

# Neural Network Summary

Jaden You

January 6, 2024

## 1 Overview

The neural network consists of layers of nodes or "neurons" with weights connecting of nodes together.

Each node from the previous layer has a weight connecting them to every node from the next layer. The nodes in the layer equal a linear combination of the previous nodes with weights as described.

Additionally, a bias is added to each linear combination and this bias is associated with the output node.

A sigmoid function is then applied on each outputted linear combination to normalize the values between 0 and 1.

These values are initialized as random values but are optimized using a cost function which determines how close the predicted output (normal) vector (completely random on initialization) is to the actual vector value (an elementary vector).

The values of the weights and biases are altered to minimize the cost function using gradient descent.

In a continuous function, the gradient of a function indicates the direction of greatest increase. The opposite direction of this gradient is the direction of greatest decrease.

The gradient is found by taking partial derivatives of the function with respect to each variable (weights and biases in this case). Because there are multiple transformations applied at each layer, we will have to use the chain rule.

The changes indicated by the gradient are repeatedly applied to the function until the gradient is 0 where the neural network is optimized on the training data.

## 2 Structure

The sequence of weights can be expressed as a matrix where columns represent the previous nodes and rows represent the output values sent to the next layer of nodes.

Let us use a two layer neural network as an example

Say our input vector  $u$  has  $n$  entries, the first layer  $l_1$  has  $p$  nodes, the second layer  $l_2$  has  $q$  nodes, and the final output vector  $v$  has  $m$  entries.

Removing biases, the weights can be arranged into a matrix.

For example, let us have a matrix representing the weights between the first input vector and the first layer would be  $p \times n$  and the initial output vector (pre bias and normalization) would be

$$A_0 u$$

Similarly, the matrix  $A_1$  between  $l_1$  and  $l_2$  would be  $q \times p$ .

Additionally, we can store our biases as a vector corresponding to the layer size (e.g.  $l_1$  would have a bias vector  $b_1$  of size  $p$ ). This vector is added to the previous steps' output

$$A_0 u + b_1 = a_1$$

The final step is normalizing each value in this vector using the sigmoid function

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

The output from this function is the layer (for convenience, it will take in a vector parameter here)

$$\sigma(a_1) = l_1$$

### 3 Cost Function

The cost function for a training case is the average squared error between the final output vector and the actual value vector as a function of the alterable values (weights and biases) so the average cost over the entire dataset is

$$C(w_{0,0}^0, w_{0,1}^0, \dots, w_{1,0}^0, \dots, b_0^0, b_1^1, \dots) = \frac{1}{2N} \sum_{k=0}^{N-1} (y_k^e - y_k^L)^2$$

$w_{i,j}^l$  represents the weight at layer  $l$  coming from the previous layer at index  $i$  and going to layer  $l$  at index  $j$

$b_j^l$  represents the bias of the node at index  $j$  at layer  $l$

$y_i^e$  is the actual value at index  $i$  and  $y_i^L$  is the computed output value at index  $i$

The cost function is derived from a common linear regression model (least squares regression line) and the  $\frac{1}{2}$  comes from making the derivative simpler.

The direction of greatest decrease of the cost function is given by

$$\vec{u} = -\nabla C$$

where the gradient is the partial derivative of the cost function with respect to each variable. Then to adjust the weights/biases, add the gradient descent vector multiplied by a given learning rate,  $\alpha$ , to the vector of the weights and biases.

$$w_{i,j} := w_{i,j} + \alpha \frac{\partial C}{\partial w_{i,j}}$$

$$b_j := b_j + \alpha \frac{\partial C}{\partial b_j}$$

### 4 Backpropagation

The idea behind backpropagation is that you can think of each node in each layer (including output) as a function of nodes from previous layers.

For now, let us assume that each node is only impacted by one node from a previous layer.

For the first example, let us also assume that we are going one layer deep.

Then, if at the final output layer,  $L$  is the final layer,  $w^L$  = weight,  $b^L$  = bias,  $y^L$  = computed output,  $y^{L-1}$  = previous node,  $y^e$  = expected output, and  $C(w^L, b^L)$  = the cost function, then

$$C(w^L, b^L) = \frac{1}{2}(y^e - y^L)^2 = \frac{1}{2}(y^e - \sigma(w^L y^{L-1} + b^L))^2$$

Let us also say that  $a^l$ , generally, is the linear output of the previous layer, meaning

$$a_j^l = \sum_{k=0}^{len(l)} w_{k,j}^l * y_k^{l-1} + b_j^l \quad (4.0.1)$$

and in our example case,

$$C(w^L, b^L) = \frac{1}{2}(y^e - \sigma(a^L))^2$$

In our one node case, the partial with respect to any given weight at any layer, by the chain rule, is therefore given by

$$\frac{\partial C}{\partial w^l} = \frac{\partial C}{\partial a^l} \frac{\partial a^l}{\partial w^l} = \delta^l \frac{\partial a^l}{\partial w^l} \quad (4.0.2)$$

The term  $\frac{\partial C}{\partial a_\alpha^l}$  is called the error term and denoted  $\delta^l$

So the partial derivative of the weight at the last layer is

$$\begin{aligned} \frac{\partial C(w^L, b^L)}{\partial w^L} &= \frac{\partial C}{\partial a^L} \frac{\partial a^L}{\partial w^L} \\ &= \delta^L y^{L-1} \end{aligned}$$

## 4.1 Derivative of the Sigmoid Function

Now in order to find  $\delta^L$ , we will have to find the derivative of the sigmoid function.

$$\frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1 - e^{-x})^2}$$

With  $u = e^{-x}$ ,

$$\frac{u}{(1-u)^2} = \frac{A}{1-u} + \frac{B}{(1-u)^2}$$

$$B(1-u) + A = u$$

$$\Rightarrow B = -1, A = 1$$

then

$$\frac{\partial \sigma(x)}{\partial x} = \frac{1}{1 - e^{-x}} - \frac{1}{(1 - e^{-x})^2}$$

$$\begin{aligned}
&= \frac{1}{1 - e^{-x}} \left( 1 - \frac{1}{1 - e^{-x}} \right) \\
&= \sigma(x)(1 - \sigma(x))
\end{aligned}$$

## 4.2 Output Layer

Using the equation we got in the previous section, we can substitute in the future and we find that the error term for the output layer is

$$\begin{aligned}
\delta^L &= \frac{\partial C(w^L, b^L)}{\partial a^L} = \frac{\partial C(w^L, b^L)}{\partial \sigma(a^L)} \frac{\partial \sigma(a^L)}{\partial a^L} \\
&= -(y^e - y^L) \sigma'(a^L)
\end{aligned} \tag{4.2.2}$$

Then the partial of the cost function with respect to  $w^L$  is

$$\frac{\partial C(w^L, b^L)}{\partial w^L} = -y^{L-1} (y^e - y^L) \sigma'(a^L) \tag{4.2.2}$$

$$a^L = w^L y^{L-1} + b^L \text{ so } \frac{\partial a^L}{\partial w^L} = y^{L-1}$$

The partial with respect to  $b^L$  is similarly found to be

$$\frac{\partial C(w^L, b^L)}{\partial b^L} = -(y^e - y^L) \sigma'(a^L) \tag{4.2.3}$$

$$a^L = w^L y^{L-1} + b^L \text{ so } \frac{\partial a^L}{\partial b^L} = 1$$

## 4.3 Adding Multiple Layers

If we add more layers (with one node) to our network, then remembering our two recursive equations, (4.0.1) and (4.0.2), we have

$$a^{L-1} = w^{L-1} y^{L-2} + b^{L-1}$$

$$\frac{\partial C}{\partial w^{L-1}} = \delta^{L-1} \frac{\partial a^{L-1}}{\partial w^{L-1}}$$

$$\frac{\partial a^{L-1}}{\partial w^{L-1}} = y^{L-2}$$

Furthermore, using the chain rule, we can find  $\delta^{L-1}$

$$\delta^{L-1} = \frac{\partial C}{\partial a^{L-1}} = \frac{\partial C}{\partial a^L} \frac{\partial a^L}{\partial a^{L-1}} = \delta^L \frac{\partial a^L}{\partial a^{L-1}}$$

We derive the second part as follows

$$a^L = w^L y^{L-1} + b^L = w^L \sigma(a^{L-1}) + b^L$$

$$\frac{\partial a^L}{\partial a^{L-1}} = w^L \sigma'(a^{L-1})$$

Furthermore, if we work backwards from the output layer as described by backpropagation, we should already know  $\delta^L$  so we know

$$\delta^{L-1} = \delta^L w^L \sigma'(a^{L-1})$$

With both  $\delta^{L-1}$  and  $\frac{\partial a^{L-1}}{\partial w^{L-1}}$ , we then know  $\frac{\partial C}{\partial w^{L-1}}$ . This methodology can be repeated recursively for any hidden layers in our neural networks because we are simply adding more recursive definitions. We arrive at the following equation

$$\frac{\partial C}{\partial w^l} = \delta^l \frac{\partial a^l}{\partial w^l} = \delta^{l+1} w^{l+1} \sigma'(a^l) y^{l-1} \quad (4.3.1)$$

#### 4.4 Adding Multiple Nodes

However, the above equations are only true for one node. We can generalize this to multiple nodes by adding terms. Below is equation (4.0.2) with one node per layer turned into an equation with multiple nodes per layer. Each time we add nodes, we know that it contributes to the derivatives as a summation because the derivative is a linear function.

$$\frac{\partial C}{\partial w^l} = \frac{\partial C}{\partial a^l} \frac{\partial a^l}{\partial w^l} \Rightarrow \frac{\partial C}{\partial w_{i,j}^l} = \frac{\partial C}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{i,j}^l} = \frac{\partial C}{\partial a_j^l} y_i^{l-1}$$

In the above equation, the second term will remain the same because even though  $a_{i,j}^l$  is a linear sum, its derivative is only with respect to one variable. However, because  $a_{i,j}^l$  impacts all the nodes in the next layer,  $\delta^{l+1}$  and  $w^{l+1}$  change and contribute to the derivative as a summation (derivative is linear and error is computed as a sum).

$$\delta_j^l = \sum_{k=0}^{len(l+1)} \frac{\partial C}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial w_{j,k}^l} \quad (4.4.1)$$

And by remembering equation (4.3.1),

$$\begin{aligned} \frac{\partial C}{\partial w_{i,j}^l} &= y_i^{l-1} \sum_{k=0}^{len(l+1)} \delta_k^{l+1} w_k^{l+1} \sigma'(a_j^l) \\ &= \sigma'(a_j^l) y_i^{l-1} \sum_{k=0}^{len(l+1)} \delta_k^{l+1} w_{j,k}^{l+1} \end{aligned} \quad (4.4.2)$$

and if  $[\delta]_j^{l+1}$  and  $[w]_j^{l+1}$  are vectors at layer  $l+1$ , then

$$= \sigma'(a_j^l) y_i^{l-1} ([\delta]^{l+1} \cdot [w]_j^{l+1})$$

Similarly, because  $\delta_j^l$  remains the same, the partial for the bias is given by

$$\frac{\partial C}{\partial b_j^l} = \sigma'(a_j^l) ([\delta]^{l+1} \cdot [w]_j^{l+1}) \quad (4.4.3)$$

## 4.5 Final Procedure

Before we start the final procedure, it is best to note that we should try to simplify all the values into vectors as it allows us to use linear algebra to express each equation in shorthand. The final procedure for backpropagation is first calculating the output layers' errors and partials. We find this by simply adding more nodes to equations (4.2.1), (4.2.2), (4.2.3)

$$\delta_j^L = -(y_j^e - y_j^L)\sigma'(a_j^L)$$

$$\frac{\partial C}{\partial w_{i,j}^L} = -y_i^{L-1}\delta_j^L$$

$$\frac{\partial C}{\partial b_j^L} = \delta_j^L$$

Now let us express  $[\delta^L]$  as a the vector of all error terms in the last layer have all the other vectors are similarly denoted. Using the Hadamard Product (element-wise multiplication), the first equation can be expressed as

$$[\delta^L] = -([y^e] - [y^L]) \odot \sigma'([a^L])$$

For the rest of the equations, you can simply multiply each  $-y_i^{L-1}$  by  $[\delta^L]$  and stack each vector to find  $[\frac{\partial C}{\partial w^L}]$

Next, we work backwards starting from layer  $L - 1$  using equations (4.4.1), (4.4.2), and (4.4.3) and the Hadamard product for

$$[\delta^l] = \sigma'([a^l]) \odot [[\delta^{l+1}] \cdot [w^{l+1}]]$$

$$\left[ \frac{\partial C}{\partial w_i^l} \right] = y_i^{l-1}[\delta^l]$$

$$\left[ \frac{\partial C}{\partial b^l} \right] = [\delta^l]$$

Finally, average these partials over the entire dataset

$$\frac{\partial C_{1 \rightarrow N}}{\partial w_{i,j}^l} = \frac{1}{N} \sum_{p=1}^N \frac{\partial C_p}{\partial w_{i,j}^l}$$

$$\frac{\partial C_{1 \rightarrow N}}{\partial b_j^l} = \frac{1}{N} \sum_{p=1}^N \frac{\partial C_p}{\partial b_j^l}$$

## 4.6 Conclusion

In the our step by step is boiled down to:

- Vectorize training data (input and output) across dataset
- Initialize random weights and biases
- Run one pass of the neural network to get guessed output
- Use weights, biases, outputs at each layer, and expected layer and equations in (4.5) to find the change to weights and biases to optimize the cost function

- Apply the change by multiplying the partial by a fixed learning rate and adding it to the existing variable
- Repeat the process a given number of times