PHY3042
Assignment 2
Student - 6463360

# Task 1 – Neural Network

The neural network structure resembles a feed forward network but with 3 hidden nodes, and the input nodes pass to the output node as well as the hidden. This is shown in figure 1.

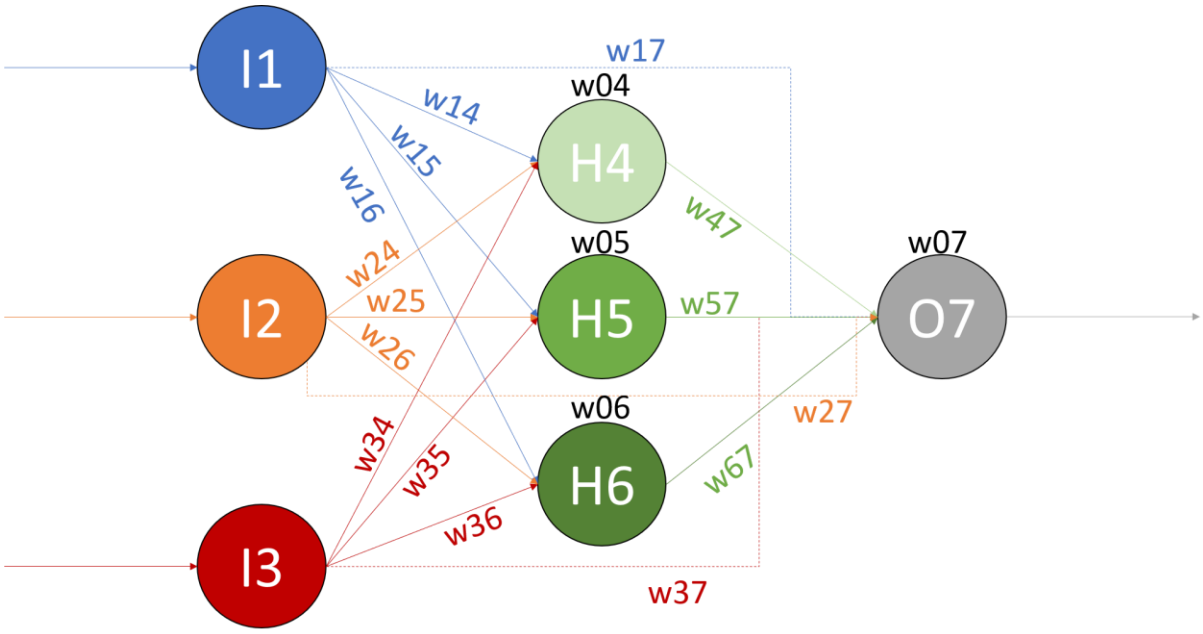


Figure 1 A representation of the structure of the neural network used in task 1. Each line connecting nodes is a neuron with weights associated with it. The faded lines represent the input nodes connection to the output node. The weights for each neuron is printed next to it. The bias weights for the hidden and output nodes is printed above them.

After playing with the beta and k parameters I found that values of 0.01(+1 per run) and 20 respectively were optimal. Using these values, I tried various kinds of neural network structure. I noticed in my trying that more weights led to a faster learning time. Feeding the input nodes to the output and housing a single hidden node proved to be very slow and often not successful in the learning process. I experimented with more nodes but the code began to feel clunky, despite the improved success. I tuned to the network shown in figure 1.

The geometry I chose is shown in figure 1. 3 hidden nodes allows for more weights to be introduced while not adding too many calculations to the programme. Feeding the input nodes to the output node increased the number of weights without substantial detriment to run time. This setup was decided to be the best fit for the learning process.

The weights of each neuron are named on the lines in figure 1, with the weights that are black and sitting on the hidden and output nodes are the bias weights of those nodes. An array of 18 random numbers between -1 and 1 was created in the code that correspond to the weights in the following way:

$$w[0] = w_{04}, w[1] = w_{14}, w[2] = w_{24}, w[3] = w_{34}$$

$$w[4] = w_{05}, w[5] = w_{15}, w[6] = w_{25}, w[7] = w_{35}$$

$$w[8] = w_{06}, w[9] = w_{16}, w[10] = w_{26}, w[11] = w_{36}$$

$$w[12] = w_{07}, w[13] = w_{17}, w[14] = w_{27}, w[15] = w_{37}, w[16] = w_{47}, w[17] = w_{57}, w[18] = w_{67}$$

My choice of weight alteration in the Metropolis Algorithm was to randomly select a weight and replace it with a different random number between -1 and 1. The error generated by the old and new weights was then calculated and compared. If the error of the new weights was smaller it was accepted and the process was run again. If the new error was larger but the probability condition was met then it was accepted anyway, else the old weights remained and the process ran again. This allows the algorithm to try different combinations of random numbers as weights until the error is sufficiently reduced. When the error was below a pre-determined value the condition of the while loop was met and the programme completed. This error value was set as $1e^{-6}$ as it produces the correct results with a small error and in a short time. Increasing this error condition to $1e^{-9}$ proved to increase the run time significantly.

To prevent too much data being printed to the user, an event log file was created to hold the information of the inner workings of the programme. It contains the time of entry, run count, weights and errors (old and new) and whether the new weights were accepted. The truth tables were printed to the user along with some helpful messages.