# 2 Layer Neural Network

This file uses the NumPy module to work with multi-dimensional arrays, which can display the program's inputs and outputs, which will predict the result of an XNOR gate input. (An XNOR gate is an XOR gate and a NOT gate. An XNOR gate will give a 1 (True) output when all the inputs are equal.

The program works using two lots of arrays, the X and y arrays. The X array is the array of possible inputs to the system, and the y array includes the system outputs, declared using NumPy.

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)

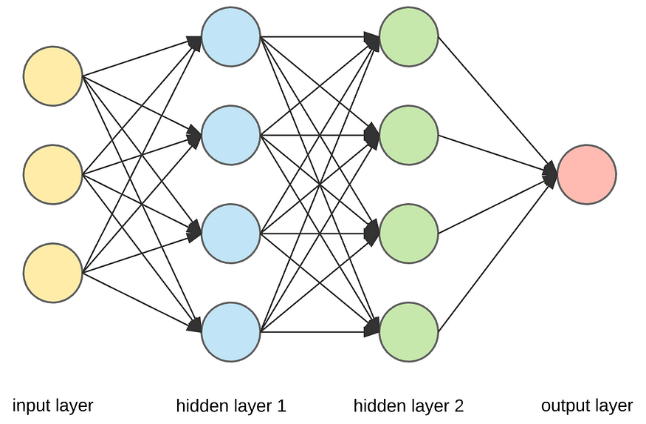
The xPredicted array is the target value of the program and is based on one specific input.

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

For this program, a Neural\_Network class is defined which ensures good code practice and allows for abstraction within the code, as the final main program loop will be very simple and only be a series of functions where the source code is hidden behind the function and class names.

class Neural\_Network (object):

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*.

In machine learning, a neuron can be thought of as a function, which allows for input values to be taken in, a calculation performed on the value, and then an output signal to be sent deeper into the network.

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 that is responsible for the processing and transformation of the input data that enables it to be learnt from and outputted. The size of the hidden layer is important as the

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