# An Introduction to Neural Networks

To many, including myself, Neural Networks can be difficult to wrap your head around. In Python, one of the most popular Machine Learning languages, there are many different libraries, of inbuilt classes and objects capable of simplifying the creation of neural networks.

In this document, we will aim to analyse and cover the basics of machine learning within Python using several example programs. I would also recommend checking out [3Blue1Brown’s video series](https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi) on Neural networks if you want a deeper look at the mechanisms behind these complicated systems.

## 2 Layer Neural Network

### Introduction

This file uses the [NumPy module](https://numpy.org/doc/stable/user/absolute_beginners.html) to work with multi-dimensional arrays, which display the program's inputs and outputs, to predict the result of an [XNOR gate](https://www.geeksforgeeks.org/xnor-gate/) input. (An XNOR gate is the combination of an [XOR gate](https://www.geeksforgeeks.org/xor-gate/) and a [NOT gate](https://www.geeksforgeeks.org/not-gate/) (Minecraft Redstone Reference!). An XNOR gate will give a True output (1) when all the inputs are the same, i.e. all True or all False, and a False output (0) otherwise, when the values are different.

### Line by Line

The program works using two sets of arrays, X and y. The X array is the array of possible inputs to the system, 3 True or False values stored as floats, and the y array has the system outputs for each input, both declared using NumPy’s array function.

import numpy as np

X = np.array(([0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]), dtype=float)

y = np.array(([1], [0], [0], [0], [0], [0], [0], [1]), dtype=float)

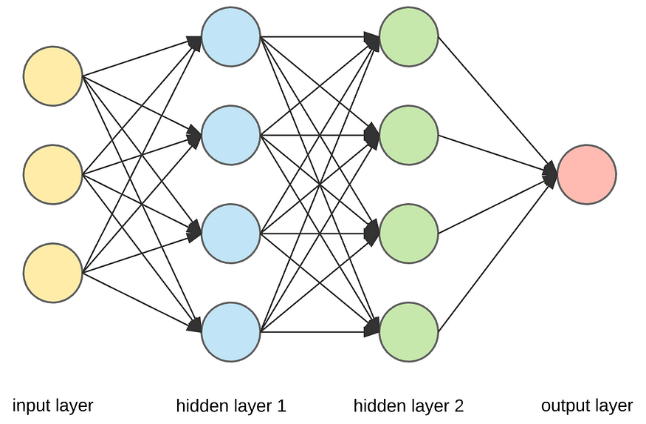
The xPredicted array is the input layer of the system and the predicted output should hence be 0.

xPredicted = np.array(([0, 0, 1]), dtype=float)

For this program, a Neural Network class is defined to ensure good code practice and allow for abstraction, as the final main program loop will thus be simple, only a series of functions within a for loop where the source code is hidden behind the function and class names.

class Neural\_Network (object):

#### A brief tangent about the structure of a neural network

A neural network is a large collection of interconnected functions, known as **neurons**, that perform algorithms on given data. Each neuron is responsible for one individual algorithm and will pass on the result deeper into the network.

Neural networks consist of three main layers of neurons, the input layer, the hidden layer, and the output layer.

The input layer is responsible for data put into the system. The more complex the data, the more input layer **neurons** are required. For example, our first program only requires 3 input layer neurons, one for each digit in the XNOR gate input, while an image may require thousands of input layer pixels.

The hidden layer is the “brain” of the neural network where all the calculations are performed on the input data to create the output data. There will likely be more than one layer where calculations are performed, and the more calculations, the more complicated the code will be, and the longer the program will run, however the more complex the program will be.

The output layer is the display, that is, what the user receives. For chatbots, this is a series of 1s and 0s which make up the ASCII symbols for the text given back to the user. For image generators, these are also 1s and 0s representing information about the pixels displayed to the user in binary, such as the HEX code and position.

At the beginning of the program, when the class is initialised down at the end of the program, the input layer size, output layer size, and hidden layer sizes are all set to be 3, 1, and 4, where the size is the number of *neurons*.

What this means is that the size of the input layer is 3, the true and false values that make up the input array, the output layer will be the size of the program output, the single true or false value, and the hidden layer size is the intermediate layer where the calculations are done.

The hidden layer is the part of the neural network responsible for processing and transforming the input data, enabling it to be outputted and learnt from. The hidden layer size is important as it determines how many functions are performed on the data, and thus its complexity. However, more hidden layers will result in a longer program runtime.

    def \_\_init\_\_(self):

        self.inputLayerSize = 3

        self.outputLayerSize = 1

        self.hiddenLayerSize = 4

        self.W1 = np.random.randn(self.inputLayerSize, self.hiddenLayerSize)

        self.W2 = np.random.randn(self.hiddenLayerSize, self.outputLayerSize)

import numpy as np

X = np.array(([0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]), dtype=float)

y = np.array(([1], [0], [0], [0], [0], [0], [0], [1]), dtype=float)

xPredicted = np.array(([0, 0, 1]), dtype=float)

X = X/np.amax(X, axis=0)

xPredicted = xPredicted/np.amax(xPredicted, axis=0)

class Neural\_Network (object):

    def \_\_init\_\_(self):

        self.inputLayerSize = 3

        self.outputLayerSize = 1

        self.hiddenLayerSize = 4

        self.W1 = np.random.randn(self.inputLayerSize, self.hiddenLayerSize)

        self.W2 = np.random.randn(self.hiddenLayerSize, self.outputLayerSize)

        self.lossFile = open("SumSquaredLossList.csv", "a")

    def feedForward(self, X):

        self.z = np.dot(X, self.W1)

        self.z2 = self.activationSigmoid(self.z)

        self.z3 = np.dot(self.z2, self.W2)

        o = self.activationSigmoid(self.z3)

        return o

    def backwardPropagate(self, X, y, o):

        self.o\_error = y - o

        self.o\_delta = self.o\_error \* self.activationSigmoidPrime(o)

        self.z2\_error = self.o\_delta.dot(self.W2.T)

        self.z2\_delta = self.z2\_error \* self.activationSigmoidPrime(self.z2)

        self.W1 += X.T.dot(self.z2\_delta)

        self.W2 += self.z2.T.dot(self.o\_delta)

    def trainNetwork(self, X, y):

        o = self.feedForward(X)

        self.backwardPropagate(X, y, o)

    def activationSigmoid(self, s):

        return 1/(1+np.exp(-s))

    def activationSigmoidPrime(self, s):

        return s \* (1 - s)

    def saveSumSquaredLossList(self, i, error):

        self.lossFile.write(str(i) + ", " + str(error.tolist())+"\n")

        self.lossFile.flush()

    def saveWeights(self):

        np.savetxt("weightsLayer1.txt", self.W1, fmt = "%s")

        np.savetxt("weightsLayer2.txt", self.W2, fmt = "%s")

    def predictOutput(self):

        print("Predicted XOR output data based on trained weights: ")

        print("Expected (X1 - X3): \n" + str(xPredicted))

        print("Output (Y1): \n" + str(self.feedForward(xPredicted)))

    def closeFile(self):

        self.lossFile.close()

myNeuralNetwork = Neural\_Network()

trainingEpochs = 1000000

for i in range(trainingEpochs):

    print("Epoch # " + str(i) + "\n")

    print("Network Input: \n" + str(X))

    print("Expected Output of XOR Gate Neural Network: \n" + str(y))

    print("Actual Output from XOR Gate Neural Network: \n" + str(myNeuralNetwork.feedForward(X)))

    Loss = np.mean(np.square(y - myNeuralNetwork.feedForward(X)))

    myNeuralNetwork.saveSumSquaredLossList(i, Loss)

    print("Sum Squared Loss: \n" + str(Loss))

    print("\n")

    myNeuralNetwork.trainNetwork(X, y)

myNeuralNetwork.saveWeights()

myNeuralNetwork.predictOutput()

myNeuralNetwork.closeFile()

## Tensor Flow Keras Network