Training Report Day-1

Ankur (URN: 2302473)
Department of Computer Science and Engineering
Guru Nanak Dev Engineering College Ludhiana

June 26, 2025

Abstract

A neural network is a multilayer connected network of neurons that we use to make predictions. A neural network mainly consists of 3 parts: An input layer, some (or many) hidden layers, and an output layer (a.k.a. prediction of the model). The input taken at the input layer goes through the complex network of hidden layers. At each hidden layer, the Neural Network takes input from the previous layer, applies some calculations, and forwards the result to the next layer in the network. In the end, an error in the output is calculated. This error is then propagated back into the network to identify and adjust the neurons that lead to the error. Thus, the network starts to learn.

1. The model makes a prediction.
2. Calculates the error in the prediction
3. Takes necessary steps to reduce the error.

1 A Neural Network

Neural networks are the fundamental building blocks of deep learning algorithms. A neural network is a type of machine learning algorithm that is designed to simulate the behavior of the human brain. It is made up of interconnected nodes, also known as artificial neurons, which are organized into layers.

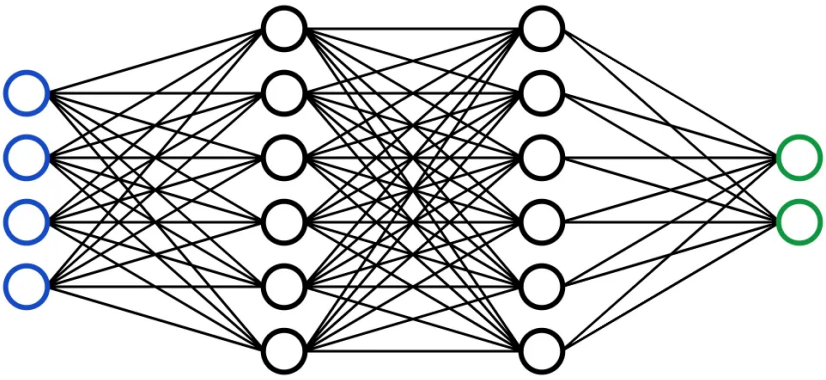

Figure 1: A neural network — Source

The blue nodes in the figure above represent the input layer, the black ones are the hidden layers, and the green ones are the output layers.

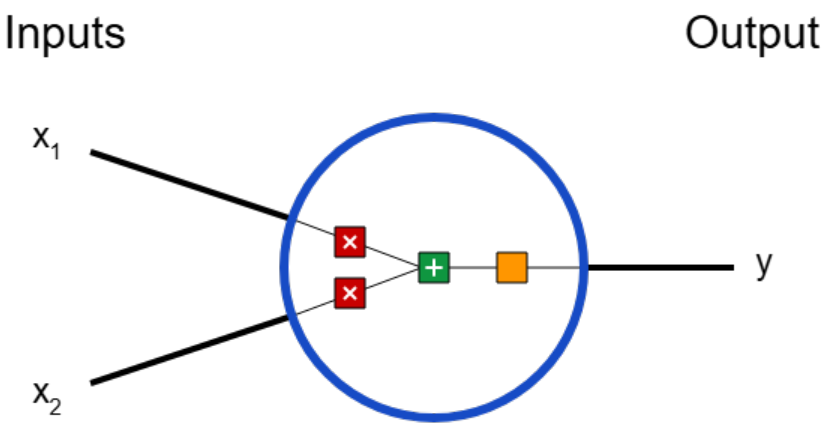

Figure 2: A neural network — Source

A neuron takes an input, does some math on it, and gives the output. Consider the following example:

3 things are happening here. First, each input is multiplied by a weight: ■

$$x_1 \rightarrow x_1 * w_1$$

$$x_2 \rightarrow x_2 * w_2$$

Next, all the weighted inputs are added together with a bias b: ■

$$(x_1 * w_2) + (x_2 * w_2) + b$$

Finally, the sum is passed through an activation function: ■

$$y = f((x_1 * w_2) + (x_2 * w_2) + b)$$

A simple neuron can be implemented as follows:

```
import numpy as np

# Activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

class Neuron:
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def feedforward(self, inputs):
        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)


weights = np.array([0, 1])
bias = 4

neuron = Neuron(weights, bias)

x = np.array([2, 3])
print(neuron.feedforward(x)) # 0.9990889488055994
```

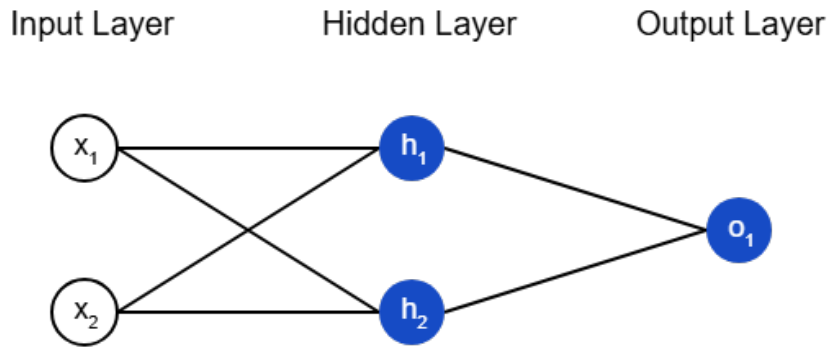

Figure 3: A neural network — Source

Now, using the neuron code, we can create our own small neural network.

```
    import numpy as np

# ... code from the previous section here

class OurNeuralNetwork:
  '''
  A neural network with:
    - 2 inputs
    - a hidden layer with 2 neurons (h1, h2)
    - an output layer with 1 neuron (o1)
  Each neuron has the same weights and bias:
    - w = [0, 1]
    - b = 0
  '''
  def __init__(self):
    weights = np.array([0, 1])
    bias = 0

    # The Neuron class here is from the previous section
    self.h1 = Neuron(weights, bias)
    self.h2 = Neuron(weights, bias)
    self.o1 = Neuron(weights, bias)

  def feedforward(self, x):
    out_h1 = self.h1.feedforward(x)
    out_h2 = self.h2.feedforward(x)

    # The inputs for o1 are the outputs from h1 and h2
    out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

    return out_o1

network = OurNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x)) # 0.7216325609518421
```

2 Some terminologies linked with Neural Networks

2.1 Loss

The **loss** is a metric to quantify **how good a model is doing** so that it can be tweaked to do better. It is a number that we try to reduce over successive passes over our training data. This is termed as training a neural network. The loss function is a multivariable function of weights and biases. We can tweak these weights and biases to reduce the loss. The loss functions can be of many types, one such commonly used loss function is **Mean Squared Loss** or MSE.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

- n is the number of samples
- y is the variable being predicted
- y_{true} represents the correct value of the variable
- y_{pred} is whatever value the neural network predicted

2.2 Backpropagation

Now, suppose that we want to see how the loss (L) will be affected by changing, say, the weight(w_1). We can do so by calculating the partial derivative of the L w.r.t w_1, i.e. $\frac{\partial L}{\partial w_1}$. When we do the calculations then we move backwards in the neural network from prediction towards the input layer. This system of calculating partial derivatives by moving backward is known as the **backpropagation**. Backpropagation is the technique that helps us reduce the loss. By calculating the gradient of the loss function, backpropagation allows the neural network to update its weights in a way that reduces the overall error or loss during training.

2.3 Weight

A **weights** refers to the parameters that are learned during the training. It determines the strength of the connections between the neurons. Each connection between the neurons is assigned a weight and inputs are multiplied with the weight to create the output for the next layer in the NeuralNet.

2.4 Bias

It is yet another parameter that is learned during the training. It is added to the weighted sum of the inputs to a neuron in a given layer. It is an additional input to the neuron that helps to adjust the output of an activation function.

2.5 Nonlinear activation function

A non-linear activation function is used in a neural network to introduce non-linearity in the model so that the model can learn complex, non-linear relationships between the inputs and outputs. Without this, the model will start remembering instead of learning. The common activation functions include **sigmoid**, **ReLU** and **softmax**

2.6 Optimization

Optimization refers to the minimization of the errors incurred by the neural network while making predictions. It is done by adjusting the weights of the neural net. The examples include **Adam** and **Stochastic Gradient Descent**