



Search Medium

Write



Get unlimited access to all of Medium. [Become a member](#)



Calculating Gradient Descent Manually

Part 4 of Step by Step: The Math Behind Neural Networks



Chi-Feng Wang · [Follow](#)

Published in Towards Data Science · 9 min read · Oct 24, 2018

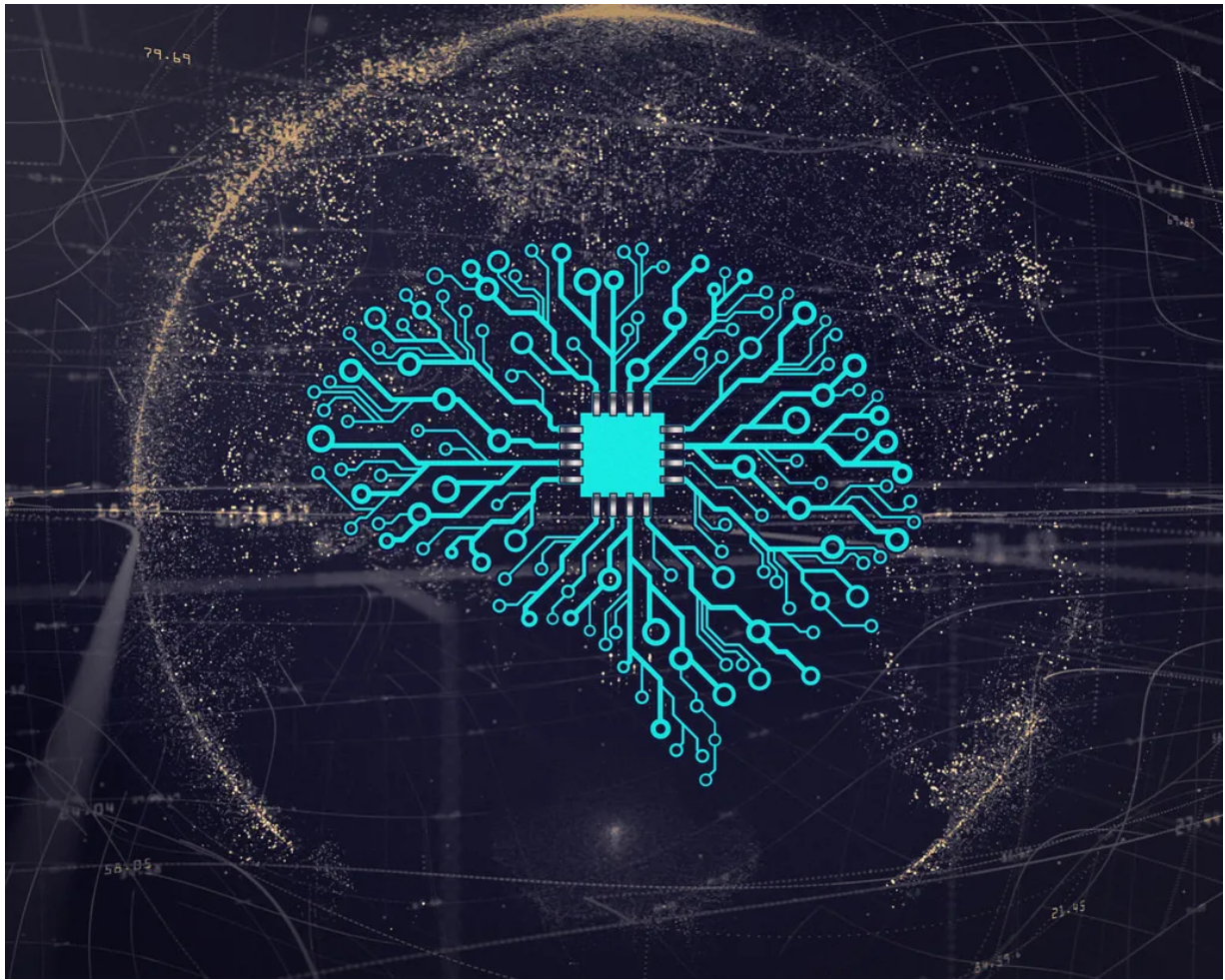


631



6





Title image: [Source](#)

Here's our problem. We have a neural network with just one layer (for simplicity's sake) and a loss function. That one layer is a simple fully-connected layer with only one neuron, numerous weights w_1, w_2, w_3, \dots , a bias b , and a ReLU activation. Our loss function is the commonly used Mean Squared Error (MSE). Knowing our network and our loss function, how can we tweak the weights and biases to minimize the loss?

In Part 1, we learned that we have to find the slope to our loss (or cost) function in order to minimize it. We found that our cost function is:

$$C(\mathbf{y}, \mathbf{w}, \mathbf{X}, b) = \frac{1}{N} \sum_{i=1}^N (y_i - \max(0, \mathbf{w} \cdot \mathbf{X}_i + b))^2$$

Image 1: Cost function

In Part 2, we learned how to find the partial derivative. This is important because there are more than one parameter (variable) in this function that we can tweak. We need to find the derivative of the cost function with respect to both the weights and biases, and partial derivatives come into play.

In Part 3, we learned how to find derivatives of vector equations. Both the weights and biases in our cost function are vectors, so it is essential to learn how to compute the derivative of functions involving vectors.

Now, we finally have all the tools we need to find the derivative (slope) of our cost function!

Gradient of A Neuron

We need to approach this problem step by step. Let's first find the gradient of a single neuron with respect to the weights and biases.

The function of our neuron (complete with an activation) is:

$$neuron(\mathbf{x}) = \max(0, \mathbf{w} \cdot \mathbf{x} + b)$$

Image 2: Our neuron function

Where it takes \mathbf{x} as an input, multiplies it with weight \mathbf{w} , and adds a bias b .

This function is really a composition of other functions. If we let $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, and $g(x) = \max(0, x)$, then our function is $neuron(\mathbf{x}) = g(f(\mathbf{x}))$. We can use the vector chain rule to find the derivative of this composition of functions!

The derivative of our neuron is simply:

Image 3: Derivative of our neuron function by the vector chain rule

Where $z = f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$.

There are two parts to this derivative: the partial of z with respect to \mathbf{w} ,

and the partial of $neuron(z)$ with respect to z .

What is the partial derivative of z with respect to w ?

There are two parts to z : $w \cdot x$ and $+b$. Let's look at $w \cdot x$ first.

$w \cdot x$, or the dot product, is really just a summation of the element-wise multiplication of every element in the vector. In other words:

Image 4: Expanded version of $w \cdot x$, or the dot product

This is once again a composition of functions, so we can write $v = w \otimes x$ and $u = \text{sum}(v)$. We're trying to find the derivative of u with respect to w . We've learned about both of these functions — element-wise multiplication and summation — before in [Part 3](#). Their derivatives are:

Image 5: Derivative of u with respect to v and derivative of v with respect to w ; where $u = \text{sum}(w \otimes x)$

(Go back and review them if you don't remember how they're derived)

Therefore, by the vector chain rule:

Image 6: Derivative of u with respect to w

That's it! Now, let's find the derivative of $z = u + b$ where $u = \mathbf{w} \cdot \mathbf{x}$ with respect to both the weights \mathbf{w} and the bias b . Remember that the derivative of a function with respect to a variable not in that function is zero, so:

Image 7: Derivative of z with respect to the weights and biases, where $z = \text{sum}(\mathbf{w} \otimes \mathbf{x}) + b$

That's it! Those two are the derivatives of u with respect to both the weights and biases.

What is the partial derivative of neuron(z) with respect to z?

$$\text{Neuron}(z) = \max(0, z) = \max(0, \text{sum}(\mathbf{w} \otimes \mathbf{x}) + b).$$

The $\max(0, z)$ function simply treats all negative values as 0. The graph would thus look something like this:

Image 8: $\text{Max}(0,z)$ // [Source](#)

Looking at that graph, we can immediately see that the derivative is a piecewise function: it's 0 for all values of z less than or equal to 0, and 1 for all values of z greater than 0, or:

Image 9: Derivative of $\text{max}(0,z)$

Now that we have both parts, we can multiply them together to get the derivative of our neuron:

Image 10: Derivative with respect to the weights of our neuron: $\max(0, \sum(w \otimes x) + b)$

And substitute $z = w \cdot x + b$ back in:

Image 11: Substituting $w \cdot x + b$ for z

Voila! We have our derivative for a neuron with respect to its weights!
Similarly, we can use the same steps for the bias:

Image 12: Derivative of our neuron with respect to bias

And there you go! We now have the gradient of a neuron in our neural network!

Gradient of Loss Function

Our loss function, defined in Part 1, is:

Image 13: Loss Function

We can immediately identify this as a composition of functions, which require the chain rule. We'll define our intermediate variables as:

Image 14: Intermediate variables for loss function

*Note, the u and v here are different from the u and v used in the previous section.

Let's compute the gradient with respect to the weights w first.

Gradient With Respect to Weights

u is simply our neuron function, which we solved earlier. Therefore:

Image 15: Derivative of $u = \max(0, \sum(w \otimes x) + b)$ with respect to the weights

$v(y,u)$ is simply $y-u$. Therefore, we can find its derivative (with respect to w) using the distributive property and substituting in the derivative of u :

Image 16: Derivative of $v=y-u$ with respect to the weights

Finally, we need to find the derivative of the whole cost function with respect to w . Using the chain rule, we know that:

Image 17: Derivative of cost function

Let's find the first part of that equation, the partial of $C(v)$ with respect to v first:

Image 18: Derivative of cost function with respect to v

From above (Image 16), we know the derivative of v with respect to w . To find the partial of $C(v)$, we multiply the two derivatives together:

Image 19: Derivative of cost function with respect to w

Now, substitute $y-u$ for v , and $\max(0, \mathbf{w} \cdot \mathbf{x} + b)$ for u :

Image 20: Derivative of cost function with respect to w

Since the \max function is on the second line of our piecewise function, where $\mathbf{w} \cdot \mathbf{x} + b$ is greater than 0, the \max function will always simply output the value of $\mathbf{w} \cdot \mathbf{x} + b$:

Image 21: Derivative of cost function with respect to w

Finally, we can move the summation inside our piecewise function and tidy it up a little:

Image 22: Derivative of cost function with respect to w

That's it! We have our derivative with respect to the weights! However, what does this mean?

$w \cdot x + b - y$ can be interpreted as an error term — the different between the predicted output of the neural network and the actual output. If we call this error term e_i , our final derivative is:

Image 23: Derivative of cost function with respect to w represented with an error term

Here, the greater the error, the higher the derivative. In other words, the derivative represents the slope, or how much we have to move our weights by in order to minimize our error. If our neural network has just begun training, and has a very low accuracy, the error will be high and thus the derivative will be large as well. Therefore, we will have to take a big step in order to minimize our error.

You might notice that this gradient is pointing in the direction of higher

cost, meaning we cannot add the gradient to our current weights — that will only increase the error and take us a step away from the local minimum. Therefore, we must subtract our current weights with the derivative in order to get one step closer to minimizing our loss function:

Image 24: Gradient descent function

Here, η represents the learning rate, which we as programmers can set. The larger the learning rate, the bigger the step. However, setting a too-large learning rate may result in taking too big a step and spiraling out of the local minimum. For more information, check out [this article on gradient descent](#) and [this article on setting learning rates](#).

Gradient With Respect to Bias

Once again, we have our intermediate variables:

Image 25: Intermediate variables for loss function

We also have the value of the derivative of u with respect to the bias that we calculated previously:

Image 26: Derivative of u with respect to the bias.

Similarly, we can find the derivative of v with respect to b using the distributive property and substituting in the derivative of u :

Image 27: Derivative of v with respect to the bias

Again, we can use the vector chain rule to find the derivative of C :

Image 28: Derivative of cost function with respect to the bias

The derivative of C with respect to v is identical to the one we calculated for the weights:

Image 29: Derivative of cost function with respect to v

Multiplying the two together to find the derivative of C with respect to b , and substituting in $y-u$ for v , and $\max(0, \mathbf{w} \cdot \mathbf{x} + b)$ for u , we get:

Image 30: Derivative of cost function with respect to the bias

Once again, because the second line explicitly states that $\mathbf{w} \cdot \mathbf{x} + b > 0$, the \max function will always simply be the value of $\mathbf{w} \cdot \mathbf{x} + b$.

Image 31: Derivative of cost function with respect to the bias

Just like before, we can substitute in an error term, $e = \mathbf{w} \cdot \mathbf{x} + b - y$:

Just like the derivative with respect to the weights, the magnitude of this gradient is also proportional to the error: the bigger the error, the larger step towards the local minimum we have to take. It is also pointing towards the direction of higher cost, meaning that we have to subtract the gradient from our current value to get one step closer to the local minimum:

Image 33: Gradient descent function for the bias

Congratulations on finishing this article! This was most likely not an easy read, but you've persisted until the end and have succeeded in doing gradient descent manually!

As I've said in [Part 1](#) of this series, without understanding the underlying math and calculations behind each line of code, we cannot truly understand what "creating a neural network" really means or appreciate the complex intricacies that support each function that we write.

I hope that these equations and my explanations make sense and have helped you understand these calculations better. If you have any questions or suggestions, don't hesitate to leave a comment below!

If you haven't already, read Parts 1, 2, and 3 here:

- [Part 1: Introduction](#)
- [Part 2: Partial Derivatives](#)
- [Part 3: Vector Calculus](#)

Download the original paper [here](#).

If you like this article, don't forget to leave some claps! Do leave a comment below if you have any questions or suggestions :)

Artificial Intelligence

Neural Networks

Calculus

Gradient Descent

Math



Written by Chi-Feng Wang

1.5K Followers · Writer for Towards Data Science

Student at UC Berkeley; Machine Learning Enthusiast

Follow



 Chi-Feng Wang in Towards Data Science

The Vanishing Gradient Problem

The Problem, Its Causes, Its Significance, and Its Solutions


3 min read · Jan 8, 2019



2.7K

 10



 Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with...

And how you can do the same with your docs

15 min read · Apr 25



3.1K

 40



 Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner's Guide to Building...

A LangChain tutorial to build anything with large language models in Python

★ · 12 min read · Apr 25



2.2K

 18



 Chi-Feng Wang in Towards Data Science

A Basic Introduction to Separable Convolutions

Explaining spatial separable convolutions, depthwise separable convolutions, and th...

8 min read · Aug 14, 2018



7.1K

 44



See all from Chi-Feng Wang

See all from Towards Data Science

Recommended from Medium

 Peter Kar... in Artificial Intelligence in Plain Eng...

Linear Regression in depth

The directive equation of a straight line, simple linear regression, math, cost...

★ · 6 min read · Jan 27



111



2



 Sayef

Logistic Regression with Gradient Descent and...

Learn how to implement logistic regression with gradient descent optimization from...

★ · 13 min read · Apr 10



94



Lists

What is ChatGPT?

9 stories · 64 saves

Staff Picks

330 stories · 83 saves

 Peter Kar... in Artificial Intelligence in Plain Eng...

L1 (Lasso) and L2 (Ridge) regularizations in logistic...

Logistic regression , Lasso and Ridge regularizations, derivations, math

★ · 6 min read · Feb 3




38



1



...

 Alexander Nguyen in Level Up Coding

Why I Keep Failing Candidates During Google Interviews...

They don't meet the bar.

★ · 4 min read · Apr 13



4.2K



127



...

 Rahul Pandey in Geek Culture

How to deal with vanishing and exploding gradients in neural...

This article provides a detailed overview of identifying and dealing with vanishing and...


★ · 7 min read · Dec 17, 2022



54



...

 The PyCoach in Artificial Corner

You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% ...

Master ChatGPT by learning prompt engineering.

★ · 7 min read · Mar 18



21K



364



...

[See more recommendations](#)