NEURAL NETWORK PACKAGE MODULE 4

THE CALCULUS OF NEURAL NETWORKS

QMIND EDUCATE

ANDREW FARLEY

TABLE OF CONTENTS

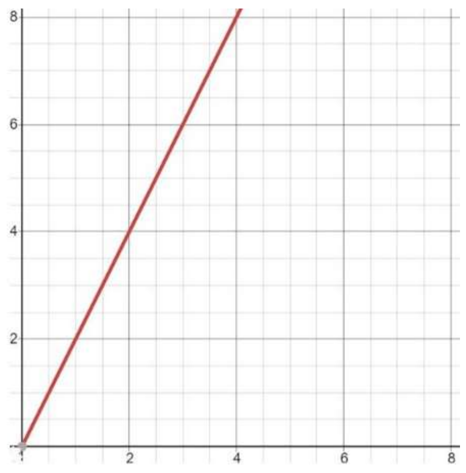
WHAT IS THIS MODULE ABOUT?

As of the end of module 3, we are done talking about theory. You should hopefully have a basic idea of what a Neural Network is, how it's structured, and how it can be used and trained to make accurate predictions. Something that hasn't been discussed too much is the calculus behind how these networks operate. This module aims to provide you with a background of how derivatives, gradients, and optimization work, all being aspects of calculus. If you are already comfortable with these topics, feel free to skip ahead to the next module where you will begin creation of a Neural Network from scratch in Python. If not, read through this module and reference it as a resource. It is crucial that you understand the mathematical theory of neural networks or building your own Neural Network library will prove to be extremely difficult. Once again, I will be referring to 3blue1brown as a video resource. He has created an amazing video series called The Essence of Calculus which covers a huge variety of topics in calculus. However, this series isn't focused on Neural Networks and has a bunch of extra information that isn't needed when dealing with Neural Networks. I would advise you to sift through the headings as you read this module and watch videos that pertain to what we are doing.

WHAT IS CALCULUS?

The cliché definition of calculus in one sentence is "the study of the rate of change, as well as the area under, curves". Let's pick this sentence apart. There are two major phrases here that should stand out: "the rate of change" and "the area under". "The rate of change" is referring to what we call derivatives which have been discussed throughout the last three modules and were mentioned heavily in module three.

Derivatives



When you have a mathematical curve, it will exist in some sort of space often known as a **vector space**. A vector space is any imagined space (of any dimension) where points can live, the x-y plane being a common example. Let's take the example of a person walking on a two-dimensional flatland, which you can imagine as the x-y plane. If they walk along this plane in a certain way, we can model their position with an equation, let's use y = 2x. This equation means that for every 1 unit the person walks in the x direction, they will have walked 2 units in the y direction. With lines like this, we call the **slope** of the line 2, because that is how many units y will change when x changes by 1. Slope is a ratio, which means it can be calculated at any two points on a line. The points p1 = (1, 2) and p2 = (4, 8) both lie on our line above. We would expect the change in the x (which we can call dx) and the change in the y (which we can call dy) to follow a similar property to our equation (y = 2x). Computing these changes for x and y:

$$dx = p2.x - p1.x = 4 - 1 = 3$$

$$dy = p2.y - p1.y = 8 - 2 = 6$$

If we compute the ratio of these:

$$\frac{dy}{dx} = \frac{6}{3} = 2$$

Which is equal to our slope! This will be true for all points as long as our curve is a line. The slope can also be referred to as the rate of change because it is computed using the difference (or change) of values. This slope is the **derivative** of our line.

But what about curves that are not lines? How will we compute the slope? Does a slope even exist? In short, the derivative is the more general version of slope. Every continuous curve (a curve with no immediate breaks or edges) has a derivative, but only lines have slope. There are many rules for different kinds of functions, all derived from the basic principle that a derivative is the rate of change of a function at a given value. Think of this as how fast the function is moving at that value. If you think back to our example above, the derivative across the entire function is just 2, because the slope is 2. This also must be the speed the person is moving relative to the x-axis!

If we rethink this example so the person is walking in a straight line, this would make a lot of sense. We know the formula for speed is just distance over time. Let our distance be the y value and the time over which movement is happening be the x value. So, for the curve y = 2x, our person must be moving 2 units every second because the value of y (or distance) increases by 2 every time x (or time) increases by 1. Our formula for this curve is very analogous to the formula for speed.

$$This\ Curve: y = 2x$$

$$Speed\ Formula:\ \ speed = \frac{distance}{time}$$

$$Rearranged: \quad distance = speed * time$$

We know that for this example the y is distance and x is time, therefore the slope must be speed.

While it isn't practical to directly relate real-world examples to derivatives of different kinds of functions, the idea that the derivative of a function at a point is the speed it is moving will hold true.

But how do we calculate the derivative of crazy functions that aren't necessarily a straight line? Mathematicians have developed some handy rules which will allow us to do this much easier. We will use these in many different ways, a good example being the derivative of the sigmoid activation function. Here are some examples of the different rules for differentiation:

Power Rule:

$$\frac{d}{dx}(x^n) = n * x^{n-1}$$

Chain Rule:

$$\frac{d}{dx}\left(\left(f(x)\right)^n\right) = n * f(x)^{n-1} * f'(x)$$

Product Rule:

$$\frac{d}{dx}\left(f(x) * g(x)\right) = f'(x) * g(x) + f(x) * g'(x)$$

Quotient Rule:

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x) * g(x) - f(x) * g'(x)}{\left(g(x)\right)^2}$$

Where f'(x) is the derivative of f(x).

Integrals

An **integral** is classically defined as "the area under a curve". Integrals aren't really used by Neural Networks, so I am not going to spend too much time on them. An integral is essentially the inverse of a derivative. Instead of looking at the rate of change of a curve, you are looking at how much "change" that curve has covered, or (in two dimensions) the area between it and the x-axis. For example, the integral of y = 2 would be y = 2x +C (pretty much the same as the example function we used in the derivatives section). The reason we have this constant 'C' at the end is because when we differentiate, any constants in the equation are eliminated. So when we do the inverse of differentiating (taking the integral) we have the potential for there to be a constant. This constant will remain unknown unless we know something about the new function we have created through integration.

GRADIENTS

What if your function depends on more than one variable? Can you still take the derivative? In short, yes, but it is a little different. What you would do is take something called the partial derivative of each variable. This partial integral will give you the rate of change in a certain direction (the direction one variable would increase or decrease). It is impossible to take the derivative in every single direction which is why we break it down to each individual variable. Let’s say you had the function:

$$f(x, y, z) = x^2y + xyz + z^2$$

Instead of finding the ‘pure derivative’ of f (which is not possible in this case), we find something called the gradient which was discussed briefly in module three. This is crucial for Neural Networks as they will allow us to step the values of our weights closer and closer to providing errorless outputs when computing an input to the network. This gradient is a vector where each component is the **partial derivative** with respect to a specific variable. Therefore, the size of the vector must be equal to the number of variables. The gradient of the function above would have three values. When we take a partial derivative, we simply derive the function with respect to one variable and treat the others like constants. For example:

$$Partial\ Derivative\ of\ f\ with\ respect\ to\ x = 2xy + yz$$

Or, with proper notation:

$$\frac{\partial f}{\partial x} = 2xy + yz$$

Notice how the z squared term is completely eliminated because it is a constant in the realm of x and how with the other terms the variables other than x are left alone. In this function, each variable is completely independent of the others. This means that if x changes then it will have no impact on y or z. When we take a derivate of something independent (like a constant) with respect to a certain variable, it will be eliminated if that certain variable is not in the term. What we are doing when taking the partial derivative is isolating a certain variable in the function and looking at how the rest of the function would change when it changes.

The gradient of the above function (symbolized with ∇) would be:

$$\nabla f = < \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} >$$

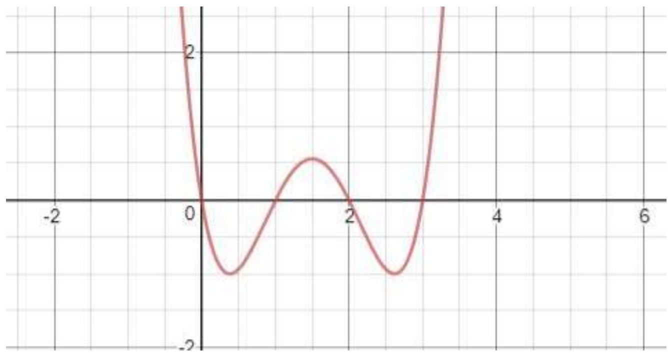
The variable denoted in the denominator of each term above is the 'certain variable' I just mentioned a couple sentences ago. Remember that it will be the only variable we care about when taking the derivative. This will evaluate to:

$$\nabla f = < 2xy + yz,\ \ x^2 + xz,\ \ xy + 2z\ >$$

This gradient represents how our function is changing in three-dimensional space. But why do we care? Why is the gradient a useful tool when it comes to neural networks? Before we answer that, we are going to need to cover one last, key, topic of calculus.

OPTIMIZATION

The derivative is a powerful tool. Being able to know how much a function is changing at any given point provides us insight into how the function operates. It can also provide us with interesting points on the curve, most importantly for us the local minimums and maximums. What are these? Let's take a look at the graph of y = (x)(x-1)(x-2)(x-3):



You see the peaks and valleys? The exact point where the function stops going up and starts going down is called a **local maximum** and the exact point where the function stops going down and starts going up is called a **local minimum**. There appears to be one *local maximum* in this graph (at around x = 1.5) and 2 *local minimums* (at around x = 0.4 and x = 2.6). Something else to notice is that these **local** minimums are at the very bottom of the function, no other x value will give a y value that is lower, and consequently we call these **global minimums**. This function has no global maximum because it tends off to infinity in the positive direction, meaning that the one peak at around x =1.5 is not the highest point in the function.

So, how do we calculate the exact x value for when the function is at these local minima and maxima? Remember that the derivative is the speed of the function at any point. If it is positive, then the function is moving **up** the y-axis. If it is negative, then the function is moving **down** on the y-axis. But if the derivative is zeroit is either at a local minimum or maximum. The derivative being zero means we aren't moving in any direction on the y-axis which means we must be at a peak or a valley. To find the x values of these peaks and valleys we just take the derivative, set it to equal zero, and solve for the x values that satisfy the equation.

Let's start with finding the derivative (y' is the derivative).

$$y' = 4x^3 - 18x^2 + 22x - 6 = 0$$

By setting it to zero, we can solve for the exact x values of our local minima and maxima.

$$x = \frac{3}{2}, \frac{3 \pm \sqrt{5}}{2} \cong 1.50, 0.38, 2.62$$

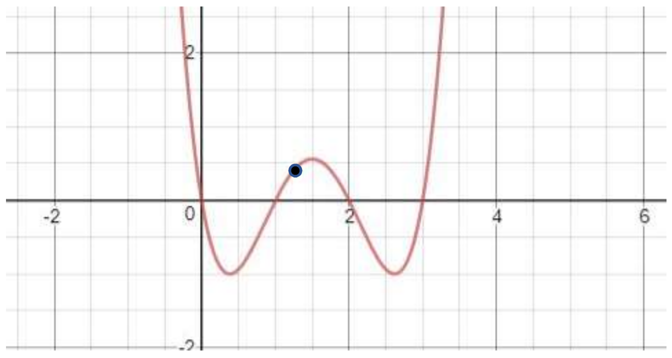
Therefore, when x is one of these values (1,50, 0.38, and 2.62), our function is at a local minimum or maximum (which we can check by looking at our guesses above).

Why do we care? How will this help us with neural networks? Think back to the first module when we related a neural network to being a massive, multi-variable, function. If we think about it this way, we can just find the global minimum of the **cost** function which should reduce error to be as low as theoretically possible!

The problem is, there are way too many variables in our Neural Network for us to simply take the derivative of the cost function. Instead of having the situation above, where we have a one variable function, we have millions, if not billions, of them. This means we are trying to optimize a function in millions if not billions of **dimensions**. Let that sink in for a second… We live in a world with 3 dimensions, it is quite vast if we consider the entire universe. Just try to imagine a function in 4 dimensions, it is incredibly difficult if not impossible. Now think about adding another million possible dimensions on top of that. This is way too much to compute and wouldn't make sense to do so- even the best computers today would have trouble trying to optimize a function like that. Instead, we rely on our friend the *gradient*.

GRADIENT DESCENT

You already know the basics of gradient descent from last module. You should, by now, be able to express what it does and how it operates. We are now going to focus on how gradient descent relates to *optimization*. The entire idea of gradient descent is to find a change in the variables to **step towards** a local minimum instead of finding its exact location. Remember that the gradient of a function is a vector, it gives us the rate of change (or speed) of each variable. If we evaluate this gradient at our current location, it will give us the change in every variable needed to take a small step towards a minimum. We can use the graph above as a one variable example. Let's say the starting point was here:



Because x is the only variable in the function, our gradient will be a vector of size 1, the change x needs to make. So we take the partial derivative with respect to x:

$$\nabla \mathrm{y} = < \frac{\partial \mathrm{y}}{\partial \mathrm{x}} > = < 4x^3 - 18x^2 + 22x - 6 >$$

At our current location, x is equal to 1.25; we evaluate the gradient at 1.25:

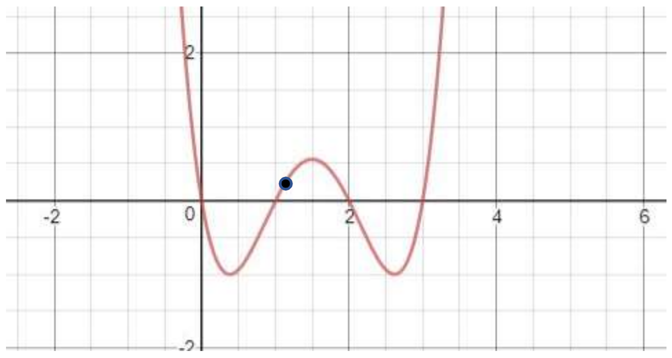
$$\nabla \mathrm{y}(1.25) = < 1.1875 >$$

You will notice here that the magnitude (or size) of the gradient is quite large, we would be taking a massive step if we simply subtracted the gradient from our current x value. This is why we have the learning rate, talked about last module. Let's take the learning rate to be 0.05.

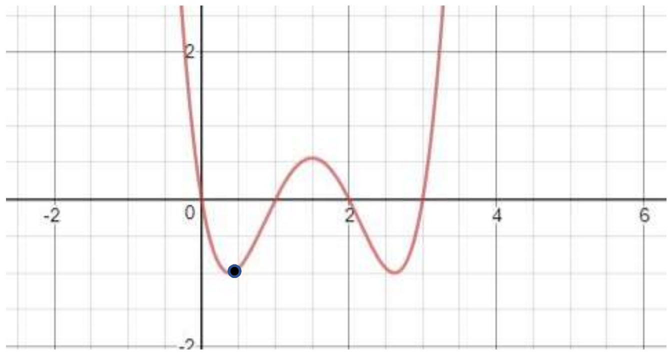
$$\Delta x = \ -Learning\ Rate * Gradient$$

$$\Delta x = \ -0.05 * 1.1875 = \ -0.059375$$

$$Next\ x = x + \Delta x = 1.19$$



As you can see, we have stepped towards the local minimum. If we repeat this process enough, we will eventually be a negligible x distance away from the minimum.



If this was a cost function, we would have minimized the error between what the network **will** produce and what it is **supposed** to produce, giving us a very accurate neural network.

This is how gradient descent works, except for gradient descent may be extended to many, many, many dimensions. Regardless of the many dimensions, we are simply optimizing a cost function using calculus! Neural Networks are not magic, they are the result of different aspects of calculus. All we are doing is trying to minimize the possible error that can occur. Just remember that the derivative gives us the change we need in our variable, both with magnitude and direction to minimize error. This is what Neural Networks are based on and is how they operate. Once reading this you should understand the core mathematical principles that Neural Nets are built on. Get ready, as you will soon be putting it all to practice.

REFERENCES

[1] G. Sanderson, "YouTube," 3blue1brown, 28 April 2017. [Online]. Available: https://www.youtube.com/playlist?list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr. [Accessed 1 April 2019].