# Linear Regression In One Variable (Univariable)

Supervised Learning on regression problem (continuous output).

There is a dataset called the training set.

Notation:

**m = Number of training examples**

**x’s = input variable/features**

**y’s = output variable/target**

**(x, y) = one training example**

**(xi, yi) = ith training example**

**(I start from 1)**

Training Set is fed to the Learning Algorithm and then the input is provided and the appropriate results is outputted.

Input -> hypothesis (h) -> Estimated Output

How to represent hypothesis.

**hƟ(x) = Ɵ0 +Ɵ1x**

(h is henceforth predicted to be straight line)

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function h : X → Y so that h(x) is a “good” predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis. Seen pictorially, the process is therefore like this:



# Cost Function

hƟ(x) = Ɵ0 +Ɵ1x

*How to choose Ɵi’s:*

*Ɵ0 is the offset and Ɵ1 is the slope*

Choose such that h is close to y for our training samples (x, y).

***Minimise (Ɵ, Ɵ1) the J(Ɵ, Ɵ1) = {Ʃ(hƟ(xi) – yi )2}/2m***

Cost Function (Squared error function)



# Cost Function Intuition I

hƟ(x) = Ɵ1x

Passing through Origin.

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make a straight line (defined by *hθ*​(*x*)) which passes through these scattered data points.

Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. Ideally, the line should pass through all the points of our training data set. In such a case, the value of *J*(*θ*0​, *θ*1​) will be 0. The following example shows the ideal situation where we have a cost function of 0.

When *θ*1​=1, we get a slope of 1 which goes through every single data point in our model. Conversely, when \theta\_1 = 0.5*θ*1​=0.5, we see the vertical distance from our fit to the data points increase.



This increases our cost function to 0.58. Plotting several other points yields to the following graph:



# Cost Function Intuition II

Contour Figures/Plots

A contour plot is a graph that contains many contour lines. A contour line of a two variable function has a constant value at all points of the same line. An example of such a graph is the one to the right below.



Taking any colour and going along the 'circle', one would expect to get the same value of the cost function. For example, the three green points found on the green line above have the same value for *J*(*θ*0​, *θ*1​) and as a result, they are found along the same line. When *θ*0​ = 360 and *θ*1​ = 0, the value of *J*(*θ*0​, *θ*1​) in the contour plot gets closer to the centre thus reducing the cost function error. Now giving our hypothesis function a slightly positive slope results in a better fit of the data.



# Gradient Descent

Used to minimise functions.

Repetitive Convergence/ Divergence

a :=b => a=b;

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields *θ*0​ and *θ*1​ (actually we are graphing the cost function as a function of the parameter estimates). We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters.

We put *θ*0​ on the x axis and *θ*1​ on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters. The graph below depicts such a setup.



We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum. The red arrows show the minimum points in the graph.

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter α, which is called the learning rate.

For example, the distance between each 'star' in the graph above represents a step determined by our parameter α. A smaller α would result in a smaller step and a larger α results in a larger step. The direction in which the step is taken is determined by the partial derivative of  *J*(*θ*0​,*θ*1​). Depending on where one starts on the graph, one could end up at different points. The image above shows us two different starting points that end up in two different places.

The gradient descent algorithm is:

repeat until convergence:

*θj*:=*θj*−*α(*∂/∂*θ)J*(*θ*0,*θ*1)

*α => learning rate*

where

j=0,1 represents the feature index number.

At each iteration j, one should simultaneously update the parameters *θ*1​,*θ*2​,...,*θn*​. Updating a specific parameter prior to calculating another one on the *j*(*th*) iteration would yield to a wrong implementation.



# Gradient Descent Algorithm

In this video we explored the scenario where we used one parameter *θ*1​ and plotted its cost function to implement a gradient descent. Our formula for a single parameter was :

Repeat until convergence:

|  |
| --- |
| *θ*1​:=*θ*1​−*α((d/dθ)1*​*J*(*θ*​)) |

Regardless of the slope's sign for *((d/dθ1*​)*J*(*θ1*​)), *θ*1​ eventually converges to its minimum value. The following graph shows that when the slope is negative, the value of *θ*1​ increases and when it is positive, the value of *θ*1​ decreases.



On a side note, we should adjust our parameter *α* to ensure that the gradient descent algorithm converges in a reasonable time. Failure to converge or too much time to obtain the minimum value imply that our step size is wrong.



### **How does gradient descent converge with a fixed step size *α*?**

The intuition behind the convergence is *((d/dθ1*​)*J*(*θ1*​)), ​ approaches 0 as we approach the bottom of our convex function. At the minimum, the derivative will always be 0 and thus we get:

|  |
| --- |
| *θ*1​:=*θ*1​−*α*∗0 |

# Gradient Descent For Linear Regression

For the specific choice of cost function  *J*(*θ*0​,*θ*1​) used in linear regression, there are no local optima (other than the global optimum).

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to :

|  |
| --- |
| repeat until convergence: {  *θ*0:= *θ*0−*α(*1/*m)*∑(*hθ*(*xi*)−*yi*)  *θ*1:= *θ*1−*α(*1/*m)*∑((*hθ*(*xi*)−*yi*)*xi*  } |

where m is the size of the training set,*θ*0​ a constant that will be changing simultaneously with *θ*1​ and *xi*​,*yi*​are values of the given training set (data).

Note that we have separated out the two cases for *θj*​ into separate equations for *θ*0​ and *θ*1​; and that for *θ*1​ we are multiplying *xi*​ at the end due to the derivative. The following is a derivation of ∂/∂*θjJ*(*θ*) for a single

example :



The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

So, this is simply gradient descent on the original cost function J. This method looks at every example in the entire training set on every step, and is called **batch gradient descent**. Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a convex quadratic function. Here is an example of gradient descent as it is run to minimize a quadratic function.

