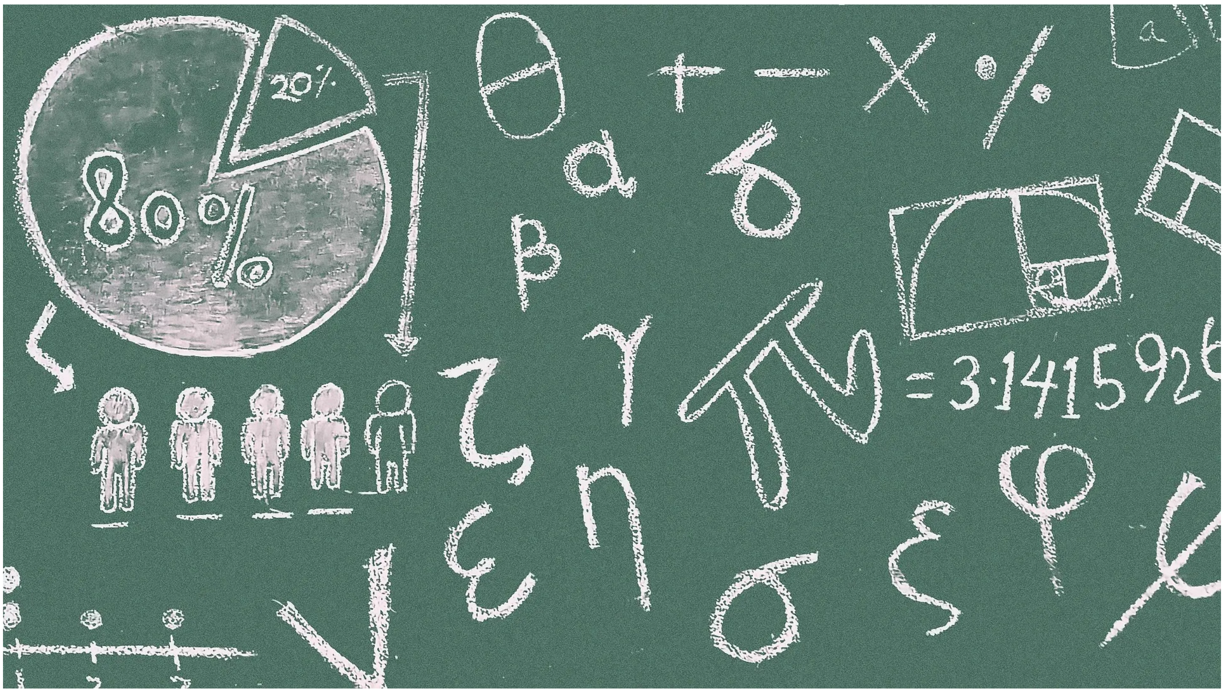
Finding the Cost Function of Neural Networks

Part 1 of Step by Step: The Math Behind Neural Networks

Chi-Feng Wang · Follow

Published in Towards Data Science · 4 min read · Sep 1, 2018



Title image: Source

Terence Parr and Jeremy Howard's paper, *The Matrix Calculus You Need for Deep Learning,* provided a lot of insights into how deep learning libraries like Tensorflow or PyTorch really worked. Without understanding the math behind deep learning, all we are doing is writing a few lines of abstract code — build a model, compile it, train it, evaluate it — without really learning to appreciate all the complex intricacies that support all

This (and the next few) articles will be the reflections of what I learned from Terence and Jeremy's paper. This article will introduce the problem, which we will solve in the upcoming articles. I'll explain most of the math and add in some of my insights, but for more information, definitely check out the original paper.

These articles (and the paper) both assume a basic knowledge of high-school level calculus (i.e. the derivative rules and how to apply them). You can check out the Khan Academy videos if you want to revisit them.

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…, 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 order to minimize loss, we use the concept of gradient descent. As explained here (read this if you aren't familiar with how gradient descent works), gradient descent calculates the slope of the loss function, then shifts the weights and biases according to that slope to a lower loss.

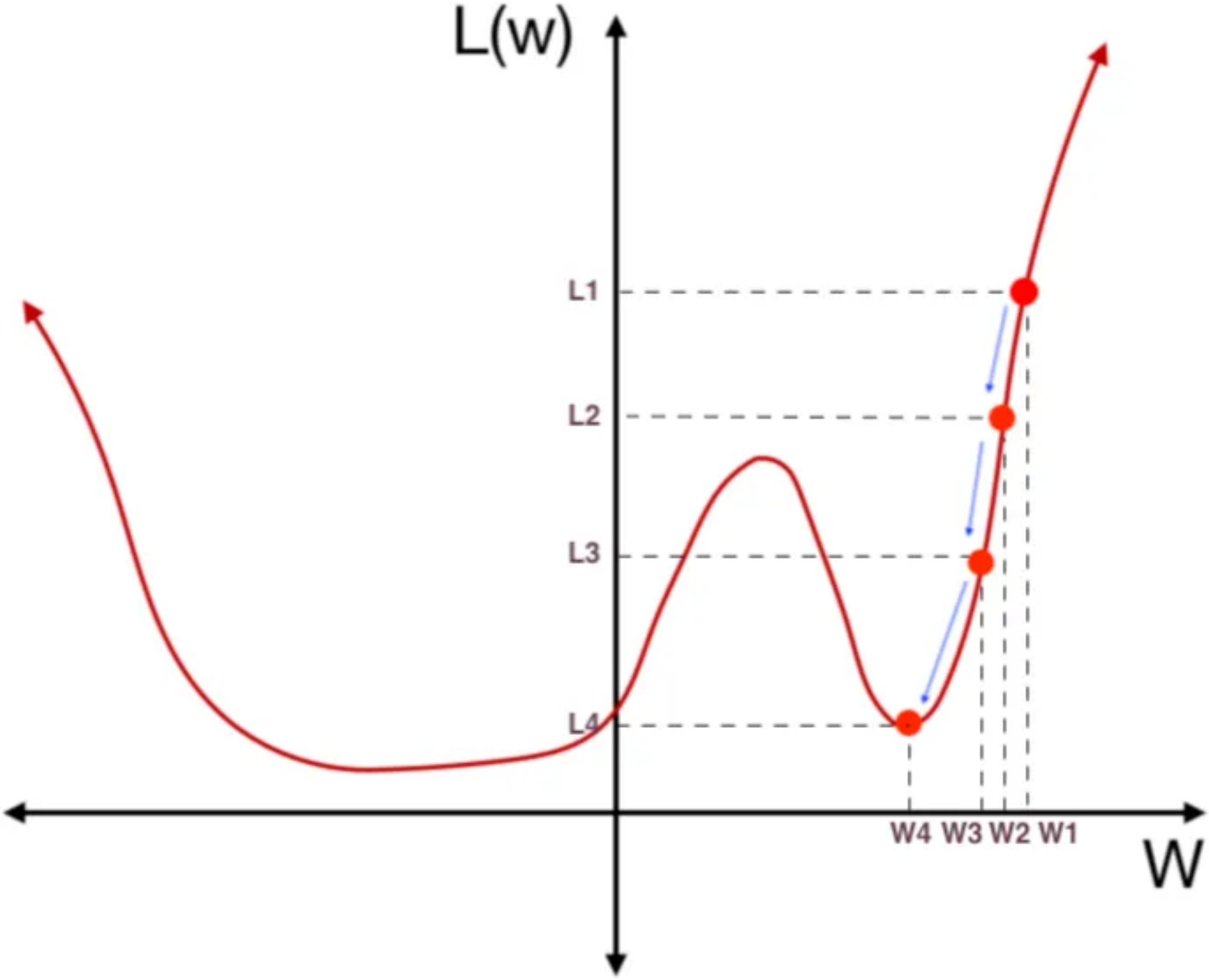


Image 1: Shifting the weights and biases of to minimize loss

We need to find the slope of our loss function. Before we do that, however, let us define our loss function. MSE simply squares the difference between every network output and true label, and takes the average. Here's the MSE equation, where C is our loss function (also known as the *cost function*), N is the number of training images, **y** is a vector of true labels ($\mathbf{y} = [target(\mathbf{x}_1), target(\mathbf{x}_2)...target(\mathbf{x}n)]$), and **o** is a vector of network predictions. (In case you haven't noticed already, variables in **bold** are vectors.)

Image 2: Loss function

We can further expand this equation. What are the network outputs? We feed in a vector input — let's call that **x** — to our fully-connected layer. The activations of that layer are our network outputs. What are the activations of our fully-connected layer? Each item in our vector input is multiplied by a certain weight each. Then, all these products are added together, and a bias added on top of that. Finally, that value is passed through a ReLu to form the activation for our one neuron in the fully-connected layer.

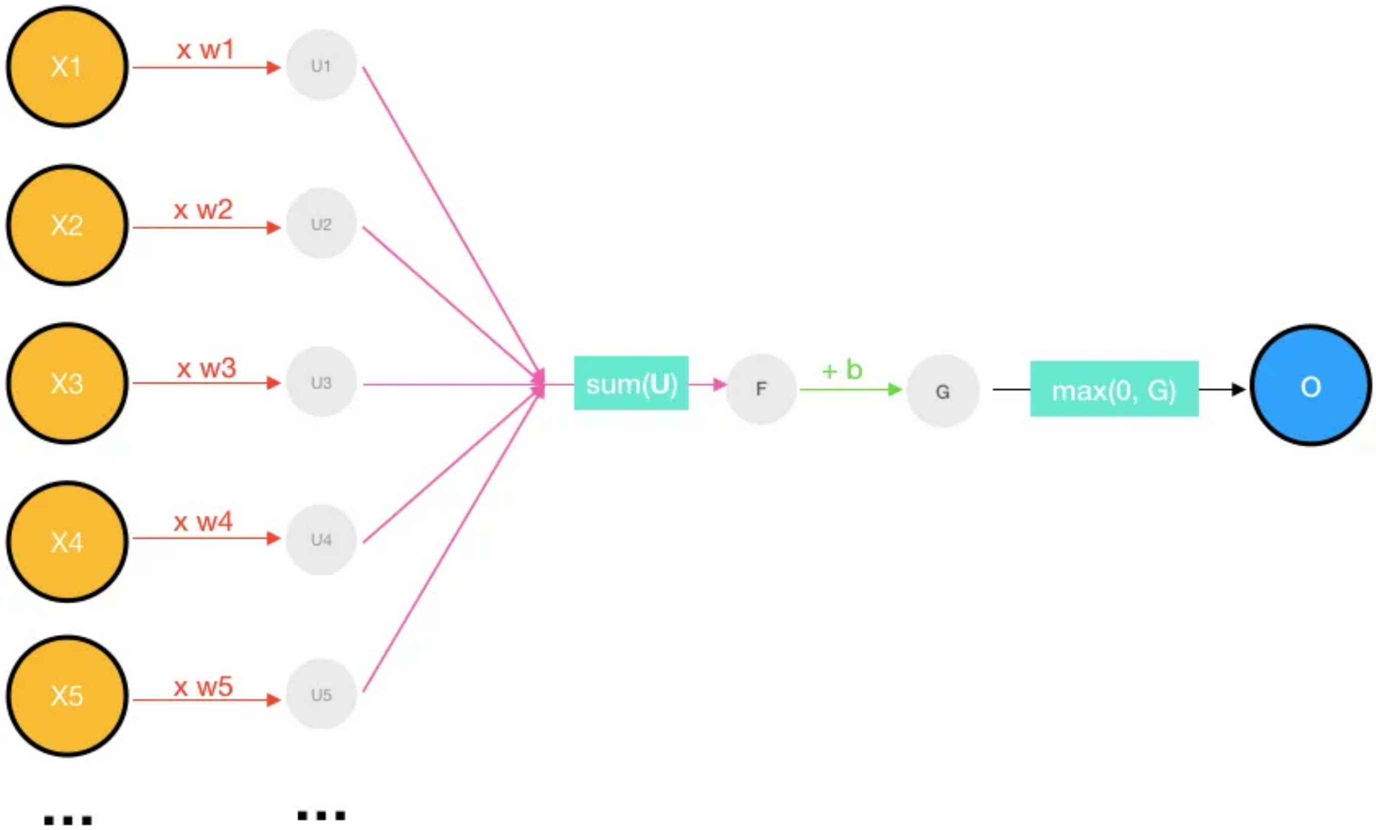


Image 3: Fully-connected layer, expanded. The orange circles are the input values, the blue circle is the output value (the prediction, since our network only has 1 layer), and the gray circles are just intermediate values used in the calculation. Each red line represents multiplication by a certain weight, the pink lines represent summing up all the values, and the green line represents addition with a certain bias.

Summing up the products of each input value and each weight is essentially a vector dot product between the input **x** and a vector of weights (let's call that **w**). A ReLU is simply a function that converts any negative values to 0. Let's rename that as the *max(0,z)* function, which returns z if z is positive and 0 if z is negative.

Put that altogether, and we get the equation of our activation for the neuron:

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

Now let’s substitute that into our loss function. Because we train with more than one input, let us define **X** as a collection of all our inputs:

$$X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T$$

Image 4: X // Source

Since we only have one layer (with one neuron), the activation of this neuron is the prediction of our model. Hence, we can substitute our activation in for *o* in our loss function:

Image 5: Loss function with activation substituted in

and then substitute the activation function:

Image 6: Loss function

This is the loss function that we have to find the slope to.

In order to find the slope, we have to find the loss function's derivative. Not just any derivative, however — it has to be the partial derivative with respect to the weights and with respect to the biases (since these are the values we are tweaking).

Check out Part 2 to learn how to calculate partial derivatives!

Jump ahead to other articles:

- Part 2: Partial Derivatives
- Part 3: Vector Calculus
- Part 4: Putting It All Together

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 :)

Neural Networks | Artificial Intelligence | Derivatives | Matrix | Calculus

Written by Chi-Feng Wang

1.5K Followers · Writer for Towards Data Science

Follow

Student at UC Berkeley; Machine Learning Enthusiast

More from Chi-Feng Wang and Towards Data Science

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

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

Ester Hlav in Towards Data Science

Xavier Glorot Initialization in Neural Networks—Math Proof

Detailed derivation for finding optimal initial distributions of weight matrices in…

· 9 min read · Dec 23, 2022

38 3

Lists

What is ChatGPT?

9 stories · 64 saves

Staff Picks

330 stories · 83 saves

Ester Hlav in Towards Data Science

Kaiming He Initialization in Neural Networks—Math Proof

Deriving optimal initial variance of weight matrices in neural network layers with ReL...

· 10 min read · Feb 15

136 3

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

Matt Chapman in Towards Data Science

The Portfolio that Got Me a Data Scientist Job

Spoiler alert: It was surprisingly easy (and free) to make

· 10 min read · Mar 25

2.9K 44

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