Neural Network Summary

Jaden You

December 22, 2023

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 preivous 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 laye (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)

$$C(w^0_{0,0}, w^0_{0,1}, \dots, w^0_{1,0}, \dots, b^0_0, b^1_1, \dots) = \frac{1}{2n} \sum_{k=0}^{n-1} (y^e_k - y^L_k)^2$$

$w^l_{i,j}$ represents the weight at layer l coming from the previous layer at index i and going to layer l at index j
b^l_j represents the bias of the node at index j at layer l
y^e_i is the actual value at index i and y^L_i 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

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

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 to the vector of the weights and biases.

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 ouput of the previous layer, meaning

$$a^l_j = w^l_{i,j} * y^{l-1}_i + b^l_j \tag{1}$$

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

$$\frac{\partial C}{\partial w^l} = \frac{\partial C}{\partial a^l}\frac{\partial a^l}{\partial w^l} = \delta^l \frac{\partial a^l}{\partial w^l} \tag{2}$$

The term $\frac{\partial C}{\partial a^l_\alpha}$ is called the error term and denoted δ^l

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

$$\frac{\partial C(w^L, b^L)}{\partial w^L} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial w^L}$$

$$= \delta^L y^{L-1}$$

4.1 Derivative of the Sigmoid Function

Now in order to find δ^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

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

$$= \frac{1}{1-e^{-x}}\left(1 - \frac{1}{1-e^{-x}}\right)$$

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

That means that the error term is

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

$$= -(y^e - y^L)(y^L)(1-y^L)$$

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

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

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

$$\frac{\partial C(w^L, b^L)}{\partial b^L} = -(y^e - y^L)(y^L)(1 - y^L) \tag{3}$$

4.2 Adding Multiple Layers

If we add more layers (with one node) to our network, then remembering our two recursive equations, (1) and (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 δ^{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 backwords from the output layer as described by backpropagation, we should already know δ^L so we know

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

With both δ^{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

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

4.3 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 (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.

$$\frac{\partial C}{\partial w^l} = \frac{\partial C}{\partial a^l}\frac{\partial a^l}{\partial w^l} \Rightarrow \frac{\partial C}{\partial w^l_{i,j}} = \frac{\partial C}{\partial a^l_{i,j}}\frac{\partial a^l_{i,j}}{\partial w^l_{i,j}} = \frac{\partial C}{\partial a^l_{i,j}} y^{l-1}$$

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

$$\frac{\partial C}{\partial w^l_{i,j}} = y^{l-1} \sum_{k=0}^{len(l+1)} \frac{\partial C}{\partial a^{l+1}_k}\frac{\partial a^{l+1}_k}{\partial a^l_j}$$

And by remembering equation (4),

$$= y^{l-1} \sum_{k=0}^{len(l+1)} \delta^{l+1}_k w^{l+1}_k \sigma'(a^l)$$

$$= \sigma'(a^l) y^{l-1} \sum_{k=0}^{len(l+1)} \delta^{l+1}_k w^{l+1}_k$$