























Gradient Descent:

Identifying weights randomly (using brute force) requires a lot of amount time to reach the best possible weights. As the number of features or dimensions increases then curse of dimensionality comes into the picture.

Start at random weights and then check the slope at that point, based on that take the next step in the direction.

Basically dJ/d\*theta >> differentiate cost function w.r.t. theta/weights. Basically we are checking how cost function is getting changed with respective to change in the weights.

Once you find this value, remove this value from actual weights.

Updated weights = weights – alpha \* dJ/d. theta

* learning rate = alpha

update these weights till the time there is decrease in the cost, once the cost stops decreasing then you stop updating the weights as we may have reached the best possible weights where the cost is low.

Ideally cost function is, sum of squared errors ½(y-y^)2

* convex problem, pick cost function in such a way that we are solving convex problem

On the y axis, cost function and on the X axis weight (in one dimension)





