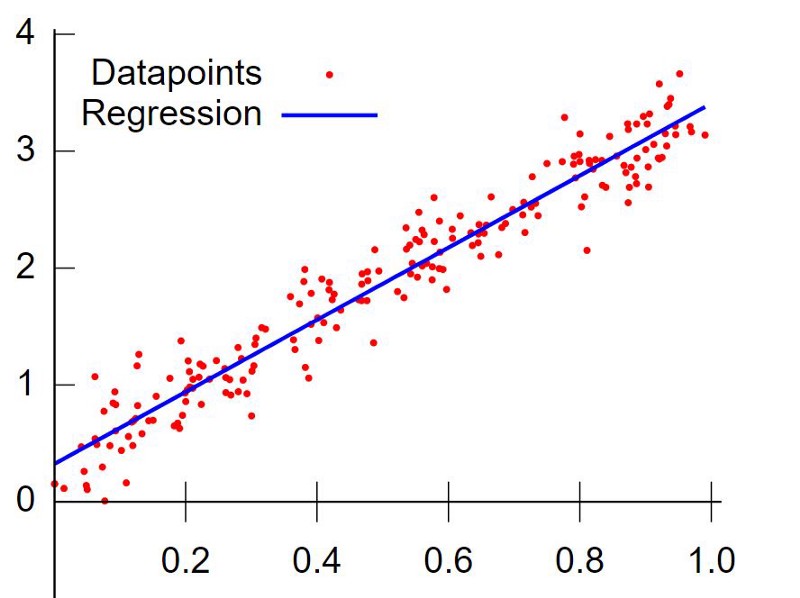
**Chapter 1 :Complete Linear Regression with Math.**

Prerequisite :**[Different types of machine learning.](https://medium.com/@madhusanjeevi.ai/different-types-of-machine-learning-and-their-types-5660bacfa20f" \t "_blank)**

**Linear Regression:** it is a linear model that establishes the relationship between a dependent variable *y(Target)* and one or more independent variables denoted *X(Inputs)*.



*Regression fits the data*

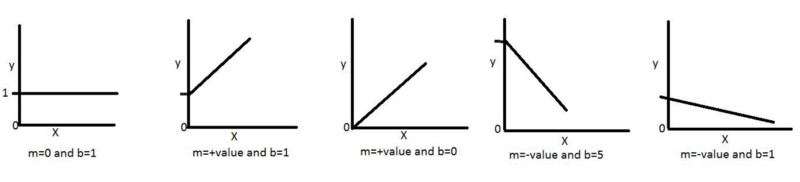
**Goal is to find that blue straight line (which is best fit) to the data.**

Our Training Data consists of X and y values so we can plot them on the graph, that’s damn easy. now *what’s next?* how to find that blue line????

First lets talk about how to draw a linear line in the graph,

In math we have an equation which is called linear equation

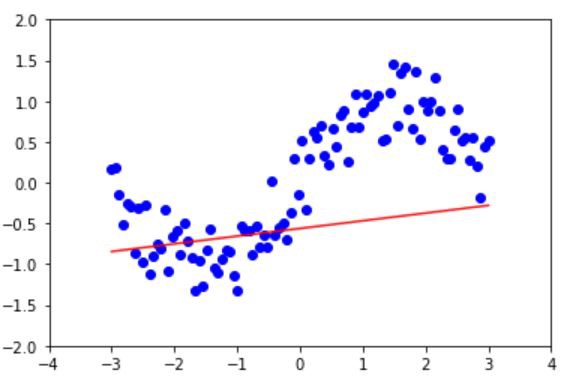
y = mX+b { m->slope , b->Y-intercept }



so we can draw the line if we take any values for m and b

How do we get the m and b values ??? and how do we know exact m and b values for the best fit line??

Lets take a simple data set (sine wave form -3 to 3) and First time we take random values of m and b values and we draw a line something like this.



Random line for m and b

How we drew the above line?

we take the first X value(x1) from our data set and calculate y value(y1)

y1=m\*x1+b {m,b->random values lets say 0.5,1   
 x1->lets say -3 (first value from our data-set)

y1=(0.5 \* -3) + 1  
y1=-0.5   
by applying all x values for m and b values we get our first line.

Above picture has its own random variables ( I hope you understand the concept)

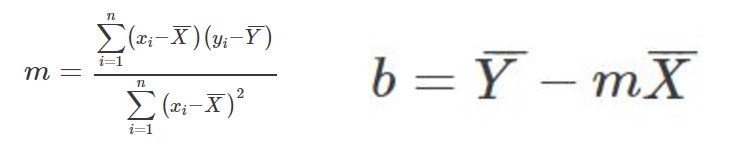
That line is *not* fitting well to the data so we need to change m and b values to get the best fit line.

How do we change m and b values for the best fit line??

Either we can use an awesome algorithm called **Gradient Descent** (Which I will cover in next story with also the math used in there.)

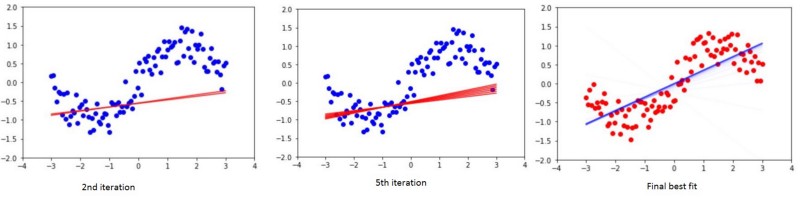
**Update:**Here is the [***Gradient Descent***](https://medium.com/@madhusanjeevi.ai/chapter-1-2-gradient-descent-with-math-ad303eb33be8) Story

Or we can borrow direct formulas from statistics(they call this **Least Square Method**) I will also cover if possible in next story.



X^ is mean of X values , Y^ mean of y values

Right now lets black box, we assume that we are getting the m and b values, Every time when the m and b values change we may get a different line and finally we get the best fit line



Pretty cool right?

So *What’s next??? Predicting new data, remember?? so* we give new X values we get the predicted y values how does it work ??

same as above **y= m X +b** , we now know the final m and b values.

This is called **simple linear regression** as we have only one independent X value. Lets say we wanna predict housing price based the size of house

X= Size (in sqft’s) y= Price (in dollar’s)

X y  
1000 40  
2000 70  
500 25  
............

What if we have more independent values of X????

Lets say we wanna predict housing price not only by the size of house but also by no of bedrooms

x1= Size (in sqft’s), x2=N\_rooms and y= Price (in dollar’s)

x1 x2 y  
1000 2 50  
2000 4 90  
500 1 35  
............

The process same as above but the equation changes a bit

Note: Lets alias b and m as − θ0 and θ1 (theta 0 and theta 1 ) respectively.

y = θ0+θ1\*X → b+mX → Simple LR → Single variable LR

y=θ0+θ1\*x1+θ2\*x2+..θn\*xn → Multiple LR → Multi variable LR

Now we can predict as many things as we wish.

That’s it for this story , Hoping it helps at least 1 person.

In the next story I will talk about [**Gradient Descent Algorithm**](https://medium.com/@madhusanjeevi.ai/chapter-1-2-gradient-descent-with-math-ad303eb33be8)**.**

until then See ya!

# Chapter 1.2: Gradient Descent with Math.

This story I wanna talk about a famous machine learning algorithm called Gradient Descent which is used for optimizing the machine leaning algorithms and how it works including the math.

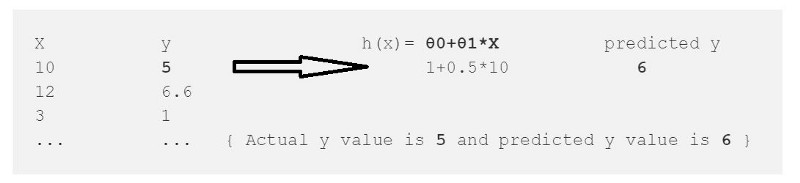
From [chapter 1](https://medium.com/@madhusanjeevi.ai/complete-linear-regression-with-math-edb05500e7ee) we know that we need to update m and b values, we call them **weights** in machine learning. Lets alias b and m as − θ0 and θ1 (theta 0 and theta 1 ) respectively.

First time we take random values for θ0 and θ1, and we calculate y

**y = θ0+θ1\*X**   
In machine learning we say hypothesis so **h(X) = θ0+θ1\*X**

h(X)=y but this y is not actual value in our data-set, this is predicted y from our hypothesis.

For example lets say our data-set is something like below and we take random values which are **1** and **0.5** for **θ0** and **θ1** respectively.

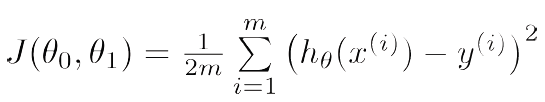


From this we calculate the error which is

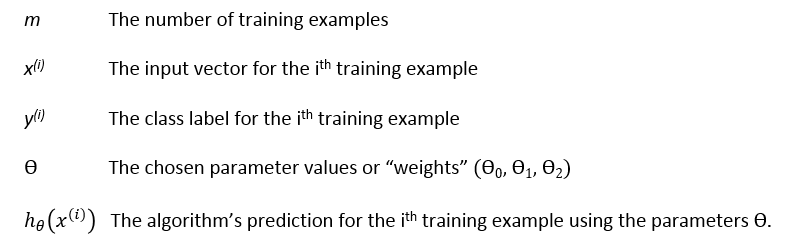
error = (h(x)-y)² --> (Predicted - Actual)²   
error = (6-5)² = 1

² is to get rid of negative values (what if Actual y=6 and Py=5)

we just calculated the error for one data point in our data-set , we need to repeat this for all data points in our data set and sum up the all errors to one error which is called **Cost Function ‘J(θ)’** in machine learning.



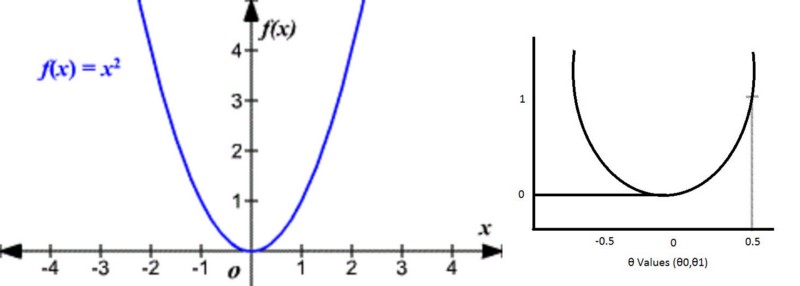
Cost Function.



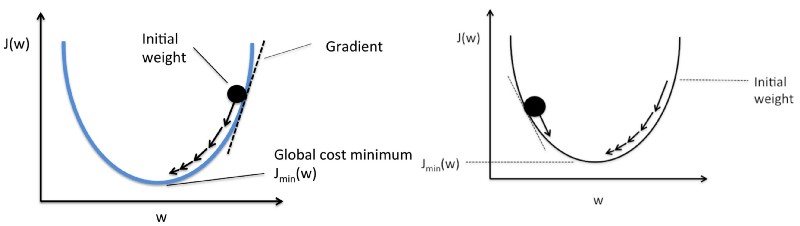
Our goal is to minimize the cost function (error) **we want our error close to zero Period.**

we have the error **1** for first data-point so lets treat that as whole error and reduce to zero for sake of understanding.

for (h(x)-y)² function we get always positive values and graph will look like this(Left) and lets plot the error graph.



Here is the gradient descent work comes into the picture.



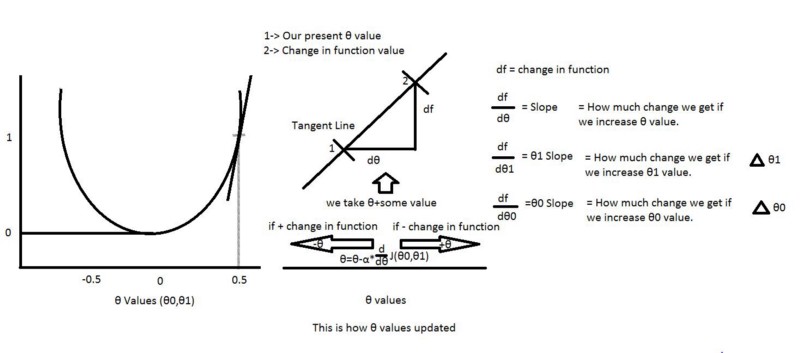
+**θ values (Left), -θ values(Right)**

By taking the little steps down to reach the minimum value (bottom of the curve) and changing the **θ** values in the process.

How does it know how much value it should go down???

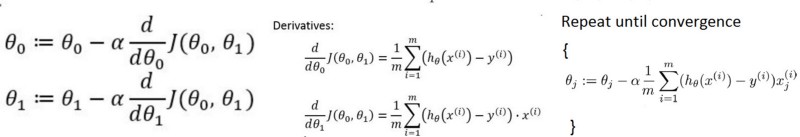
The answer is in Math.

1. It draws the line(Tangent) from the point.
2. It finds the slope of that line.
3. It identifies how much change is required by taking the partial derivative of the function with respective to **θ**
4. The change value will be multiplied with a variable called **alpha**(learning rate) we provide the value for alpha usually 0.01
5. It subtracts this change value from the earlier **θ** value to get new **θ** value .



From above picture we can define our **θ0 and θ1.**

And alpha here is a learning rate usually we give 0.01 but it depends, it tells how big the step-size is towards reaching the minimum value.

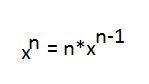


**θ0 and θ1 values(Left),more than two θ’s (Right)**

Again we know our J(**θ0,θ1)** so if we apply this to above equations for **θ0** and **θ1**, we get our new **θ0 and θ1** values**.**

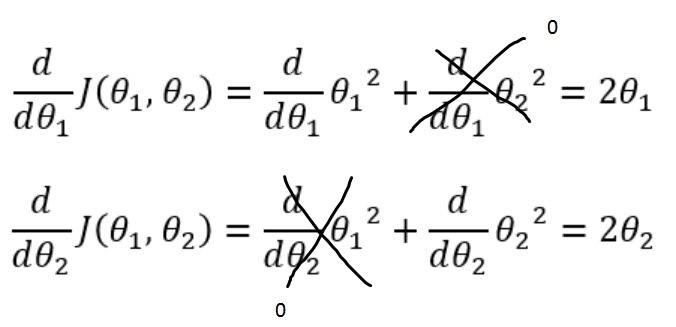
How to calculate the derivatives???

For example f(x) =x² → df/dx=2x How ???

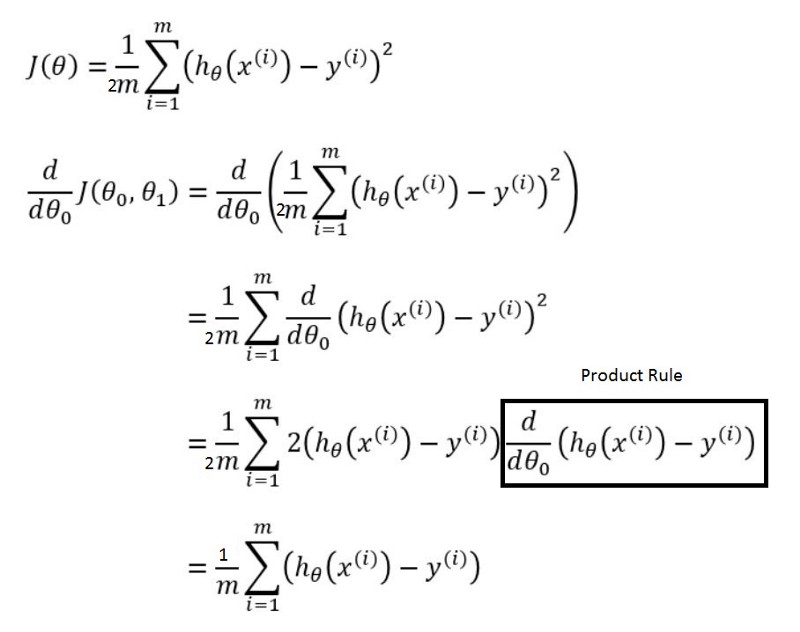


How to calculate the partial derivatives???

its same as calculating derivatives but here we calculate the derivative with respective to that value , others are constants (so d/dx(constant)=0)

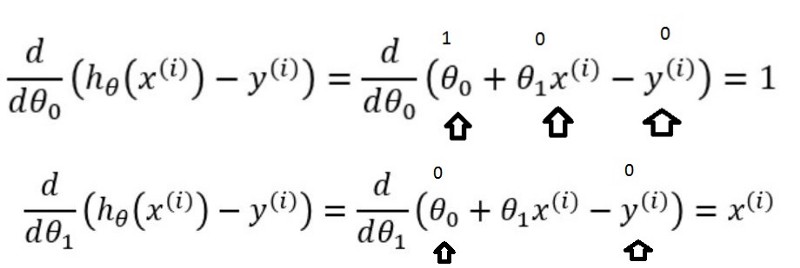


The same thing we can apply for calculating partial derivative with respective to **θ0** and **θ1**.



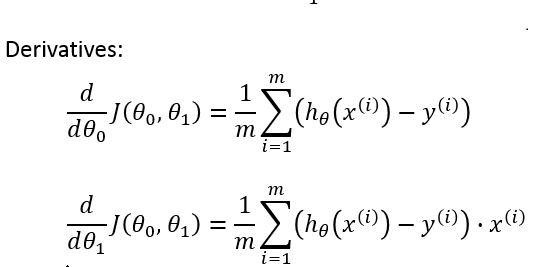
How come that box drawn disappeared in the next step above? just wait and see.

For calculating partial derivative with respective to **θ1** is also same as above except one little part is added



**θ0 box disappeared because value is 1 (Top)**

So Final picture is



Final **θ0** and **θ1**values

Hope its not confusing , and I know its little bit hard to grasp in the beginning but I am sure that this will make sense as you go through again and again.

So That’s it for this story , In the next story I will cover another interesting topic in machine learning so See ya!

**Machine Learning week 1: Cost Function, Gradient Descent and Univariate Linear Regression**

I have started doing Andrew Ng’s popular machine learning course on Coursera. The first week covers a *lot,* at least for someone who hasn’t touched much calculus for a few years

* Cost Functions (mean difference squared)
* Gradient Descent
* Linear Regression

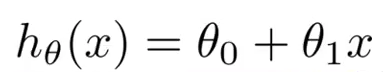
These three topics were a lot to take in. I’ll talk about each in detail, and how they all fit together, with some python code to demonstrate.

*Edit May 4th: I published a follow up focusing on how the Cost Function works* [*here*](https://medium.com/@lachlanmiller_52885/understanding-and-calculating-the-cost-function-for-linear-regression-39b8a3519fcb)*, including an intuition, how to calculate it by hand and two different Python implementations. I can do gradient descent and then bring them together for linear regression soon.*

**Model Representation**

First, the goal of most machine learning algorithms is to construct a model: a hypothesis that can be used to estimate Y based on X. The hypothesis, or model, maps *inputs* to *outputs.* So, for example, say I train a model based on a bunch of housing data that includes the size of the house and the sale price. By training a model, I can give you an estimate on how much you can sell your house for based on it’s size. This is an example of a regression problem — given some input, we want to predict a continuous output.

The hypothesis is usually presented as



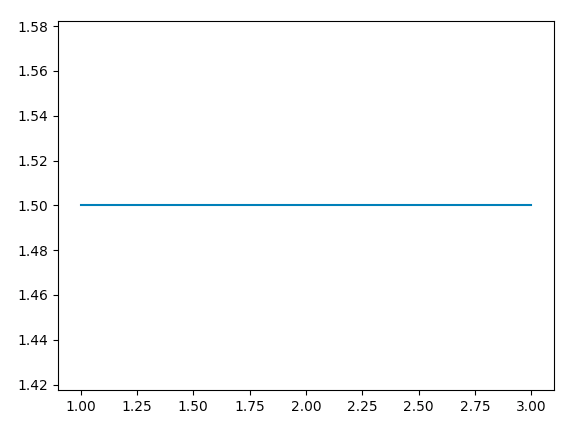
Hypothesis

The theta values are the *parameters.*

Some quick examples of how we visualize the hypothesis:



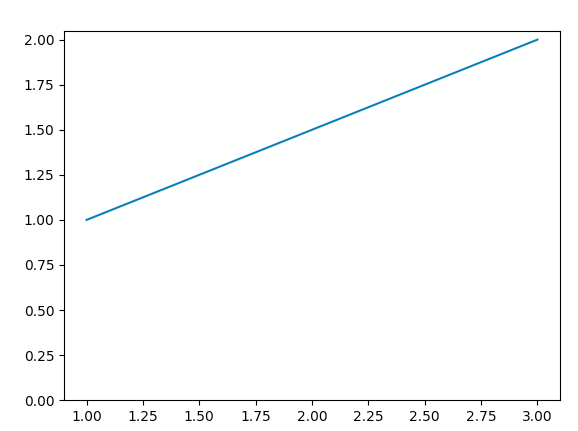
This yields h(x) = 1.5 + 0x. 0x means no slope, and y will always be the constant 1.5. This looks like:



h(x) = 1.5 + 0x

How about

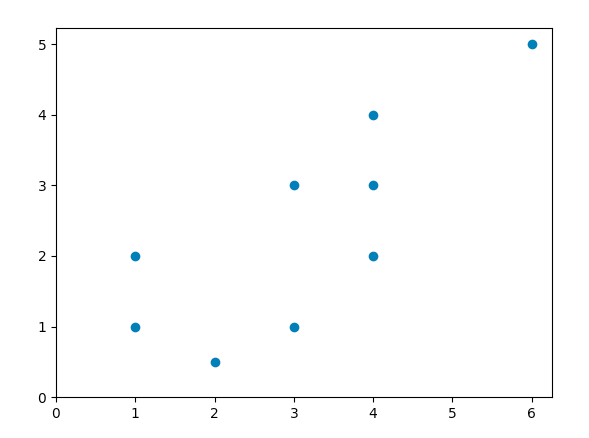




h(x) = 1 + 0.5x

The goal of creating a model is to choose parameters, or theta values, so that h(x) is close to y for the training data, x and y. So for this data

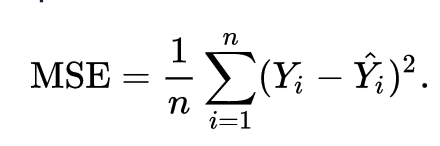
x = [1, 1, 2, 3, 4, 3, 4, 6, 4]  
y = [2, 1, 0.5, 1, 3, 3, 2, 5, 4]



I will try and find a line of best fit using *linear regression.* Let’s get started.

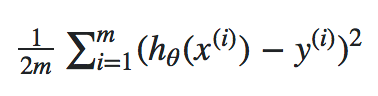
**Cost Function**

We need a function that will minimize the parameters over our dataset. One common function that is often used is [mean squared error](https://en.wikipedia.org/wiki/Mean_squared_error), which measure the difference between the estimator (the dataset) and the estimated value (the prediction). It looks like this:



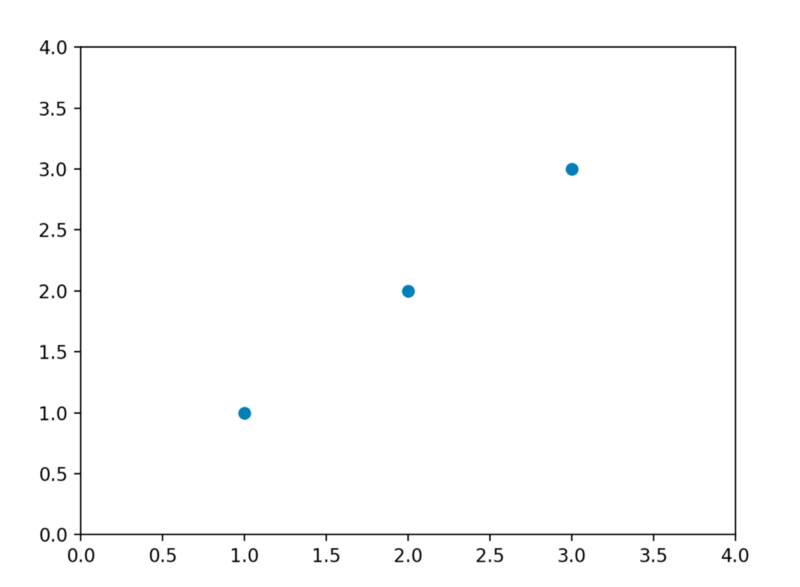
Mean Squared Error

It turns out we can [adjust the equation a little](https://datascience.stackexchange.com/questions/10188/why-do-cost-functions-use-the-square-error) to make the calculation down the track a little more simple. We end up with:



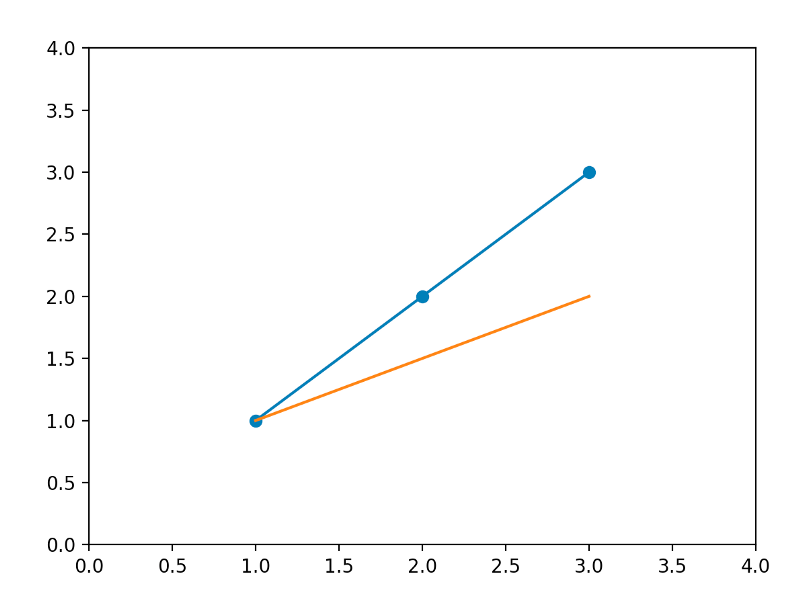
Mean Squared Error

Let’s apply this const function to the follow data:



For now we will calculate some theta values, and plot the cost function by hand. Since this function passes through (0, 0), we are only looking at a single value of theta. From here on out, I’ll refer to the cost function as J(ϴ).

For J(1), we get 0. No surprise — a value of J(1) yields a straight line that fits the data perfectly. How about J(0.5)?

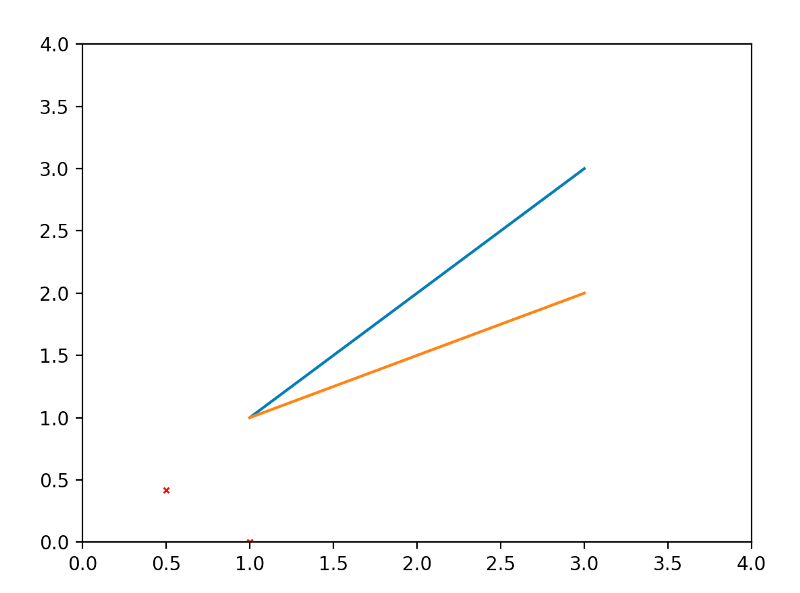


J(0.5)

The MSE function gives us a value of 0.58. Let’s plot both our values so far:

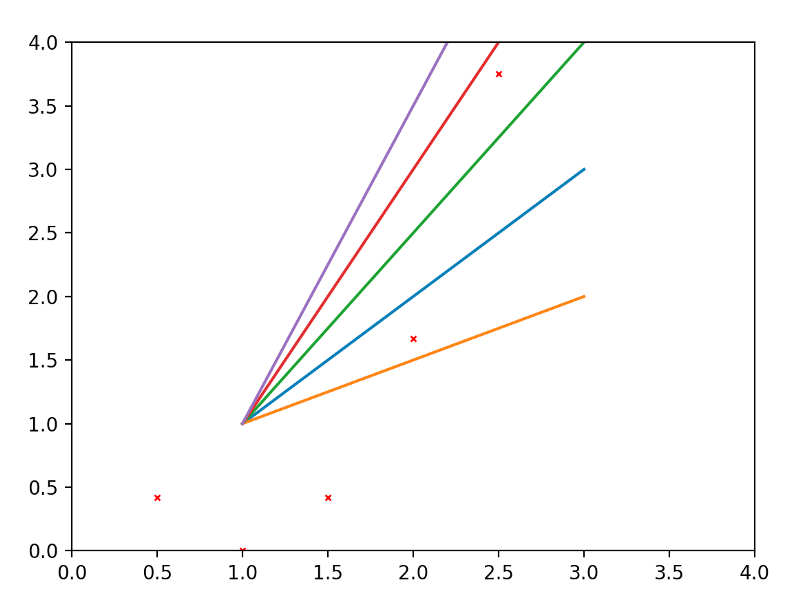
J(1) = 0

J(0.5) = 0.58

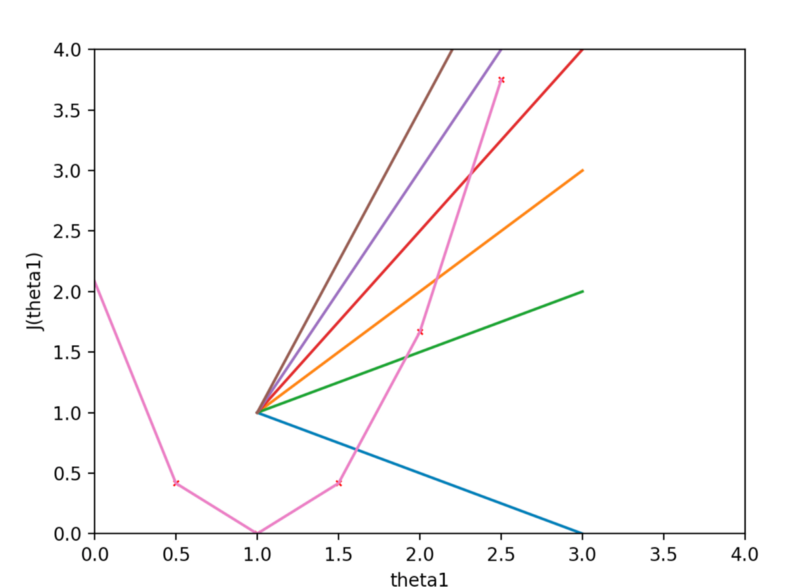


With J(1) and J(0.5)

I’ll go ahead and calculate some more values of J(ϴ).



And if we join the dots together nicely…



Visualizing the cost function J(ϴ)

We can see that the cost function is at a minimum when theta = 1. This makes sense — our initial data is a straight line with a slope of 1 (the orange line in the figure above).

**Gradient Descent**

We minimized J(ϴ) by trial and error above — just trying lots of values and visually inspecting the resulting graph. There must be a better way? Queue *gradient descent.* Gradient Descent is a general function for minimizing a function, in this case the Mean Squared Error cost function.

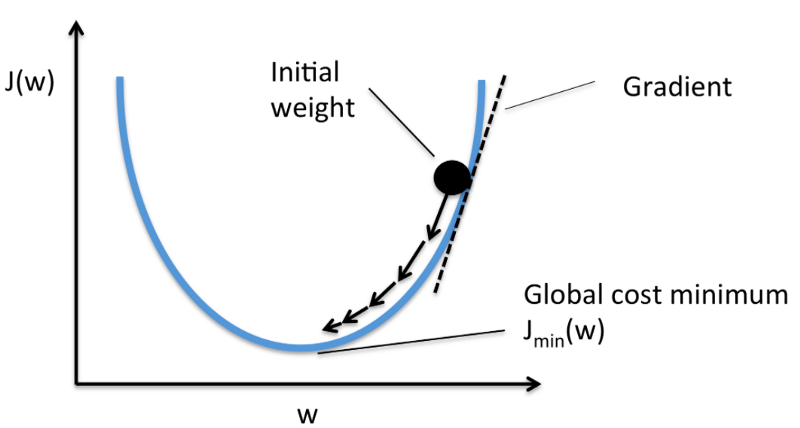
Gradient Descent basically just does what we were doing by hand — change the theta values, or parameters, bit by bit, until we *hopefully* arrived a minimum.

We start by initializing theta0 and theta1 to any two values, say 0 for both, and go from there. Formally, the algorithm is as follows:

https://cdn-images-1.medium.com/max/800/1*QKHtyn4Rr-0R-s0an1eSsA.png

Gradient Descent

where α, alpha, is the learning rate, or how quickly we want to move towards the minimum. If α is too large, however, we can overshoot.

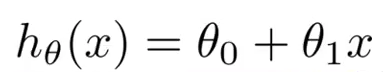


Gradient Descend Visualization. Credit: [rasbt.github.io](https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/)

**Bringing it all together — Linear Regression**

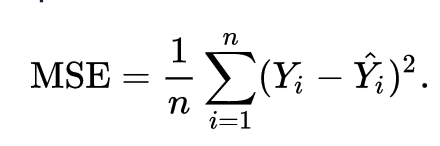
Quickly summarizing:

We have a hypothesis:



Hypothesis

which we need fit to our training data. We can use a cost function such Mean Squared Error:



Mean Squared Error

which we can minimize using gradient descent:

https://cdn-images-1.medium.com/max/800/1*QKHtyn4Rr-0R-s0an1eSsA.png

Gradient Descent

Which leads us to our first machine learning algorithm, *linear regression.* The last piece of the puzzle we need to solve to have a working linear regression model is the partial derivate of the the cost function:

https://cdn-images-1.medium.com/max/800/1*ZtBYLccJZnG0z_pQ6DNxtQ.png

Partial Derivate of the Cost Function which we need to calculate

Which turns out to be:

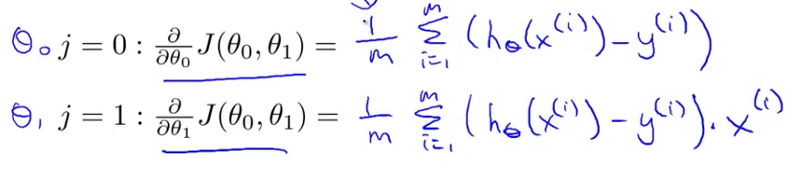
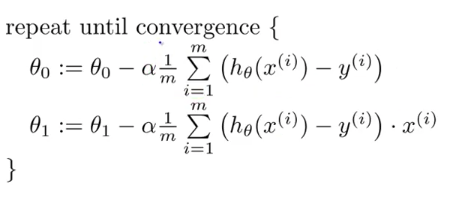


Image from Andrew Ng’s machine learning course on Coursera.com

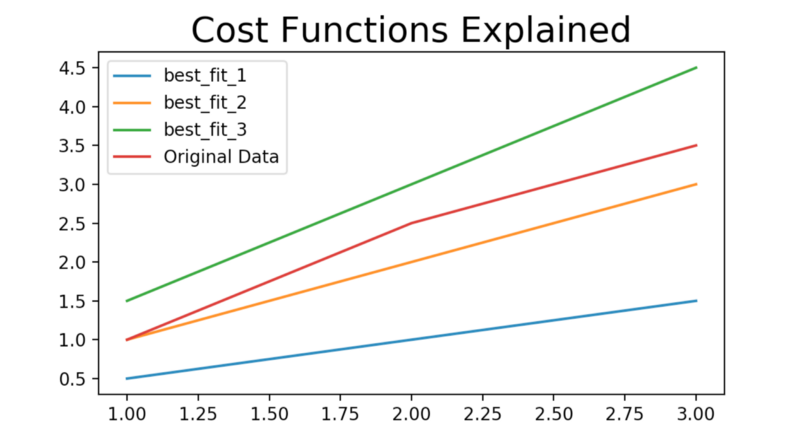
Which gives us linear regression!



Linear Regression

With the theory out of the way, I’ll go on to implement this logic in python in the next post.

*Edit May 4th: I published a follow up focusing on how the Cost Function works* [*here*](https://medium.com/@lachlanmiller_52885/understanding-and-calculating-the-cost-function-for-linear-regression-39b8a3519fcb)*, including an intuition, how to calculate it by hand and two different Python implementations. I can do gradient descent and then bring them together for linear regression soon.*



#### nderstanding and Calculating the Cost Function for Linear Regression

This post will focus on the properties and application of cost functions, how to solve it them by hand. Then we will implement the calculations twice in Python, once with for loops, and once with vectors using numpy. This goes into more detail than my [previous article](https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd) about linear regression, which was more a high level summary of the concepts.

When learning about linear regression in [Andrew Ng’s Coursera course](https://www.coursera.org/learn/machine-learning/), two functions are introduced:

* the cost function
* gradient descent

At first I had trouble understanding what each was for. Together they form [linear regression](https://en.wikipedia.org/wiki/Linear_regression), probably the most used learning algorithm in machine learning.

#### What is a Cost Function?

In the case of gradient descent, the objective is to find a line of best fit for some given inputs, or X values, and any number of Y values, or outputs. A cost function is defined as:

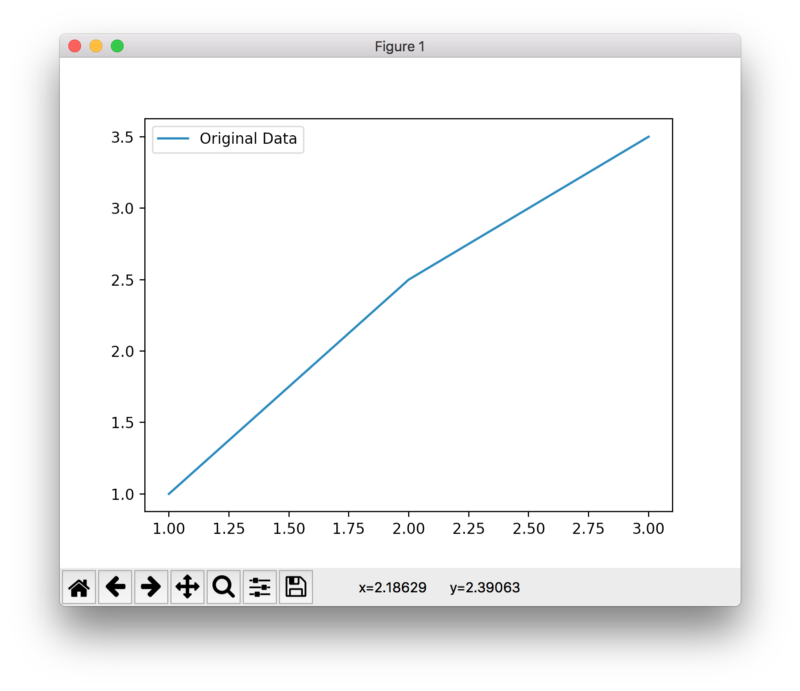
…a function that maps an event or values of one or more variables onto a real number intuitively representing some “cost” associated with the event.

from [Wikipedia](https://en.wikipedia.org/wiki/Loss_function)

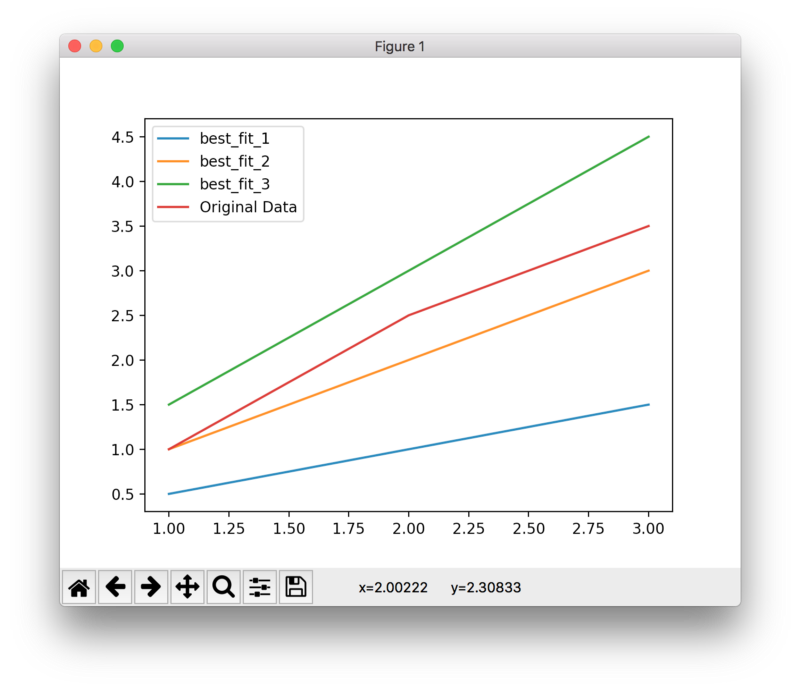
In this situation, the event we are finding the cost of is the difference between estimated values, or the hypothesis and the real values — the actual data we are trying to fit a line to.

To state this more concretely, here is some data and a graph.

╔═══════╦═══════╗  
║ X ║ y ║  
╠═══════╬═══════╣  
║ 1.00 ║ 1.00 ║  
║ 2.00 ║ 2.50 ║  
║ 3.00 ║ 3.50 ║  
╚═══════╩═══════╝



Pretty boring graph. The goal here is to find a line of best fit — a line that approximates the values most accurately. Here are some random guesses:



Some random guesses at a line of best fit

╔═══════╦═══════╦═════════════╦════════════╦════════════╗  
║ X ║ y ║ best\_fit\_1 ║ best\_fit\_2 ║ best\_fit\_3 ║  
╠═══════╬═══════╬═════════════╬════════════╬════════════╣  
║ 1.00 ║ 1.00 ║ 0.50 ║ 1.00 ║ 1.50 ║  
║ 2.00 ║ 2.50 ║ 1.00 ║ 2.00 ║ 3.00 ║  
║ 3.00 ║ 3.50 ║ 1.50 ║ 3.00 ║ 4.00 ║  
╚═══════╩═══════╩═════════════╩════════════╩════════════╝

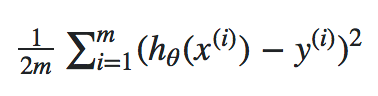
Making that beautiful table was really hard, I wish Medium supported tables. Anyway, we have three hypothesis — three potential sets of data that might represent a line of best fit. The slope for each line is as follows:

* best\_fit\_1: 0.5
* best\_fit\_2: 1.0
* best\_fit\_3: 1.5

best\_fit\_2 looks pretty good , I guess. But we are data scientists, we don’t guess, we conduct analysis and make well founded statements using mathematics.

#### Understanding the Cost Function

Let’s do an analysis using the squared error cost function.

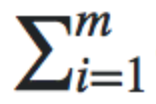


Squared Error cost function

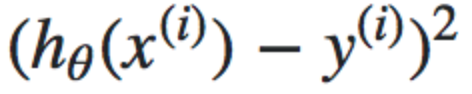
Remember a cost function maps event or values of one or more variables onto a real number. In this case, the event we are finding the cost of is the **difference between estimated values**, or the difference between the hypothesis and the real values — the actual data we are trying to fit a line to.

Let’s unwrap the mess of greek symbols above. On the far left, we have 1/2\*m. m is the number of samples — in this case, we have three samples for X. Those are 1, 2 and 3. So 1/2\*m is a constant. It turns out to be 1/6, or 0.1667 .

Next we have a sigma.



This means the sum. In this case, the sum from i to m, or 1 to 3. We repeat the calculation to the right of the sigma, that is:



for each sample.

The actual calculation is just the hypothesis value for h(x), minus the actual value of y. Then you square whatever you get.

The final result will be a single number. We repeat this process for all the hypothesis, in this case best\_fit\_1 , best\_fit\_2 and best\_fit\_3. Whichever has the lowest result, or the lowest “cost” is the best fit of the three hypothesis.

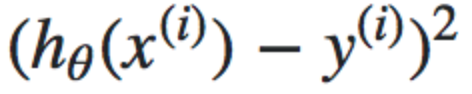
Let’s go ahead and see this in action to get a better intuition for what’s happening.

#### Calculating the Cost Function by Hand

Let’s run through the calculation for best\_fit\_1.

╔═══════╦═══════╦═════════════╗  
║ X ║ y ║ best\_fit\_1 ║  
╠═══════╬═══════╬═════════════╣  
║ 1.00 ║ 1.00 ║ 0.50 ║  
║ 2.00 ║ 2.50 ║ 1.00 ║  
║ 3.00 ║ 3.50 ║ 1.50 ║  
╚═══════╩═══════╩═════════════╝

We know the 1/2m is simply 1/6, so we will focus on the summing the result of:



for best\_fit\_1, where i = 1, or the first sample, the hypothesis is 0.50. This is the h\_theha(x(i)) part, or what we think is the correct value. The actual value for the sample data is 1.00. So we are left with (0.50 — 1.00)^2 , which is 0.25. Let’s add this result to an array called results.

results = [0.25]

The next sample is X = 2. The hypothesis value is 1.00 , and the actual y value is 2.50 . So we get (1.00 — 2.50)^2, which is 2.25. Add it to results.

results = [0.25, 2.25]

Lastly, for X = 3, we get (1.50 — 3.50)^2 , which is 4.00.

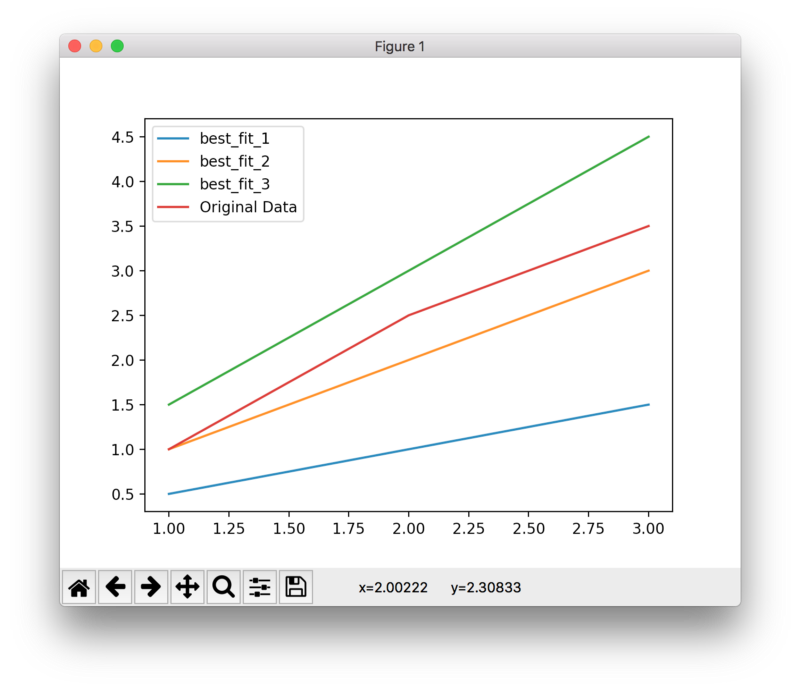
results = [0.25, 2.25, 4.00]

Finally, we add them all up and multiply by 1/6.

6.5 \* (1/6) = 1.083. The cost is 1.083. That’s nice to know, but we need some more costs to compare it to. Go ahead and repeat the same process for best\_fit\_2 and best\_fit\_3. I already did it, and I got:

best\_fit\_1: 1.083  
best\_fit\_2: 0.083  
best\_fit\_3: 0.25

A lowest cost is desirable. A low costs represents a smaller difference. By minimizing the cost, we are finding the best fit. Out of the three hypothesis presented, best\_fit\_2 has the lowest cost. Reviewing the graph again:



best\_fit\_2 has the lowest cost

The orange line, best\_fit\_2, is the best fit of the three. We can see this is likely the case by visual inspection, but now we have a more defined process for confirming our observations.

#### Use with Linear Regression

We just tried three random hypothesis — it is entirely possible another one that we did not try has a lowest cost than best\_fit\_2. This is where [Gradient Descent](https://en.wikipedia.org/wiki/Gradient_descent) (henceforce GD) comes in useful. We can use GD to find the minimized value automatically, without trying a bunch of hypothesis one by one. Using the cost function in in conjunction with GD is called linear regression.

This will be the topic of a future post. For now, I want to focus on implementing the above calculations using Python.

#### Calculating the cost function using Python (#1)

As promised, we perform the above calculations twice with Python. Once using for loops, and once using vectors. Firstly, with for loops. The focus of this article is the cost function, not how to program Python, so the code is intentionally verbose and has lots of comments to explain what’s going on.

Personally, the biggest challenge I am facing is how to take the theoretically knowledge and algorithms I learned in my undergraduate calculus classes (I studied electrical engineering) and turn them into working code.

The way I am breaking this barrier down is by really understanding what is going on when I see a equation on paper, and once I understand it (usually after doing several iterations by hand), it’s lot easier to turn into code. Hopefully this helps other people, too.

cost function calculations using for loops

It is more common to perform the calculations “all at once” by turning the data set and hypothesis into matrices. This process is called vectorization.

#### Calculating the cost function using Python (#2)

It’s a little unintuitive at first, but once you get used to performing calculations with vectors and matrices instead of for loops, your code will be much more concise and efficient. Here is the same calculation implemented with matrices using numpy.

calculating cost function with vectors

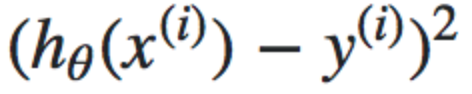
Even without reading the code, it’s a lot more concise and clean.

We are using numpy, and defining X and y as np.array. They provide many more properties for doing vector and matrices multiplication. There are two things to note:

1. We prepend the X vector with a vector 1 s. This is explained well by Andrew Ng [here](https://www.coursera.org/learn/machine-learning/lecture/dpF1j/matrix-matrix-multiplication), in week 1’s “matrix matrix multiplication” lecture. Basically, when you multiply matrices, you need to have the correct dimensions. The vector of 1s allows for this. This is best explained using video, I recommend you watch it on Coursera.
2. The calculation we did in for loops previously is now expressed in two lines:

inner = np.power(((X @ theta.T) - y, 2)

Which is doing this:

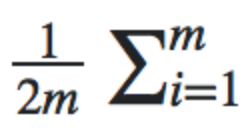


the ‘inner’ calculation

and

np.sum(inner) / 2 \* len(X)

which is doing



Again, I encourage you to sign up for the course (it’s free) and watch the lectures under week 1’s “linear algebra review”. It takes a while to really get a feel for this style of calculation. I went through and put a ton of print statements, and inspected the contents of the array, as well as the array.shape property to really understand what was happening.

#### Conclusion

Researching and writing this really solidified by understanding of cost functions. I hope to write a follow up post explaining how to implement gradient descent, first by hand then using Python.

* [Machine Learning](https://medium.com/tag/machine-learning?source=post)
* [Linear Regression](https://medium.com/tag/linear-regression?source=post)
* [Gradient Descent](https://medium.com/tag/gradient-descent?source=post)
* [AI](https://medium.com/tag/ai?source=post)
* [Mathematics](https://medium.com/tag/mathematics?source=post)