



TASK

Additional Reading – Calculus

Visit our website

Introduction

Calculus is the study of change in mathematical functions. For those of you who remember anything from calculus in high school, you will remember that the derivative of a function gives you the instantaneous gradient of that function. This is the driving force behind the 'learning' part of Machine Learning! Every model used in Machine Learning comes with an error function. Every error function is (ideally) differentiable (i.e. it has a derivative). If we can find the instantaneous gradient of that error function, we can get our code to be able to move down that error function to find the optimal parameters.

WHAT IS A DERIVATIVE?

Say we have $y = f(x)$. In simpler terms, this just means that we have y , which is obtained by applying a function f to a variable x . The gradient of f at exactly point x is given as:

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

So, now what? How does this apply to Machine Learning? What do all of these symbols even mean? Well, let's take a look at a toy example, and see what we discover.

A NOTE ON NOTATION

Function notation

I'm sure you've heard of functions before: we make them all the time in Python! In mathematical notation, it is basically the same concept. A function is defined as something that takes an input and returns an output.

Let's say we want to define a function f that takes an input x and returns double the input. In mathematical notation, we would use something like:

$$f(x) = 2x$$

In Mathematics, $2x$ is just the lazy man's way of saying 2 *multiplied by* x , which just means $2 \times x$.

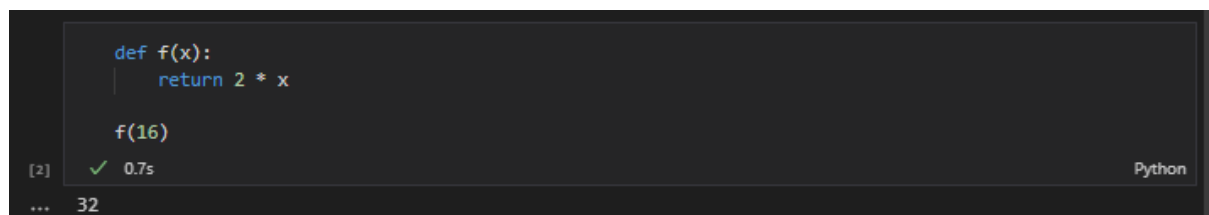
So, using this definition of f , let's find out what double 16 is:

$$f(16) = 2(16)$$

(The brackets here just means that we are multiplying 2 by 16. We use this notation because we are replacing x with 16)

$$f(16) = 32.$$

Let's see what this looks like in Python:

A screenshot of a Python REPL (Jupyter Notebook) with a dark background. The first cell contains the code 'def f(x):' followed by an indented 'return 2 * x'. The second cell contains 'f(16)'. Below the code, the output is shown as '[2] ✓ 0.7s' followed by '32' on the next line. The word 'Python' is visible in the bottom right corner of the interface.

```
def f(x):  
    return 2 * x  
  
f(16)  
[2] ✓ 0.7s  
... 32
```

That simple? Yes, it is!

Limit notation

Limits are an important concept in calculus. Calculus is simply the study of change in Mathematics; limits describe how functions change as a specific variable approaches a specific value.

Let's look at a simple example. Let's define a function:

$$f(x) = \frac{1}{x}$$

Let's say that we want to see what happens to the output of the function as x gets closer and closer to 2:

$$\lim_{x \rightarrow 2} f(x) = f(2) = \frac{1}{(2)} = 0.5$$

This is read as: *the limit of $f(x)$ as x approaches 2 is 0.5*. Another way to say this is *as x gets infinitely closer to 2, $f(x)$ gets infinitely closer to 0.5*.

Well, this is fine, but how is this useful? Why don't we just use $f(2)$?

To answer that, let's look at $f(0)$:

$$f(0) = \frac{1}{(0)}.$$

Well, this isn't good. Anything divided by 0 is undefined: the value **doesn't exist!**

However, if we attach a limit to this, we can mathematically say this:

$$\lim_{x \rightarrow 0} f(x) = \lim_{x \rightarrow 0} \frac{1}{x}$$

This is where we start using our words to reason things out. We know that we are looking at values as x approaches 0. So let's look at this:

x	$f(x)$
2	0.5
1	1
0.5	2
0.1	10
0.0001	10 000

Well, it looks like, as x approaches 0, $f(x)$ gets larger. So how large does $f(x)$ get when x gets infinitely close to 0? Infinitely large! Yes, so you are correct in saying that it is infinity!

$$\lim_{x \rightarrow 0} f(x) = \lim_{x \rightarrow 0} \frac{1}{x} = \infty$$

Okay, but infinity doesn't help us at all! We can't even program that into our computers!

Now, let's get more complicated:

$$f(x) = \lim_{x \rightarrow 0} x \frac{x}{x}$$

Well, the denominator would evaluate to 0, and the numerator would evaluate to 0.

So, what does $\frac{0}{0}$ evaluate to? In limits, everything divided by 0 would give infinity, but 0 divided by everything would give 0. So is it infinitely large, or is it 0? It is neither!

Those eagle-eyed readers will notice one other thing: $\frac{x}{x}$ is always 1! But how can we prove this is the correct answer? Same as earlier, let's inspect these values:

x	$f(x)$
2	1
1	1
0.5	1
0.1	1
0.0001	1

So it looks like it remains the same as x gets infinitely closer to 0.

FIRST-ORDER DERIVATIVES

Calculus absolutely doesn't end here. Remember our equation for derivation? No? It is the first one in this document:

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

This formula gives you the instant gradient of any function f at any point x . This is just denoted as $\frac{df}{dx}$. This can be read as *the gradient of f with respect to x* . In other words, given a specific x , what is the instantaneous gradient of the function f ?

This is what we call the first-order derivative. For now, we will just call it the derivative, as you don't need to know anything more than this.

For a full explanation of this equation, you can refer to this useful video by [Khan Academy](#). They do a great job of explaining it. Nonetheless, here is a summary here below.

Let's look at the numerator. As h approaches 0, $f(x+h) - f(x)$ becomes 0. The same applies for the denominator. This gives us one of those classic $\frac{0}{0}$ situations, which means that there should be something useful here!

Next, let's look at the formula for gradients. Because we denote y as $f(x)$ (because y is the output of this function), we will use this to denote our y value. To calculate gradient, we need two points. Let's say we have any point x , which gives us $f(x)$ when we put it into the function. Now, we define another point $(x+h)$, where we just add a value h onto our x value. This will give us $f(x+h)$. The gradient of these two points, according to the [equation for gradient](#), is:

$$m = \frac{f(x+h) - f(x)}{(x+h) - x} = \frac{f(x+h) - f(x)}{h}$$

Because we want to find the instantaneous gradient, this means that the difference in x values (which is h) must get smaller and smaller, until it is infinitely small. Hence, we have the equation:

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

The derivative of a function gives another function. This derivative function, $\frac{df}{dx}$ gives the instantaneous gradient of f at the value of x . There are a lot of rules when it comes to differentiation, but this is beyond the scope of this lesson. If you are interested in learning some of these rules, you can begin [here](#).

SUMMATION

Now, what about this weird Greek symbol Σ ? This is the capital letter Sigma, and is the eighteenth letter of the Greek alphabet, which makes it an 's' in the English alphabet.

But we are not here to learn Greek, are we? What does this 's' stand for? It stands for **sum**, of course!

Let's look at an example. Let's say we have a set of values $x = \{5, 10, 3\}$. Then, we say

$$\sum_{i=0}^n x_i = 18$$

where n denotes the number of items in x . This is just all values added up. The bottom of the \sum tells us two things: we have an index i , which we begin with a value of 0. Essentially, the bottom part tells us how to start the summation.

The top part of \sum tells us where to stop. In this case, we stop when $i = n$.

If we say

$$\sum_{i=0}^n x_i + 2$$

This gives us the value 24. This is because we just add 2 to each element in x .

Like all good Maths, this can be translated directly to code. Don't believe me? Take a look:

```

# Our list of values
x = [5, 10, 3]

# Sum of all values
sum_total = 0
n = len(x) # we denote n as the number of elements in x

```

✓ 0.7s Python

$$\sum_{i=0}^n x_i$$

```

for i in range(n):
    sum_total += x[i]

print(f"Our sum total is {sum_total}")

```

✓ 0.6s Python

Our sum total is 18

$$\sum_{i=0}^n x_i + 2$$

```

sum_total = 0 # Reset our sum_total back to 0
for i in range(n):
    sum_total += x[i] + 2

print(f"Our sum total is {sum_total}")

```

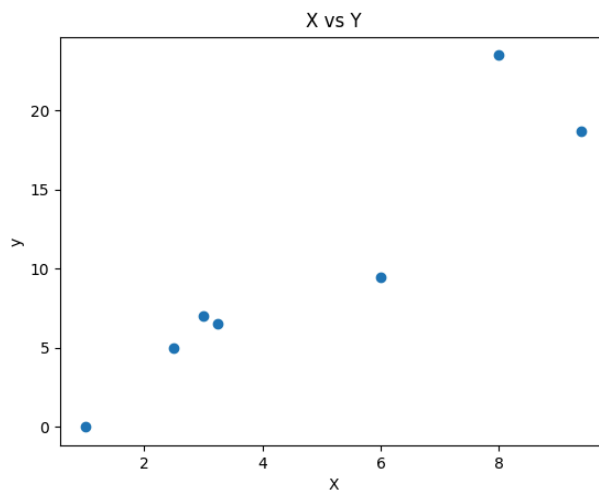
✓ 0.4s Python

Our sum total is 24

Use this to your advantage!

A simple line

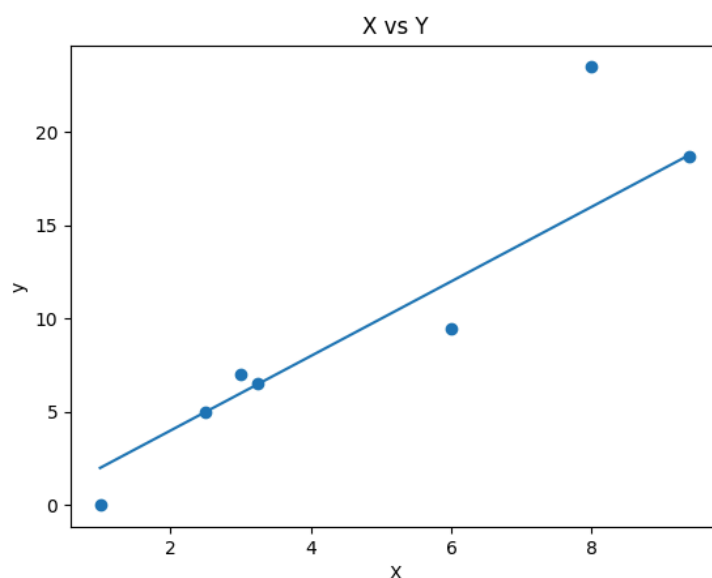
Let's look at a practical example. Let's say we are given the following data:



We can kind of guess that we can model the data with a line of best fit. For the sake of simplicity, we will just assume that this line starts at the origin (i.e. at $x = 0$ and $y = 0$). The [formula for this would be](#):

$$f(x) = mx$$

where f is our model, x is our input, and m is our model parameter. The idea is that we need to find an m that will give us a model f that, when given an input x , will give us a good prediction of the data. This will probably look something like this:



THE ERROR FUNCTION

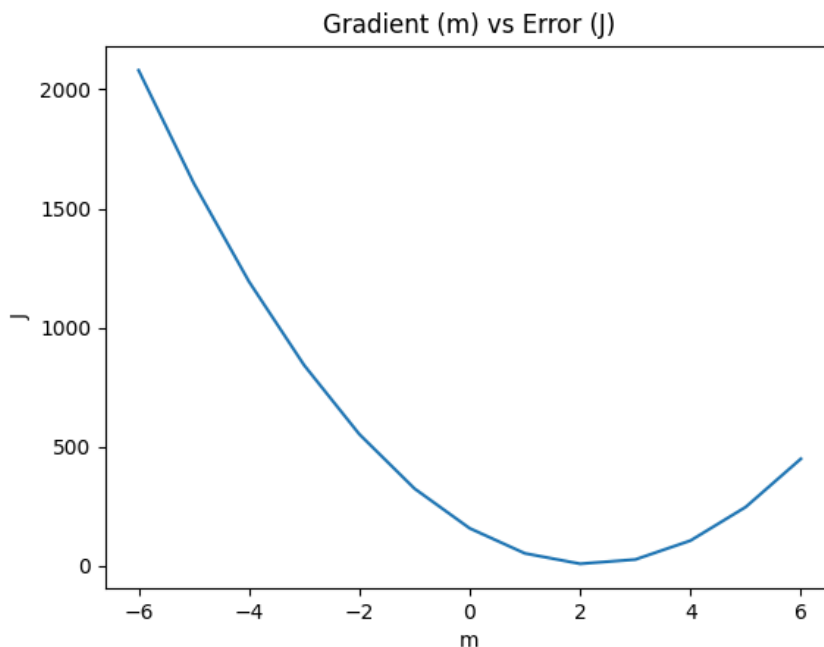
So, how do we do this? How does that complicated derivative factor in? Well, as we mentioned earlier, if we can find the gradient of the error function, then we will know how to go down that gradient. We know how to find the gradient of any function. So now, we just need to find the error function of this.

For this type of data, we will use the [Mean Squared Error](#) (MSE) function. Those of you who still remember will know that the formula for this is:

$$J(m) = \frac{\sum_i (y_i - f(x_i; m))^2}{n}$$

Well, darn, now what does all of this mean? Let's start with the numerator. In English, this translates to "the sum over the index of i in the data, of y_i minus $f(x_i)$ (a.k.a our model's prediction of y) given a specific m , squared". The denominator, n , just refers to the number of data points.

So, we denote our error as J . That means that now we have the error function we were looking for. Let's see what it looks like. Firstly, we must be mindful that $J(m)$ depends on m , which makes J the dependent variable, and m the independent variable.



Now, what does this mean? Well, it looks like there is a lowest point near $m = 2$. This lowest point represents the lowest value of J , which represents our error. Mathematically, this means that 2 is our target. But how will the computer know this?

Machine Learning models are often computationally expensive, so it's not like we will be able to compute the error for every value of m . This is where instantaneous gradient comes in.

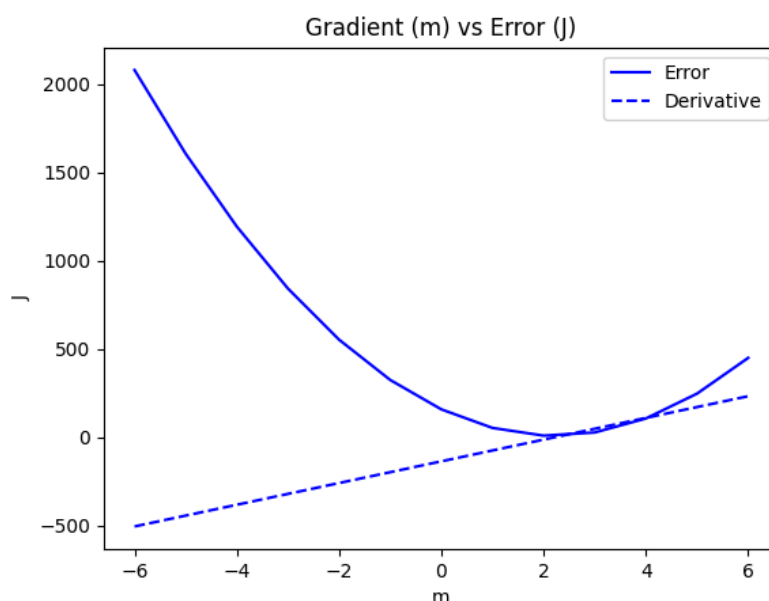
Imagine a ball on this curve, starting at $m = -6$. This ball, affected by gravity, will naturally roll downhill. Downhill is just a fancy way of saying that it will change its value of m to tend towards the lowest value of J . This is, naturally, affected by the instantaneous gradient of the curve underneath the ball. Let's take a dive in to understand how this works.

Calculus to the rescue

I am going to spare you the details of how to do this derivation – if you want a more in-depth explanation, feel free to check out [this great guide on how to derive the error function](#). The derivation of the error function is as follows:

$$\frac{dJ}{dm} = \frac{\sum_i -2x_i(y_i - f(x_i; m))}{n}$$

Let's see what this looks like:



We can see that the derivative of the error function is 0 near the lowest point of the error function. This means that, at this point, the error function goes from having a

negative gradient (downhill) to a positive gradient (uphill). The ball, always searching for the lowest point, will always settle when the gradient is 0.

Rolling the ball downhill

Now, all we have to do to roll our ball downhill is to find a mathematical way of saying “change m to go closer to the point at which the derivative of the error function is 0”. Simple, right? Maybe not immediately. Let’s look at how we descend the gradient.

GRADIENT DESCENT

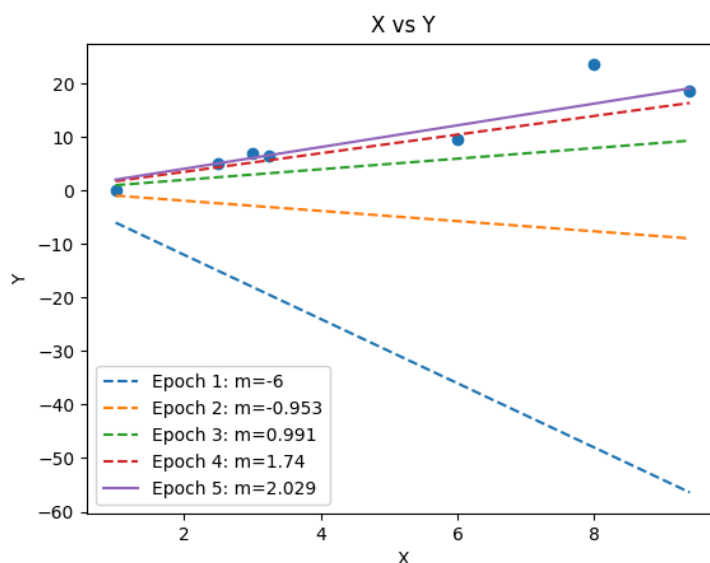
Gradient descent is the process of rolling the ball down the hill. The [formula for gradient descent](#) is:

$$m' = m - \alpha \left(\frac{dJ}{dm} \right)$$

In English, this just means “the current gradient minus alpha times the derivative of the error function”. Wait, no one mentioned an alpha!? No need to worry, this is just the symbol we use to define something called the “learning rate”.

Think of it this way: each time you update m , your ball rolls down at a certain rate. You don’t need to worry too much about the learning rate for now; we are just interested in calculus!

Let’s see how this gradient update looks:



Fantastic! We have a working solution! A note on the naming: we refer to each time the model goes through the data and updates the gradient as an epoch. You can see the gradient of the line getting consistently better with each epoch, and you can also see the difference between m values getting smaller between epochs. Think of it this way: the ball starts at a steep gradient, and so rolls down the hill faster. As it reaches the bottom of the hill, the gradient of that hill gets more gentle, so the ball slows down.



Rate us
Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

[Click here](#) to share your thoughts anonymously.



REFERENCE

IEEE. (1993). *IEEE Standards Collection: Software Engineering*. IEEE Standard 610.12-1990.