Acquired Intelligence & Adaptive Behaviour

Artificial Neural Networks
Lab Session 1

Goals:

1. To become familiar with the Matlab programming environment.
2. To implement a simple feed-forward neural network.

Tasks:

A. First we will construct a single layer network with simple threshold neurons, to solve a simple logical function. I'll walk you through this one.

Begin by creating our data sets:

1. Open Matlab and create a new m-file (script). Call it something like 'simpleNN.m'. Add a comment (% ...) at the top that explains who wrote it, when and what it is for.

   Tip: See 'Goals', above.

2. Define a matrix of numbers to represent the inputs and outputs of the logical AND function. Assign it to a suitably named variable (e.g. 'AND').
   I.e.

   ```
    AND_In = [ [0; 0], [1;0], [0;1], [1;1]]
   AND_Out = [0 0 0 1];
   ```

   Familiarise yourself with the syntax and the form of the resulting matrix using the command line interface.

   Tip: You can also 'run' your script in the command window, to ensure it works.

3. Do the same for OR, XOR, NOT and NAND.

Now we can begin constructing the network itself.

4. Create a suitably sized matrix of weights, whose initial values are drawn from a random uniform distribution in the range [0,1].

   Tip: Use the `rand` function to achieve this.

5. Use matrix indices to extract a pair of inputs from the AND matrix as a vector.
   E.g.

   ```
   vec = AND_In(1:2,1);
   ```

   Familiarise yourself with the syntax and the form of the resulting vector using the command line interface.

Next we perform forward propagation to determine how our network behaves.

6. Multiply the weight matrix by this new vector. Notice the difference between using the standard multiplication operator (.*) and the matrix multiplier (*).

   Which is most useful here? Do we need another step?

7. Take the output of your multiplication and apply a linear threshold to it, such that values over 0.5 return 1 and those below return 0. Use an `if...end` statement.

   (What should an input of 0.5 return?)

8. Compare this output to the appropriate value stored in the AND matrix and calculate the error.

9. Implement a for loop around the relevant parts of your code, to produce outputs and errors for all entries in the AND matrix. Check the calculations are correct.

10. Set the weights in your matrix *by hand*, to solve the AND problem.

B. We may now extend what we've learnt to other logical operators.

Try the following:

1. Produce a weight matrix that solves the OR problem.
2. Extend this architecture to handle the NOT problem.

   Tip: You may need a bias node!

3. Plot a graph of your network's output for the inputs

C. Implement a random training regime to adjust the weights in a randomly initialised network, such that it can be taught any one of the linearly separable logical operations.

Tip: You may wish to create a *function* to do this.

D. Add another layer to your network and set weights to solve the XOR problem *by hand*.

E. Extend your algorithm from **C** to allow the XOR operation to be learned.

F. Thresholds are ugly. Implement a sigmoid activation function instead:

$$P(t) = \frac{1}{1+e^{-t}}$$

What further considerations does this require? How is novel input handled?

G(optional). Try to implement a simple shape-recognising network.

Or

Implement the backpropagation algorithm (see me)