Simple, Fully Connected, Vectorized Neural Networks

November 2021

Matthew Christopher Jansen

ABSTRACT

Neural networks are amazing tools capable of solving classification and prediction problems. This discussion serves to reflect my understanding of neural networks by considering a vectorized approach.

1 INTRODUCTION: WHAT IS A NEURAL NETWORK?

A Neural Network is a computational graph structure which consists of layers of nodes called 'neurons' which have biases and connecting weighted edges called 'synapses'. The concept of the neural network is biologically inspired by the function of a biological brain and replicates its learning capabilities.

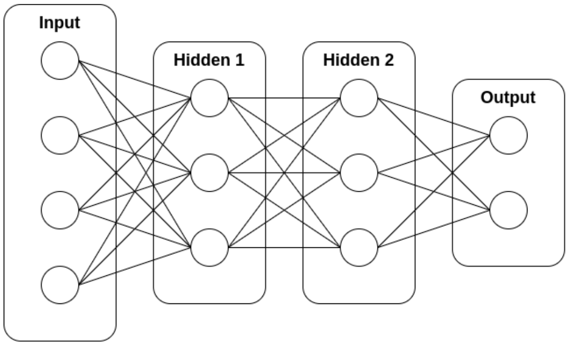

Figure 1: Example of a simple neural network with 2 hidden layers.

Neural networks are used for pattern recognition in real-world data to recognise general patterns which would be rather tedious to identify with explicit analytical methods. A clear example is the classification of hand-written digits from 0 to 9. Although there are a set number of digits from 0 to 9, it is still too difficult for a machine to classify them with an explicitly constructed algorithm as the handwriting pattern with which they are written differ among all persons. An impractically large set of rules would be required for such an algorithm to classify the digits and would require more rules as new digit images become more apparent since the algorithm would not be able to adapt to the new patterns in handwriting.

A neural network would be the ideal approach as it can be trained to recognise generalised patterns within the handwriting of the digit images in order to classify the digits implicitly. This will also allow for the recognition of new unseen digit images as the neural network can learn the 'explicit rules/ patterns' implicitly from the images themselves.

2 SO HOW DOES A NEURAL NETWORK LEARN?

A neural network is trained and tested with datasets which are split into training and testing sets. The process of training a neural network consists of three steps:

1. **Forward Propagation:**
   Here the inputs are fed to the network in order to obtain an output similar to the expected output from the dataset.

2. **Backpropagation:**
   This step covers the process through which the *relevance* of each neuron's activation values is computed with respect to the cost of the neural network's output.

3. **Update Network:**
   Finally, based on the results from backpropagating the output, the new weights and biases of the neural network are computed and updated.

The steps above are executed each time the neural network is fed input data from the dataset it is being trained on. Now, let us discuss these steps in detail.

2.1 Forward Propagation

The objective of a trained neural network is to, in rather rigorous terms, approximate a mapping which takes elements from the input space X and maps them to the expected output space Y. Or rather, the neural network attempts to approximate the following mapping:

$$NN : X \mapsto Y \tag{1}$$

Let us begin by considering Fig.1 which depicts a neural network with two hidden layers. During forward propagation an input column vector x, or rather a_1, is fed to the network in order to compute an output $\bar{y}$ which resembles the expected output y for the input x.

In order to achieve this, the input x needs to be propagated from the input layer a_1, through the hidden layers of the network and finally to the output layer a_L. This is achieved by computing the activation a_{i+1} for each layer's neurons up to a_L by means of computing the linear combination of the previous layer z_i and passing it into a nonlinear function known as an activation function, denoted as σ_i. So, generally to compute the activations for any layer:

$$\begin{cases} z_{i+1} : W_i^T a_i + b_i \\ a_{i+1} : \sigma_i(z_i) \end{cases} \tag{2}$$

where:

- W_i**:** weight matrix containing weight values of edges connecting layer[i, i + 1].

- a_i**:** activation vector containing activation values for layer[i].
- b_i**:** bias vector containing the bias values for each neuron in layer[i].
- z_i**:** resulting vector from the linear combination of the weights, activation and bias from the previous layer[i-1].
- σ_i: activation function defined for layer[i] (usually the same function for all layers in the network).

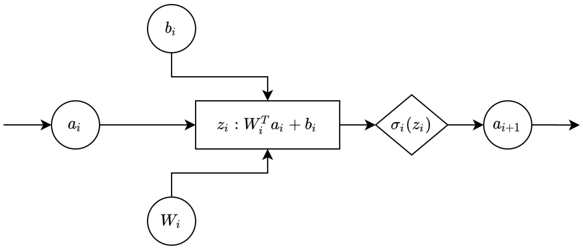

Figure 2: Depiction of forward propagation between layers of a neural network.

Note if layer[i] has m neurons and layer[i+1] has n neurons, then the weight matrix containing the values of the weights connecting layer[i, i+1] has dimension (n, m). That is:

$$\dim W_i = (\dim a_{i+1}, \dim a_i) = (n, m) \tag{3}$$

By using this convention it requires the weight matrices to be transposed for nearly all computations in the neural network. So, looking at our example, the following equations describe how the input would be propagated through the neural network:

$$\begin{aligned} a_1 &: x & z_1 &: W_1^T a_1 + b_1 \\ a_2 &: \sigma_1(z_1) & z_2 &: W_2^T a_2 + b_2 \\ a_3 &: \sigma_2(z_2) & z_3 &: W_3^T a_3 + b_3 \\ a_4 &: \sigma_3(z_3) \end{aligned} \tag{4}$$

Fig.[2] depicts the process of forward propagation which we have now discussed. Now onto the exciting part, the backpropagation algorithm.

2.2 Backpropagation Algorithm

The goal of the backpropagation algorithm is to determine how the weights and biases of the neural network influences its output and to update these weights and biases such that it improves the network's capability to classify/ predict input data better. This is achieved by introducing a cost function which is used to compute the 'effort' the network requires to compute an output relative to the expected output. We can also view the cost function as a measure of error in the output of the neural network relative to the expected output.

A commonly used cost function is the Mean-Square-Error (MSE) function, defined as:

$$MSE = \frac{1}{N} \sum_j (Y_j - \bar{Y}_j)^2 \tag{5}$$

However, we'll consider a slightly modified version of the MSE as the cost function to simplify further derivations:

$$C(y, \bar{y}) = \frac{1}{2} MSE = \frac{1}{2N} (y - \bar{y})^2, \quad y, \bar{y} \in Y, \bar{Y} \tag{6}$$

Now for a neural network to 'learn', the backpropagation algorithm is used to minimize this 'effort' a neural network requires to compute an output $\bar{y}$ relative to the expected output y by subjecting the 'effort' to the influence of the networks weights and biases. The problem can be expressed as follows:

$$\min_{\mathbf{W},\, \mathbf{b}} \left\{ C(y, \bar{y}) = \frac{1}{2N} (y - \bar{y})^2 \right\} \tag{7}$$

The minimization of the cost function is achieved by implementing gradient-descent. Think of the Cost space, the set of all possible cost values, as a landscape made up of hills and valleys. Now think of a ball rolling down a hill towards a valley. Now this ball is the current set of weights and biases for the neural network as it's 'learning' and the rolling of the ball as the backpropagation process. The position of the ball on the landscape is the neural network's current cost value or 'effort' for computing the outputs relative to the expected outputs.

Now there must be some way we can alter how fast or slow a neural network should 'learn' and that we can do by changing the texture or drag-coefficient of the landscape. This is done by making use of a learning rate η. By making the landscape extremely rough (small valued η) we would cause the ball to roll down at a slow pace, and making the landscape super smooth (large valued η) will result in a fast rolling ball.

Coming back to the minimization problem in eq. 7, we begin to derive a solution by looking into what is called the rule of deltas. Consider the following:

$$\begin{aligned} \delta_L &: \quad \frac{1}{N} (y - \bar{y}) \sigma_{L-1}(z_{L-1}) \\ \delta_i &: \quad \delta_{i+1} W_{i+1}^T \sigma_i'(z_i) \end{aligned} \tag{8}$$

These delta variables in eq.8 are defined as such since they appear as terms in the gradient of the cost function w.r.t. the weights and biases. Keep these in mind as we derive the expressions for the gradients of the cost function w.r.t. the weights and biases.

In order to solve the minimization problem stated in eq.7, we must compute the gradient of the cost function w.r.t. the weights and biases of the neural network so that we can compute the new and improved weights and biases for the network by applying gradient descent. We will first start by computing the gradients of the cost function w.r.t. the weights in the network (remember we are using fig.[1] as an example):

$$\begin{aligned} \nabla_{W_3} C &= \frac{\partial C}{\partial \bar{y}} \frac{\partial \bar{y}}{\partial z_3} \frac{\partial z_3}{\partial W_3} \\ \nabla_{W_2} C &= \frac{\partial C}{\partial \bar{y}} \frac{\partial \bar{y}}{\partial z_3} \frac{\partial z_3}{\partial a_3} \frac{\partial a_3}{\partial z_2} \frac{\partial z_2}{\partial W_2} \\ \nabla_{W_1} C &= \frac{\partial C}{\partial \bar{y}} \frac{\partial \bar{y}}{\partial z_3} \frac{\partial z_3}{\partial a_3} \frac{\partial a_3}{\partial z_2} \frac{\partial z_2}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial W_1} \end{aligned} \tag{9}$$

So if we compute the partial derivatives we obtain the following expressions for the gradients:

$$\begin{aligned}\nabla_{W_3} C &= \frac{1}{N}(y - \bar{y})\sigma_3'(z_3)a_3 \\ \nabla_{W_2} C &= \frac{1}{N}(y - \bar{y})\sigma_3'(z_3)W_3^T\sigma_2'(z_2)a_2 \\ \nabla_{W_1} C &= \frac{1}{N}(y - \bar{y})\sigma_3'(z_3)W_3^T\sigma_2'(z_2)W_2^T\sigma_1'(z_1)a_1\end{aligned} \tag{10}$$

Now if we factor in the defined expressions of deltas in eq.8, the expressions for the weight gradients simplify as follows:

$$\begin{aligned}\nabla_{W_3} C &= \delta_4 a_3 \\ \nabla_{W_2} C &= \delta_3 a_2 \\ \nabla_{W_1} C &= \delta_2 a_1 \\ \nabla_{W_i} C &= \delta_{i+1} a_i \quad \text{(general def}^n\text{)}\end{aligned} \tag{11}$$

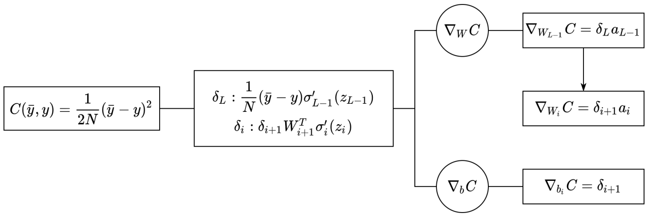
$C(\bar{y}, y) = \frac{1}{2N}(\bar{y} - y)^2$
$\delta_L : \frac{1}{N}(\bar{y} - y)\sigma'_{L-1}(z_{L-1})$
$\delta_i : \delta_{i+1} W_{i+1}^T \sigma_i'(z_i)$
$\nabla_W C$
$\nabla_{W_{L-1}} C = \delta_L a_{L-1}$
$\nabla_{W_i} C = \delta_{i+1} a_i$
$\nabla_b C$
$\nabla_{b_i} C = \delta_{i+1}$

Figure 3: Outline of required operations for the backpropagation algorithm.

Following a similar process for computing the bias gradients yields the following results:

$$\begin{aligned}\nabla_{b_3} C &= \frac{\partial C}{\partial \bar{y}} \frac{\partial \bar{y}}{\partial z_3} \frac{\partial z_3}{\partial b_3} \\ \nabla_{b_2} C &= \frac{\partial C}{\partial \bar{y}} \frac{\partial \bar{y}}{\partial z_3} \frac{\partial z_3}{\partial a_3} \frac{\partial a_3}{\partial z_2} \frac{\partial z_2}{\partial b_2} \\ \nabla_{b_1} C &= \frac{\partial C}{\partial \bar{y}} \frac{\partial \bar{y}}{\partial z_3} \frac{\partial z_3}{\partial a_3} \frac{\partial a_3}{\partial z_2} \frac{\partial z_2}{\partial a_2} \frac{\partial a_2}{\partial z_1} \frac{\partial z_1}{\partial b_1}\end{aligned} \tag{12}$$

And by computing the partials we obtain the following expressions:

$$\begin{aligned}\nabla_{b_3} C &= \frac{1}{N}(y - \bar{y})\sigma_3'(z_3) \cdot 1 \\ \nabla_{b_2} C &= \frac{1}{N}(y - \bar{y})\sigma_3'(z_3)W_3^T\sigma_2'(z_2) \cdot 1 \\ \nabla_{b_1} C &= \frac{1}{N}(y - \bar{y})\sigma_3'(z_3)W_3^T\sigma_2'(z_2)W_2^T\sigma_1'(z_1) \cdot 1\end{aligned} \tag{13}$$

And again, by the applying rule of deltas we obtain the simplified expressions for the bias gradients:

$$\begin{aligned}\nabla_{b_3} C &= \delta_4 \\ \nabla_{b_2} C &= \delta_3 \\ \nabla_{b_1} C &= \delta_2 \\ \nabla_{b_i} C &= \delta_i \quad \text{(general def}^n\text{)}\end{aligned} \tag{14}$$

Using the rule of deltas greatly reduces the complexity of the expressions for the gradients of the cost function w.r.t. the weights and biases of the neural network, especially in the case of a vectorized neural network. In fig.[3] we see an outlined flow of all the required computations needed for the implementation of backpropagation. Now for the final step, updating the neural network.

2.3 Updating the Neural Network

Now that we discussed the inner workings of backpropagation we need to look at how a neural network is updated, or rather how it 'learns'. From backpropagating the computational 'effort' or error in the output of the network we now have the weight and bias gradient vectors. The process of updating the neural network depends on the optimization algorithm which is used to solve eq.7. In our case it is gradient descent, so updating is rather simple. The following rules are used to update the network:

$$\begin{aligned}W_i &:= W_i + \eta\nabla_{W_i} C(y, \bar{y}) \\ b_i &:= b_i + \eta\nabla_{b_i} C(y, \bar{y})\end{aligned} \tag{15}$$

These rules are followed as the network is trained (rolling towards a valley). The following diagram depicts the flow of updating a neural network:

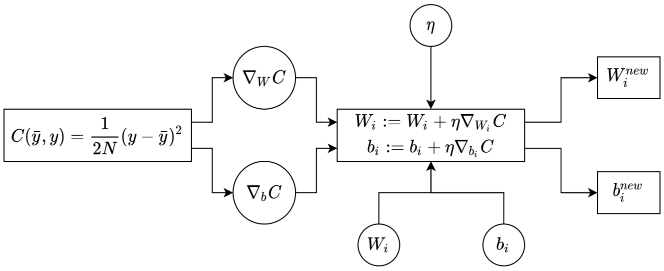
η
$\nabla_W C$
W_i^{new}
$C(\bar{y}, y) = \frac{1}{2N}(y - \bar{y})^2$
$W_i := W_i + \eta\nabla_{W_i} C$
$b_i := b_i + \eta\nabla_{b_i} C$
$\nabla_b C$
b_i^{new}
W_i
b_i

Figure 4: Flow diagram representing the process of updating a neural network.

And that concludes the discussion of training a neural network. Neural networks can be computationally difficult to implement, but once the mathematics behind them are understood the programming of neural networks becomes second nature.

3 CONCLUSION: OBSERVATIONS & REMARKS

It has been quite the journey for me to understand neural networks. Especially the computation of the gradients as the commonly used convention of summation signs in the derivations make understanding the mathematics less intuitive compared to a vectorized approach. I am an advocate of vectorizing mathematical processes as it brings clarity when attempting to construct an algorithm to replicate the process.

The only drawback is that one requires a strong background in linear algebra and vector calculusin order to understand the mathematics. Also, when working with vectorized mathematical processes it's easy to get the dimensions of the matrices and vectors wrong, so be cautious of such bugs when programming these vectorized algorithms. It has been a pleasure writing this discussion and I hope that you, the reader, finds this beneficial.