Backpropagation Explanation for a scalable multilayer perceptron

Callum Frederiksen

May 2025

1 Introduction

This is the explanation behind the math for my MLP project.

2 Model Overview

2.1 MLP Architecture

TODO: INSERT MLP DIAGRAM

The underlying activation for this model is the sigmoid activation function, as it will scale output values between 0 and 1, as well as being non-linear and continuous for $\sigma(x) \in [0, 1]$ for $x \in \mathbb{R}$. our activation $\sigma(x)$ is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}},\ x \in \mathbb{R}$$

. Interestingly, when you take the derivative of $\sigma(x)$ with respect to x,

$$\sigma\prime(x) = \sigma(x)(1 - \sigma(x))$$

This will be used extensively when backpropagating throughout the algorithm to determine the derivative of the Loss function with respect to each set of parameters (defined as weights, and biases).

2.2 The Goal of backpropogation

The end goal of backpropogation is to use the gradient descent algorithm

$$\theta_{i+1} := \theta_i - \alpha \nabla L(\theta_i)$$

Where θ_i represents the collective paramters (weights and biases) for the ith iteration, and α represents the learning rate of the model.

Backpropogation can be used to achieve this, by iterating the gradient descent algorithm until convergence, in the best case scenario.

2.3 The Forward Pass

The forward pass is where the deep neural network can take the inputs $\vec{X}$, can be passed throughout the neural network, and compute the final binary prediction $\hat{y}$.

This is computed throughout the network by the linear formula

$$\vec{z}^{(n)} = \boldsymbol{W}^{(n)} \cdot \vec{a}^{(n-1)} + b^{(n)}$$

Where n refers to the layer for the activation vectors, and the layer the weight is being passed into for the weights.

The non-linear sigmoid activation function can be used to normalise the values between 0 and 1.

$$\therefore \vec{a}^{(n)} = \sigma(\vec{z}^{(n)})$$

This is repeated across the network, throughout each layer until the *final layer*. This has then computed the model's prediction $\hat{y}$ *(note as there is only one value, this will be a scalar, but is represented as a* 1×1 *matrix to simplify the numpy implementation in the code)*

2.4 The loss function

To calculate the models loss, we will use the sum of squared errors loss (SSE). This allows us to determine how incorrect the model is (i.e. if the model has a high loss, it is inaccurate, therefore minimizing the loss maximises its accuracy) The SSE formula is defined below:

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

(for a single training example)

3 The backpropagation algorithm

3.1 The high-level intuition

Essentially, the backpropagation algorithm is a partial derivative that computed the gradient of the loss with respect to each parameter. These parameters refer to the weights and biases for each layer.

This can be done in steps, each step will be notated as $\delta^{(n)}$, where

$$\delta^{(n)} = \frac{\partial L}{\partial \vec{z}^{(n)}}$$