

1. Introduction

For our neural network to learn we need to adjust its weights and biases. This means calculating the gradient of the cost with respect to every weight and bias in the network. For convenience, we will store this gradient as two separate matrices for each layer: a 2D matrix of how much the cost changes when you adjust each weight ($\frac{\partial C}{\partial w}$) and a 1D matrix of how much the cost changes when you adjust each bias ($\frac{\partial C}{\partial b}$). Each row of the 2D weight gradient will represent a destination neuron, and each column an input neuron. This simplifies our vector calculations ([more on this here](#)).

Once we have the gradient, we can adjust the weights & biases by subtracting from them the gradient multiplied by a learning rate (typically a very small number).

Note that we can only calculate the gradient after running a sample (or a batch of samples) through the network. We must then start by doing so, making sure to store the activations of each layer as we go - as those will be needed for our calculations.

This document was written for a neural network where every layer except the last has a sigmoid activation function, and where the last layer has a softmax activation followed by the cross entropy cost. The entirety of the document is agnostic to the number of layers and their size. A large part of the document is also agnostic to the cost or activation functions, which only show up in part 3.

2. Chain rule

The overarching goal of backpropagation is to calculate the gradient for every weight and bias in the network. The chain rule tells us that:

$$\frac{\partial C}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad (1)$$

and

$$\frac{\partial C}{\partial b_i^l} = \frac{\partial z_i^l}{\partial b_i^l} \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad (2)$$

Where C is the cost, w_{ij}^l is the weight connecting neuron j of layer $l-1$ to neuron i of layer l , z_i^l is the weighted input of neuron i of layer l , a_i^l is the activation of neuron i of layer l , and b_i^l is the bias of neuron i of layer l . In this document we use the superfix *(this thing here)* not for exponentiation but for indexing the layer a variable belongs to.

The last factor $\frac{\partial C}{\partial a_i^l}$ of these formulas shows us that in order to compute the weight and

bias gradients we will also need to calculate the gradient with respect to the activations. For the last layer of the network it can be calculated directly using the derivative of the cost function, but for the other layers it is expanded as follows:

$$\frac{\partial C}{\partial a_i^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \cdot \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \cdot \frac{\partial C}{\partial a_k^{l+1}} \quad (3)$$

Where the sum is over all k neurons of layer $l + 1$.

This formula is very similar to (1) and (2), with the last two factors also being present in the others. This hints at the fact that we can reuse some of the calculations when computing these formulas later.

The last factor $\frac{\partial C}{\partial a_k^{l+1}}$ will need to be expanded again if layer $l + 1$ is not yet the last layer, and so on. This makes evident how calculating the gradients involves starting with the last layer, and then backpropagating through the network: the gradients of each layer depend on how that layer's activations affect the following layers.

3. Calculating the derivatives

We can now move on to calculating the derivatives present in the chain rule expressions above, so that we can turn them into something we can actually use.

The weighted input for each neuron is calculated in the following way:

$$z_i^l = \left(\sum_j w_{ij}^l \cdot a_j^{l-1} \right) + b_i^l$$

The sum is over all j neurons of the previous layer $l - 1$. The first layer for which the weighted input is calculated comes after the input layer, so that we always have a layer $l - 1$. We will need the partial derivatives of z with respect to each of its inputs:

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = a_j^{l-1} \quad \frac{\partial z_i^l}{\partial b_i^l} = 1 \quad \frac{\partial z_i^l}{\partial a_j^{l-1}} = w_{ij}^l$$

We will also need the derivatives involving the cost function and the activation functions. Different layers can have different activation functions, which is the case for our network: our last layer has a softmax activation function, while the previous layers all have a sigmoid activation function. We will deal with these and the cost function later in this document.

We can now replace the weighted input derivatives in (1), (2), and (3) with the results we just calculated:

$$\frac{\partial C}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad \Rightarrow \quad \frac{\partial C}{\partial w_{ij}^l} = a_j^{l-1} \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad (4)$$

$$\frac{\partial C}{\partial b_i^l} = \frac{\partial z_i^l}{\partial b_i^l} \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad \Rightarrow \quad \frac{\partial C}{\partial b_i^l} = 1 \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad (5)$$

$$\frac{\partial C}{\partial a_i^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \cdot \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \cdot \frac{\partial C}{\partial a_k^{l+1}} \quad \Rightarrow \quad \frac{\partial C}{\partial a_i^l} = \sum_k w_{ki}^{l+1} \cdot \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \cdot \frac{\partial C}{\partial a_k^{l+1}} \quad (6)$$

4. Backpropagating the last layer

We now need to calculate the derivatives for the cost and activation functions used in our neural network. We will start by dealing with the last layer, where we are using a softmax activation followed by the cross entropy cost. This cost is calculated in the following way:

$$\frac{1}{n} \sum_n \sum_i y_i \ln \left(\frac{1}{a_i} \right)$$

Where the first sum is over all n training samples, the second sum is over all i neurons in the last layer, y_i is the ground truth activation for neuron i , and a_i is that neuron's actual activation.

However, the function we derive for the cost is the loss function, which calculates the cost for a single sample:

$$C = \sum_i y_i \ln \left(\frac{1}{a_i} \right)$$

There is a clever simplification for calculating the derivatives of the last layer. When it applies a softmax activation followed by the cross entropy loss, as described on page 3 of <https://www.ics.uci.edu/~pjsadows/notes.pdf>, we end up with the formula:

$$\frac{\partial C}{\partial z_i} = a_i - y_i \quad (7)$$

Where y_i is the ground truth activation for neuron i , and a_i is its actual activation. This formula replaces the last two factors of (1), (2), and (3):

$$\frac{\partial a_i}{\partial z_i} \cdot \frac{\partial C}{\partial a_i} = \frac{\partial C}{\partial z_i} = a_i - y_i$$

Formulas (4) and (5) then become:

$$\frac{\partial C}{\partial w_{ij}^l} = a_j^{l-1} \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad \Rightarrow \quad \frac{\partial C}{\partial w_{ij}^l} = a_j^{l-1} \cdot (a_i^l - y_i)$$

$$\frac{\partial C}{\partial b_i^l} = 1 \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad \Rightarrow \quad \frac{\partial C}{\partial b_i^l} = a_i^l - y_i$$

There is no need to calculate (6) for the last layer, as the formulas above stand on their own.

With a little analysis, we can convert them into vectorized expressions:

$$\frac{\partial C}{\partial w_{ij}^l} = a_j^{l-1} \cdot (a_i^l - y_i) \quad \Rightarrow \quad \frac{\partial C}{\partial w^l} = (a^l - y) \cdot (a^{l-1})^T \quad (8)$$

$$\frac{\partial C}{\partial b_i^l} = a_i^l - y_i \quad \Rightarrow \quad \frac{\partial C}{\partial b^l} = a^l - y \quad (9)$$

Note that all vectors are assumed to be column vectors, meaning the same as a matrix with just one column. Subtraction is element-wise, and \cdot represents matrix multiplication.

5. Backpropagating the remaining layers

The remaining layers of our neural network apply a sigmoid activation to the weighted inputs:

$$\sigma(z_i) = \frac{1}{1 + e^{-z_i}}$$

And thus the derivative of the activations becomes:

$$\frac{\partial a_i}{\partial z_i} = \sigma'(a_i) = \sigma(a_i) \cdot (1 - \sigma(a_i))$$

We will use the same expression σ' to denote both the scalar and the vectorized version of the sigmoid's derivative. The version being used can be inferred from context: if we are passing in a vector, we are using the vectorized version:

$$\sigma'(a) = \sigma(a) \odot (1 - \sigma(a))$$

Where $\sigma(a)$ represents the sigmoid function applied elementwise to vector a , \odot represents the hadamard (element-wise) product, and the subtraction operation is also applied element-wise.

Formulas (4) and (5) then become:

$$\frac{\partial C}{\partial w_{ij}^l} = a_j^{l-1} \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad \Rightarrow \quad \frac{\partial C}{\partial w_{ij}^l} = a_j^{l-1} \cdot \sigma'(a_i^l) \cdot \frac{\partial C}{\partial a_i^l}$$

$$\frac{\partial C}{\partial b_i^l} = 1 \cdot \frac{\partial a_i^l}{\partial z_i^l} \cdot \frac{\partial C}{\partial a_i^l} \quad \Rightarrow \quad \frac{\partial C}{\partial b_i^l} = 1 \cdot \sigma'(a_i^l) \cdot \frac{\partial C}{\partial a_i^l}$$

We can rewrite these as vectorized expressions:

$$\frac{\partial C}{\partial w^l} = a^{l-1} \cdot \sigma'(a^l) \cdot \frac{\partial C}{\partial a^l} \quad \Rightarrow \quad \frac{\partial C}{\partial w^l} = \left(\sigma'(a^l) \odot \frac{\partial C}{\partial a^l} \right) \cdot (a^{l-1})^T \quad (10)$$

$$\frac{\partial C}{\partial b^l} = 1 \cdot \sigma'(a^l) \cdot \frac{\partial C}{\partial a^l} \quad \Rightarrow \quad \frac{\partial C}{\partial b^l} = \sigma'(a^l) \odot \frac{\partial C}{\partial a^l} \quad (11)$$

Where all vectors are assumed to be column vectors.

Note that we still have to specify $\frac{\partial C}{\partial a_i^l}$, which depends on the kind of layer that comes after layer l .

If the next layer is not yet the last, and has a sigmoid activation (as all non-final layers in our network), formula (6) will become:

$$\frac{\partial C}{\partial a_i^l} = \sum_k w_{ki}^{l+1} \cdot \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \cdot \frac{\partial C}{\partial a_k^{l+1}} \quad \Rightarrow \quad \frac{\partial C}{\partial a_i^l} = \sum_k w_{ki}^{l+1} \cdot \sigma'(a_k^{l+1}) \cdot \frac{\partial C}{\partial a_k^{l+1}}$$

Which can be vectorized into the form:

$$\frac{\partial C}{\partial a^l} = (w^{l+1})^T \cdot \left(\sigma'(a^{l+1}) \odot \frac{\partial C}{\partial a^{l+1}} \right) \quad (12)$$

If layer $l + 1$ is the last layer, however, two things are different: we have a softmax activation function, and we can calculate the derivative of the cost directly. Since we already calculated an expression (7) that combines the two missing derivatives, we just need to insert it into (6):

$$\frac{\partial C}{\partial a_i^l} = \sum_k w_{ki}^{l+1} \cdot \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \cdot \frac{\partial C}{\partial a_k^{l+1}} \quad \Rightarrow \quad \frac{\partial C}{\partial a_i^l} = \sum_k w_{ki}^{l+1} \cdot (a_k^{l+1} - y_k)$$

We can also vectorize this expression:

$$\frac{\partial C}{\partial a^l} = (w^{l+1})^T \cdot (a^{l+1} - y) \quad (13)$$

Where the subtraction applies element-wise, \cdot represents matrix multiplication, and vectors are 1-column matrices by default.

6. In practice

When it comes to implementing backpropagation in code, the formulas to apply are the final vectorized expressions: (8), (9), (10), (11), (12), and (13).

Here is a rough overview of the steps that need to be taken:

1. Feedforward a sample through the network, storing activations of each layer along the way
2. Calculate cost and store for reference (since we are not training in batches, we can track the accuracy of the model as we train by logging how many of the last x samples were guessed correctly. The global accuracy can be logged every epoch)
3. Calculate gradient for weights & biases of last layer
4. Calculate gradient for activations of the previous layer
5. Use the activations gradient to calculate weights & biases gradient
6. Repeat until input layer is reached
7. Update weights & biases by subtracting the gradient multiplied by a small learning rate
8. Go back to 1.