Simple Backpropagation Proof

London Lowmanstone IV

July 2018

1 Introduction

This is a very simple proof/explanation of the math behind the backpropagation algorithm.

Here we assume we have a neural network with no biases and no activation functions. That is, each layer of the neural network is computed by multiplying the output of the previous layer by a weight matrix.

2 Definition of variables

We have a neural network with L layers. A simple neural network with just an input layer and an output layer and one set of weights between the two, would have $L = 2$. The Lth layer is the last layer, also known as the output layer.

The outputs of a layer n are notated as a_n. Thus, the final output of the network is written as a_L. The weight matrix that goes from the nth layer to the $(n+1)$th layer is written as w_n.

The cost function, $C(a_L, y)$ is a function of the output layer of the neural network, a_L, and the target output that was wanted from the neural network y. A common example of a cost function is known as mean-squared-error and is generally computed as $C(a_L, y) = \frac{1}{2}(a_L - y)^2$. Since cost functions are always functions of the last layer and a target output, the cost function will generally be written as just C.

To summarize:
L: the number of layers in the neural network (including the input and output layers)
a_n: the output of the nth layer
a_L: the output of the last layer in the network
w_n: the weight matrix from the nth layer to the $(n+1)$th layer
C: the cost function for the network

3 Goal

The goal of backpropagation is to minimize the cost function C by changing the weights w inside of the network. The idea is as follows. Let's assume we can compute the slope of the function with respect to how we change the weights in our network. That is, let's assume we can compute $\frac{\partial C}{\partial w_n}$ for each weight matrix w_n in the network. If we can, then we can move the weights in the direction that the slope suggests will make C smaller, thus minimizing C.

In practice, we choose some learning rate α and then update each weight matrix w_n by doing the update

$$\text{new}\, w_n = (\text{old}\, w_n) - \alpha * \frac{\partial C}{\partial w_n}.$$

That is, the bigger the slope of the cost function, the more we move the weights in the direction where we think the cost will be lower.

And that's backpropagation. The hope is that if we minimize the cost for a lot of different inputs, the same weights will also minimize the cost for inputs it hasn't seen yet, and so the neural network will be accurate and generalize across new inputs.

So, this paper focuses on proving that we can indeed compute $\frac{\partial C}{\partial w_n}$ for each weight matrix w_n, and thus we can perform that update step to train the network to minimize the loss.

Overall, our goal is to prove that $\frac{\partial C}{\partial w_n}$ can be computed.

4 Assumptions

These are things we assume to be true by the nature of neural networks and how we've defined them.

4.1 Equations

Since this is a very simple neural network with no activation functions and no bias, the output of any layer is merely the output of the previous layer multiplied by the weight matrix in between them. Thus,

$$a_{n+1} = w_n a_n.$$

We're also going to define a new quantity called the "error" in a layer, notated as E_n for the error in layer n. The error is defined as

$$E_n = \frac{\partial C}{\partial a_n}.$$

This is the slope of the cost function with respect to the output of a layer. We define this intermediate quantity because it will help with our proof later on.

4.2 Computable Quantities

Since our goal is to show that $\frac{\partial C}{\partial w_n}$ can be computed, we need to make assumptions about what things are already computable, and then prove that the computation for $\frac{\partial C}{\partial w_n}$ relies only on quantities that are computable.

In this proof, we assume the following entities are computable:

a_1: the input to the neural network. We assume that we know what we put into our own neural network.

w_n: the weights of the neural network. It is assumed that we can access the weights of the network (especially since we're going to be updating them).

a_n: the output of each layer in the neural network. It is assumed that we can run the neural network to obtain these values. (Technically, we could prove that this is computable since w_n and a_1 are assumed to be computable and $a_{n+1} = w_n a_n$, but it seemed easier to just include this as an assumption.)

y: the target output of the neural network. We assume that during training we know what we want the network to output. (This is called "supervised learning.")

$\frac{\partial C}{\partial a_L}$: the slope of the cost function with respect to the output of the network. It is assumed that this can be computed for any a_L and y. For example, for the mean-squared-loss formula, $\frac{\partial C}{\partial a_L} = \frac{\partial \frac{1}{2}(a_L - y)^2}{\partial a_L} = a_L - y$, which is computable. (We have assumed that both a_n and y are computable, thus, thus $a_L - y$ is indeed computable.)

To summarize, we assume that the following quantities are computable: a_1, w_n, a_n, y, and $\frac{\partial C}{\partial a_L}$.

5 Proof

Now that everything is defined, and we know which things are computable, we can begin our proof that $\frac{\partial C}{\partial w_n}$ is computable.

First, let's show that the error in the last layer is computable. This error is is defined as

$$E_L = \frac{\partial C}{\partial a_L},$$

which is assumed to be computable. Thus E_L is computable.

Now, let's see if we can come up with a computable formula for E_n, the error in any layer, based on the error in the last layer, E_L.

From the definition of E_n we have:

$$\begin{aligned} E_n &= \frac{\partial C}{\partial a_n} \\ &= \frac{\partial C}{\partial a_{n+1}} \frac{\partial a_{n+1}}{\partial a_n} \text{ (By the Chain Rule)} \\ &= E_{n+1} \frac{\partial a_{n+1}}{\partial a_n} \text{ (By the definition of } E_n\text{)} \\ &= E_{n+1} \frac{\partial w_n a_n}{\partial a_n} \text{ (By the definition of } a_{n+1}\text{)} \\ &= w_n^\top E_{n+1}. \end{aligned}$$

(Note that in the last line I rearranged the terms and transposed the weight matrix in order to make sure the dimensions matched up.)

So, since E_L is computable, we can use it to compute $E_{L-1} = w_{L-1}^\top E_L$ and similarly use E_{L-1} to compute $E_{L-2} = w_{L-2}^\top E_{L-1}$, etc. (This is a simplification of a formal proof by induction.) Thus, E_n is computable.

You may be beginning to see why this algorithm is called "backpropagation," as we are using the results from future layers to compute values for previous layers.

Our last step is to finally prove that $\frac{\partial C}{\partial w_n}$ is computable. From the Chain Rule, we have:

$$\begin{aligned} \frac{\partial C}{\partial w_n} &= \frac{\partial C}{\partial a_{n+1}} \frac{\partial a_{n+1}}{\partial w_n} \\ &= E_{n+1} \frac{\partial a_{n+1}}{\partial w_n} \text{ (By the definition of } E_n\text{)} \\ &= E_{n+1} \frac{\partial w_n a_n}{\partial w_n} \text{ (By the definition of } a_{n+1}\text{)} \\ &= E_{n+1} a_n^\top. \end{aligned}$$

(Again, note the transpose to make sure the dimensions match up.)

We've shown that E_n is computable, and we assumed a_n is computable, and thus, $\frac{\partial C}{\partial w_n}$ is computable as well.

Note that backpropagation all starts by determining the error in the last layer, and then using that to compute how the weights in previous layers should change.

To understand more of the math behind neural networks with activation functions and biases, **http://neuralnetworksanddeeplearning.com/chap2.html** is a good resource. (It's also where I learned the equations for this simpler proof.)