This site was designed with the **WIX** .com website builder. Create your website today. Start Now

WHAT DO NEURAL NETWORKS DO

Introduction

Neural Networks are one of the most popular machine learning algorithms, but as it comprises of many layers of interconnected neurons, sometimes it can be hard to interpret what they are

In this blog, we will try to understand a *neural network model* by breaking it down into components. We will analyze the significance of each component and see how they are contributing to the final output.

Table of content

1. Neural Network
2. Weights
3. Biases
4. Activation Function
5. Dataset (regression)
6. Model with sigmoid activation.
8. Model with relu activation
9. Comparison between ReLU and Sigmoid Functions
10. Extrapolation
11. Hidden Layers
12. Dataset (classification)
13. Model with sigmoid activation
14. Model with relu activation
15. Conclusion

What we will not cover

1. Training a model
2. Backpropagation, Gradient Descent, Optimization
3. Vanishing and Exploding gradient
4. Dropouts

Prerequisites

Familiarity with these topics will make it easier for you to understand this blog.

1. Linear Algebra
2. Basic understanding of machine learning models
3. Reading Graphs

Neural Networks

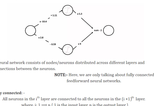
Before diving into the details of the model, let's see what a neural network is.

A *neural network* consists of nodes/*neurons distributed across different layers* and connections between the neurons.

NOTE: Here, we are only talking about fully connected feedforward neural networks.

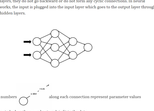
Fully connected :-

All neurons in the ith layer are connected to all the neurons in the $(i+1)^{th}$ layer where, $i = 1 \Longrightarrow 1$ is the input layer, n is the output layer $\}$



Feed Forward:

The model is called feedforward because the information travels only in one direction within the layers, they do not go backward or do not form any cyclic connections. In neural networks, the input is plugged into the input layer which goes to the output layer through the hidden layers.



The numbers $\xrightarrow{\text{edge}}$ along each connection represent parameter values

estimated when the neural network is fit to the data.

NOTE: A neural network starts with random parameter values and then they are modified after each epoch when we fit it to a data by a method called **Backpropagation.**

Weights

To get the input from one neuron to the other, the connection first multiplies a number to it, the number is known as the weight of the input.

The weights can be interpreted as both ways:

1. SLOPE

Imagine, you want to a shop to buy some chocolates. The cost price of each chocolate is 1 rupee. But the shopkeeper sells them at 1.5 rupees, 50% increase in the price. To buy x bar of chocolate you have to pay 1.5 rupees, to buy 2 you have to pay 3 and so on.

quantity	cost
1	1.5
2	3.0
3	4.5
4	6.0
5	7.5
6	9.0
7	10.5
8	12.0
9	13.5
10	15.0

MONEY SPENT = (1 + PROFIT) *(NO.S QUANTITY

It can be thought of as the weight multiplied to the quantity

If we plot the quantity of the chocolate you buy with the corresponding price you have to pay in a graph we will get a straight line passing through the origin.

MONEY SPENT = w * QUANTITY

If the shopkeeper increases the profit rate even more the straight line shifts upward, the slope of the line increases, the slope of the line is increasing, and if he decreases the profit rate the slope also decreases.

NOTE: The rate by which he is increasing the price can also be thought of as the importance he is giving to the chocolate.

2. Scaling Factor

The weight is also the amount by which the input is bigger or smaller, for example if the weights are larger output values and vice versa.

Negative weight gives negative output for a given input. Below, I graphically plot the four opposite doors (same image).

Biases

Now think that the shop is far away from your home and you have to go by bus to reach there. The bus fee is Rs. 5 if you go to the shop whether you buy chocolate or not you have to spend 5 rupees.

The extra 5 rupees of the bus fare shifted the line up by 5 units. The bias term can be thought of as the intercept of the straight line.

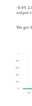
If the bias is positive it shifts the graph up by that unit, and if it is negative it shifts the graph down by that unit.

Activation Functions

The graph inside the node is known as an activation function, it is responsible for adding non-linearity in the output.

Imagine we plug a range of values from [0-1] into the input node, the connection multiplies it by 3 and then adds 1.5. The result of the connection is the x-coordinate of the activation function and the output of the neuron is then corresponding y-coordinate.

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



The input values zoom in the range [0-1] but after multiplying the input with 3 and adding 1.5 to it, we get a narrow range of y axis coordinate values for the activation function, and the range of corresponding y axis values are the output of the neuron (in this case the y axis values form the green curve, similarly for different weights and biases the output curve will look which we will see in next section).

This is how an activation function adds non-linearity to the output. As you can see the yield of the neuron is in the range [0-1] because of the sigmoid activation function, to get the desired output, the result of this neuron (curve) can further be scaled and shifted by weights and biases respectively.

The shape of the curve at all points has also increased, but as the weight multiplied is very small, the change in slope is not clearly visible.

Now that we have learned a few things let's apply them to a real dataset and analyze the output.

Dataset (regression)

The dataset has a column of VARIABLE X, which contain unequally spaced numeric values from the range [-2.7], and another column VARIABLE Y with corresponding y values calculated from the below relation between x and y

$$y = \left(\frac{(x+2)(x-1)}{x+3}\right)^2 + x$$

After plotting the (x,y) points on the plot, this is a graph we get.

NOTE: We already know the relation between X and Y then why we need a neural network model to find it?

The model will only see the data points and try to estimate the relation between them, knowing the relation beforehand will help us to evaluate if the model is learning the actual function or doing something else.

Model with sigmoid activation function.

We have already fit the data to this neural network, which means the parameter values are already estimated

This neural network has one input node where we plug the X values, one output node which tells us the predicted Y value, and 3 hidden nodes between them.

Now let's learn how a Neural Network uses activation function to create different curves and then adds them together to make it into a sensor shape that fits the data.

If we plug the lowest x value, i.e -3 into the input node, to get to the 1st neuron of the hidden layer, the connection multiplies it by -1.68 and adds 4.55.

-3x-1.68 + 4.55 = 9.59

If we increase the input a little bit to -1,

-1x1.68 + 4.55 = 2.87

Similarly if we put the value of 1 into the input node,

1x1.68 + 4.55 = 6.23

-0.49, 2.87, 16.31 are the input to the activation function and to calculate corresponding output values of the neuron, we plug -0.49,2.87 and 16.31 into

$$f(x) = 1/(1 + e^{-x})$$

We get the green curve as the output of the neuron.

Now let's see what the 2nd neuron in the hidden layer does. This time we multiply the input with 1.79 and subtract 8.34 from it to get the input for the activation function. After multiplying weight and adding bias to the input the activation function we get a new green curve, very different from the 1st one.

Similarly for the 3rd neuron, we multiply -0.94, add 1.75 and plug it into the activation function we again get a very different curve.

NOTE: when we plug -3 in the input node the input to the 3rd neuron is

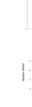
-3x-0.94 + 1.75 = 4.72

and when we plug the highest value

3x-0.94 + 1.75 = -1.16

The actual shape of the curve will be flipped inverse when we plot it vs activation function graph.

Different weights and biases results in using different portions of the activation function to create new curves



This is a graph of the results of the activation function with the x values.

NOTE: - The range of the results of the hidden neurons are in the range [0-1].

The hidden nodes are then connected to the output node(s) with the output node.

To get into the output node results of 1st hidden node is multiplied by 2.2

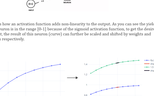
Similarly for 2nd neuron is 24.61 and 3rd by 0.06.

Now neural network adds the y-coordinate values of all the 3 curves.

NOTE: You can think of adding the y values of all curves as **SUPERIMPOSITION of waves.**

After adding all the hidden layer values, we get a blue curve that looks exactly like our dataset, the output of the model is the x values and get the red curve which is the final output of the neural network.

Model with relu activation function.



Now, let's see what happens if instead of the sigmoid activation function we use the relu activation function in the hidden layer

The parameter values are already estimated

If we plug the lowest x value, i.e -3 into the input node, to get to the 1st neuron of the hidden layer, the connection multiplies it by 2.85 and adds 2.85.

-3x2.85 + 2.85 = -5.70

If we increase the input a little bit to 3.5:
$7x2.85 + 2.85 = 22.94$
Similarly if we put the highest value of x in the input we get:
$7x2.85 + 2.85 = 22.94$
-5.66, 0.04, 22.84 are the inputs to activation function and to calculate corresponding output values of the neuron, we plug -5.66,0.04 & 22.84 into:
$f(x) = sin(x)/x$
We get a given level line as the output of the neuron.

Now let's see what the 2nd output neuron does with these values. This time we multiply the input by 1.31 and subtract 1.31 from it to see how the coordinate of the activation function. After multiplying weights and adding biases to activation function we get a new green level line, the horizontal part is more stretched in this one.

Similarly for the 3rd neuron, we multiply by -0.73, add 3.022 and plug it into the activation function to get the green straight line.

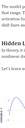
This is a graph of the results of all three neurons with the VARIABLE X values.

NOTE: Unlike the sigmoid function, the outputs or results of the node are not defined in a certain range.

The hidden nodes are then connected (with connections) with the output node. To get into the output node result of 1st hidden node is scaled by 1.5, the result of the 2nd neuron by 4.54, and 3rd by 3.65.

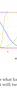
Now second neuron gets the x-coordinate values of all the 3 *x-values*.

Note:- You might have noticed that the scaling factor of the neural network when we were using sigmoid activation was much larger because the output of the sigmoid function is confined between [0-1], whereas the scaling factor of the model with relu activation function is small.



After adding all the hidden layer values, we get a blue curve that looks like something like our data but it's far from perfect. Now we add 2.58 from the y-values and get the red curve which is the final output of the neural network.

Comparision between ReLU and Sigmoid Functions



Although sigmoid activation function provide curves of different shapes better than relu activation function, which only outputs bent and straight lines.

Relu activation functions are more used because they do not have vanishing gradient and exploding gradient problem unlike sigmoid activation functions.

Extrapolation

As we have trained our model with VARIABLE X values which are in the range [-3 7], the model has learned the parameter values according to the given input range.
What if we extend the values from [-3 7] to [-5 10] and see what the model predicts.

The model perfectly fits the data in the range [-3 7] but made terrible predictions outside that range. This implies that a neural network does not learn a particular function but uses activation functions for curves and bent lines and weights and biases to scale squash, and shift lines and curves in a given input range to fit the data.

Hidden Layers

In theory, it is proved that one hidden layer with enough neurons can fit any complex nonlinear data. Then why we need so many hidden layers for some of the tasks??

Let's learn why with an example.

This is a more complex curve than previous one and for this we will need more neurons in the hidden layer.

If we use only one hidden layer, we will need 18 nodes, the output of all the nodes is shown in the graph. These curves are combined, scaled and added to get the final shape.

Let's see what happens if instead of one layer, we train our data on the neural network with two hidden layers with 4 and 3 nodes respectively.

These are the outputs of all the 4 nodes in the 1st hidden layer.

These are the outputs of all the nodes in the 2nd layer. The nodes in the second layer tries to learn more complex shapes in the second layer.

The curves have become more complicated and they are multiplied with their respective weights (to get to the output node).

The result of all 3 nodes of the second layer is added and shifted to fit the data.

If we only use a single layer to get the curve we need 18 nodes but if we train the same dataset on 2 hidden layer network with 4 and 3 nodes respectively we can get the same shape with a reduction of 3 nodes compared to one hidden layer network.

Less computation power is needed to train the network.

In practice, the shapes are more complex and a single hidden layer will need a large number of nodes to approximate it, on the other hand, hidden layers will decrease the nodes needed and also the computation needed

Dataset (classification)

The model will take the x and y coordinates of a point and with that information, it will predict whether it is a particular class label or not.

Model with sigmoid activation function

This *neural network* contains two 2 nodes for plugging x and y coordinate values respectively two *output* nodes where we get if the input is a particular class label or not, and a single hidden layer
The input nodes will map to both the output nodes, so let's analyze both the output node one by one.

Since we have two input nodes, each gets an axis x-axis is for the x-coordinate of the input, the y-axis is for the y-coordinate of the input and on the z-axis we will analyze the results of each node.

For all combinations of X and Y in the range [-7.5 10] and [10 0] (coordinates of the input) we get the [-5.5, 1] to 0.1, and then with 0.11, and then pass it through the sigmoid activation function to get the output we get this colorful curve, where yellow represents the highest value of z and purple lowest.

This is the output of the 1st output node which predicts if the data point is class label 0 or not. To get to the 1st output node from the hidden node the connection multiplies it by -5.2 (this means the output of the hidden node is flipped from the x-axis to the origin and scaled 5.2 times), and adds 4.6 to it.

You can see that the z value are higher when both (X, Y) points are positive. This shows that this output node detects the class label 0.

This is a contour plot of the 1st output node, to visualize what is happening in 2D.

This is the output of the 2nd output node which predicts if the data point is the class label 1 or not. To get to the 2nd output node from the hidden node the connection multiplies it by 5.3 (the result of the hidden neuron is just scaled 5.56 times) and subtracts 4.6 from it. You can see that the z values are higher when both (X, Y) points are negative. This shows that this output node detects the class label 1.

This is a contour plot of the 2nd output node, to visualize what is happening in 2D.

Model with relu activation

I have trained the same classification data on a neural network with relu activation function.

MODEL

OUTPUT OF HIDDEN LAYER

OUTPUT OF 1st OUTPUT NODE

CONTOUR PLOT OF 1st OUTPUT NODE TO VISUALIZE IN 2D

OUTPUT OF 2nd OUTPUT NODE

CONTOUR PLOT OF 2nd OUTPUT NODE TO VISUALIZE IN 2D

NOTE:- As you can see from the results of output nodes, the raw output values are not always between 0 & 1.
This broad range of values make the raw output values harder to understand, because of this the raw output values are sent to a sofmax layer.
The softmax layer makes the output value in the range [0-1].

Conclusion

In practice, the neural networks are deeper, are spread across a broader dimension, and much more complex but they are still made up of these same components used in simple networks.