order to find the best fit line. We are not going to try all the permutation and combination of m and c (inefficient way) to find the best-fit line. For that, we will use Gradient Descent Algorithm.

$$\underset{\theta_0,\theta_1}{\text{minimize}}\, J(\theta_0, \theta_1)$$

The MSE measures the average amount that the model's predictions vary from the correct values, so you can think of it as a measure of the model's performance on the training set. The cost is higher when the model is performing poorly on the training set. The objective of the learning algorithm, then, is to find the parameters θ which give the minimum possible cost J.

This minimization objective is expressed using the following notation, which simply states that we want to find the θ which minimizes the cost $J(\theta)$.

$$\underset{\theta}{\text{minimize}}\, J(\theta)$$

Gradient Descent Algorithm:

Gradient Descent is an algorithm that finds the best-fit line for a given training dataset in a smaller number of iterations.

In this method, we will plot the values of m and against the MSE. The values for which the MSE will be minimum (Global minima) will be the final values for our model which will ultimately give us the best fit line in predicting the dependent variable.

Gradient Descent Minimization - Single Variable Example

We're going to be using gradient descent to find θ that minimizes the cost. But let's forget the MSE cost function for a moment and look at gradient descent as a minimization technique in general.

Let's take the much simpler function $J(\theta)=\theta_2$, and let's say we want to find the value of θ which minimizes $J(\theta)$.

Gradient descent starts with a random value of θ, typically θ=0, but since θ=0 is already the minimum of our function θ2, let's start with θ=3.

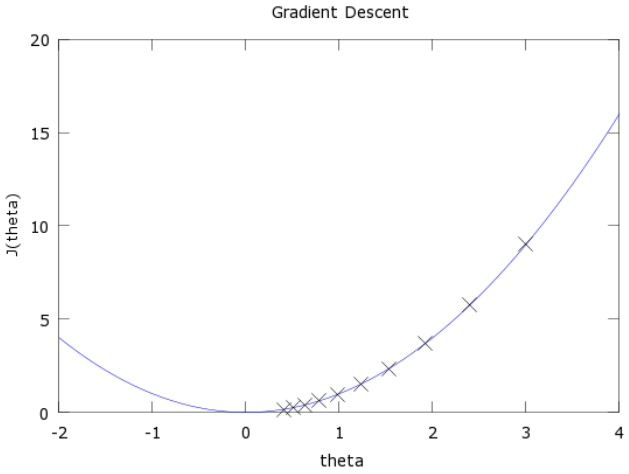
Gradient descent is an iterative algorithm which we will run many times. On each iteration, we apply the following "update rule" (the: = symbol means replace theta with the value computed on the right):

$$\theta := \theta - \alpha \frac{d}{d\theta} J(\theta)$$

Alpha is a parameter called the **learning rate** which we'll come back to, but for now we're going to set it to 0.1. The derivative of $J(\theta)$ is simply 2θ

$$\alpha = 0.1, \qquad \frac{d}{d\theta} J(\theta) = 2\theta$$

Below is a plot of our function, $J(\theta)$, and the value of θ over ten iterations of gradient descent.



Iteration	θ	$\alpha \frac{d}{d\theta} J(\theta)$
1	3	0.5
2	2.4	0.48
3	1.92	0.384
4	1.536	0.307
5	1.229	0.256
6	0.983	0.197
7	0.786	0.157
8	0.629	0.126
9	0.503	0.101
10	0.403	0.081

Cost Function Derivative

Why does gradient descent use the derivative of the cost function? Finding the slope of the cost function at our current θ value tells us two things.

The first is the direction to move theta in. When you look at the plot of a function, a positive slope means the function goes upward as you move right, so we want to move left in order to find the minimum. Similarly, a negative slope means the function goes downward towards the right, so we want to move right to find the minimum.

The second is how big of a step to take. If the slope is large, we want to take a large step because we're far from the minimum. If the slope is small, we want to take a smaller step. Note in the example above how gradient descent takes increasingly smaller steps towards the minimum with each iteration.

The Learning Rate - Alpha

The learning rate gives the engineer some additional control over how large of steps we make.

Selecting the right learning rate is critical. If the learning rate is too large, you can overstep the minimum and even diverge. For example, think through what would happen in the above example if we used a learning rate of 2. Each iteration would take us farther away from the minimum!

The only concern with using too small of a learning rate is that you will need to run more iterations of gradient descent, increasing your training time.

Convergence / Stopping Gradient Descent

Note in the above example that gradient descent will never actually converge on the minimum, θ =0. Methods for deciding when to stop gradient descent are beyond my level of expertise, but I can tell you that when gradient descent is used in the assignments in the Coursera course, gradient descent is run for a large, fixed number of iterations (for example, 100 iterations), with no test for convergence

2. What does the sign of gradient say about the relationship between the parameters and cost function?

   Gradient descent is an operation to find the local minimum of cost function. So while finding local minima the size of steps taken are related to the sign of gradient. We should take steps which are proportional to negative of gradient at the current point.

   for e.g.: m = m - LDm Where D is the gradient, L is the learning rate, m is coefficient or slope m So, when gradient is positive, steep size will increase as follows m=m-L (+D) m

   When gradient is negative, step size will decrease as follows m=m-L (-D) = m + L m D

3. Why Mean squared error is taken as the cost function for regression problems.

   The Mean Squared Error (MSE) is one of the simplest and most common loss

   functions. It is defined by the equation:

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2$$

   The MSE tells us how close a regression line is to a set of points. It will never be

   negative, since the errors are always squared to remove the negative signs. It also

   gives more weight to larger differences.

   The error keeps decreasing as the algorithm gains more and more experience. The

   smaller the MSE, the closer we are to finding the line of best fit.We use MSE to ensure that our trained model has no outlier predictions with huge

   errors, since the MSE puts larger weight on these errors due to the squaring part of

   the function.

4. What is the effect of learning rate on optimization, discuss all the cases?

   Learning rate (λ) is one such **hyper-parameter** that defines the **adjustment in the weights of our network with respect to the loss gradient descent.** It determines how fast or slow we will move towards the optimal weights.

   Smaller learning rate will result in values of m and c to take smaller steps while larger larger learning rate will result into larger steps.

   If learning rate is too large then the model may converge to a local minima giving a false optimal solution and if it is too small then model will converge very slowly requiring more number of epochs.

   Hence it is necessary to have proper learning rate value and it is usually between 0.0 and 1.0.