# Neural Network From Scratch

Emmanuel Misley

March 2023

## 1 Neural Network From Scratch

This document shows a detailed description of the computations needed to implement a neural network from scratch.

### 1.1 Forward Propagation

#### 1.1.1 What does each perceptron do? (hidden layer 1)

The computation of $h^{(1)}$ implies the following linear combination between the inputs and the weights asociated with each perceptron

$$h^{(1)} = \left[h_1^{(1)}, h_2^{(1)}, h_3^{(1)}\right] \implies \begin{cases} h_1^{(1)} = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} \\ h_2^{(1)} = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} \\ h_3^{(1)} = x_1 w_{13} + x_2 w_{23} + x_3 w_{33} \end{cases}$$

as can be seen, this linear combination is focused on the inputs received by each node. In other words, it views each perceptron as an isolated system, so to speak. Therefore, these and the following operations are focused on the operations carried out by each perceptron in the neural network.

$$\begin{aligned} h^{(1)} = \mathbf{x} W^{(1)} &= [x_1, x_2, x_3] \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \left[h_1^{(1)}, h_2^{(1)}, h_3^{(1)}\right] \\ &= [0.3, 0.7, 0.5] \begin{bmatrix} 0.2 & 0.1 & 0.9 \\ 0.5 & 0.1 & 0.9 \\ 0.1 & 0.5 & 0.6 \end{bmatrix} = [0.46, 0.35, 1.2] \end{aligned}$$

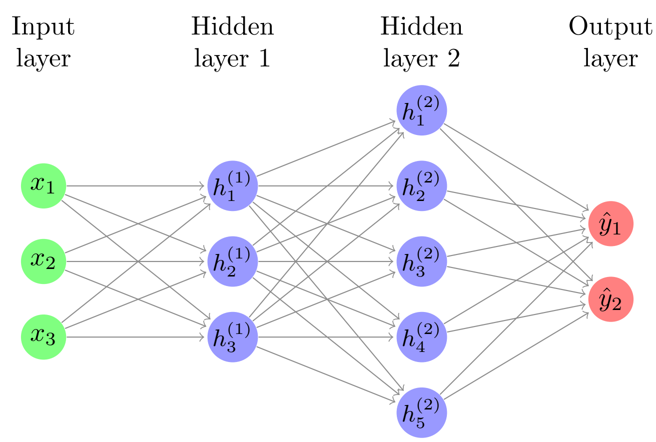


Figure 1: Artificial Neural Network

So the next step is using the activation function, in this case the sigmoid activation fuction is used.

$$f(z) = \frac{1}{1+e^{-z}}$$

the results were rounded up to 3 decimal places

$$\begin{aligned}
a^{(1)} = f\left(h^{(1)}\right) &= \left[\frac{1}{1+e^{-h_1^{(1)}}}, \frac{1}{1+e^{-h_2^{(1)}}}, \frac{1}{1+e^{-h_3^{(1)}}}\right] \\
&= \left[\frac{1}{1+e^{-0.46}}, \frac{1}{1+e^{-0.35}}, \frac{1}{1+e^{-1.2}}\right] \\
&= \left[0.613, 0.587, 0.769\right]
\end{aligned}$$

finally, that is the output of the first hidden layer, where each component of the vector $a^{(1)}$ is the output of each perceptron.

### 1.1.2 What does the rest of perceptrons do? (hidden layer 2)

As all hidden layers are connected the computation of $h^{(2)}$ uses the vector $a^{(1)}$

$$\begin{aligned}
h^{(2)} = a^{(1)}W^{(2)} &= \left[a_1^{(1)}, a_2^{(1)}, a_3^{(1)}\right] \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \end{bmatrix} \\
&= \left[0.613, 0.587, 0.769\right] \begin{bmatrix} 0.5 & 0.1 & 0.8 & 0.6 & 0.6 \\ 0.9 & 0.1 & 0.9 & 0.3 & 0.8 \\ 0.4 & 0.9 & 0.4 & 0.8 & 0.2 \end{bmatrix} \\
&= [1.142,\ 0.812,\ 1.326,\ 1.159,\ 0.991]
\end{aligned}$$

then the activation function is used

$$\begin{aligned}
a^{(2)} = f\left(h^{(2)}\right) &= \left[\frac{1}{1+e^{-1.142}}, \frac{1}{1+e^{-0.812}}, \frac{1}{1+e^{-1.326}}, \frac{1}{1+e^{-1.159}}, \frac{1}{1+e^{-0.991}}\right] \\
&= \left[0.758,\ 0.692,\ 0.790,\ 0.761,\ 0.729\right]
\end{aligned}$$

### 1.1.3 The Output Layer

Just as in the previous cases $h^{(3)}$ is calculated as

$$\begin{aligned}
h^{(3)} = a^{(2)}W^{(3)} &= \left[a_1^{(2)}, a_2^{(2)}, a_3^{(2)}, a_4^{(2)}, a_5^{(3)}\right] \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \\ w_{51} & w_{52} \end{bmatrix} \\
&= \left[0.758,\ 0.692,\ 0.790,\ 0.761,\ 0.729\right] \begin{bmatrix} 0.9 & 0.5 \\ 0.4 & 0.1 \\ 0.1 & 0.1 \\ 0.7 & 0.5 \\ 0.8 & 0.7 \end{bmatrix} \\
&= \left[2.154, 1.418\right]
\end{aligned}$$

and then the activation function

$$\begin{aligned}
a^{(3)} = f\left(h^{(3)}\right) &= \left[\frac{1}{1+e^{-a_1^{(4)}}}, \frac{1}{1+e^{-a_2^{(4)}}}\right] \\
&= \left[\frac{1}{1+e^{-2.154}}, \frac{1}{1+e^{-1.418}}\right] \\
&= \left[0.896,\ 0.805\right]
\end{aligned}$$

### 1.1.4 Code

the following code shows how to implement the neural network from scratch in python 3.

```
import numpy as np
class MLP:
    def __init__(self,num_imputs,num_hidden,num_outputs,weights):
        self.num_inputs = num_imputs # num_inputs atribute is created
        self.num_hidden = num_hidden # num_hidden atribute is created
        self.num_outputs = num_outputs # num_outputs atribute is created
        self.weights = weights # weights atribute is created

    def forward_propagate(self, inputs):
        """
        Method that implements the forward propagation
        """
        activations = inputs
        for w in self.weights:
            net_inputs=np.dot(activations, w)
#applies the dot product between weights and inputs
            activations = self.sigmoid(net_inputs) #applies the activation function
        return activations

    def sigmoid(self,x):
        """
        Method that implements sigmoid function
        """
        return 1/(1+np.exp(-x))
```

Figure 2: Neural Network from scratch code part 1

```python
if __name__ == "__main__":
    #The Multi Layer Perceptron (MLP) parameters are provided
    num_imputs=3
    num_hidden=[3, 5]
    num_outputs=2
    weights = [[[0.2, 0.1, 0.9],
                [0.5, 0.1, 0.9],
                [0.1, 0.5, 0.6]],
               [[0.5, 0.1, 0.8, 0.6, 0.6],
                [0.9, 0.1, 0.9, 0.3, 0.8],
                [0.4, 0.9, 0.4, 0.8, 0.2]],
               [[0.9, 0.5],
                [0.4, 0.1],
                [0.1, 0.1],
                [0.7, 0.5],
                [0.8, 0.7]]]

    mlp = MLP(num_imputs,num_hidden,num_outputs,weights)
    inputs = [0.3,0.7,0.5]
    outputs = mlp.forward_propagate(inputs) #excute the forward propagation

    #imprimo los resultados
    print(f"The network input is: {inputs}")
    print(f"The network output is: {outputs}")
```

Figure 3: Neural Network from scratch code part 2