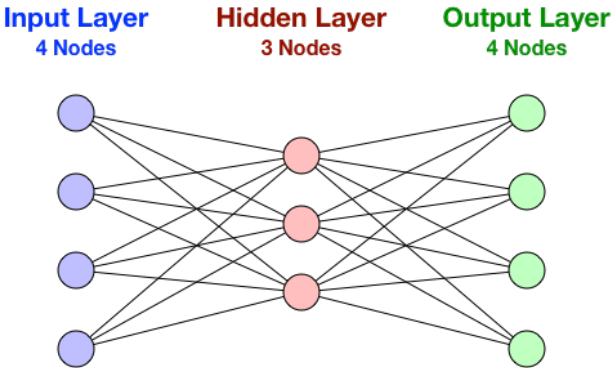
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

Controlling a Game Character Action Using a Neural Network

Overview

In this practical, we will use a neural network to control a game character's actions given a set of different input attributes.



The input layer contains 4 nodes that map an input vector to the following attributes.

1. **Health** (2 = Healthy, 1 = Minor Injuries, 0 = Serious Injuries)
2. Has a **Sword** (1 = Yes, 0 = No)
3. Has a **Gun** (1 = Yes, 0 = No)
4. Number of **Enemies**

For example, the input vector {2, 0, 1, 2} means "healthy, no sword, has a gun and two enemies". The "number of enemies" input value will have to be normalized if its range is large relative to the other input values. In such a case, some nodes in a neural network will have a large impact on the final result, i.e. some parts of a network can be more biased than others. The neural network should map the set of inputs to the output vector {0, 0, 1, 0} that relates to one of the following categories of action, i.e. "Hide".

1. Panic
2. Attack
3. Hide
4. Run

Each action is represented in the neural network by one of the four nodes in the output layer.

Exercises

- Create a new class called ***GameRunner*** and add a ***main()*** method. Inside ***main()***, add the declarations for the 2D arrays ***data*** and ***expected*** from the file **game.txt** in the Zip archive

aiNeuralNetData.zip.

- Create an instance of the class ***NeuralNetwork*** with 4 input nodes, 3 hidden nodes, 4 output node and a sigmoidal activation function:

```
NeuralNetwork nn = new NeuralNetwork(Activator.ActivationFunction.Sigmoid, 4, 3, 4);
```

- Instantiate the back-propagation training algorithm and ask it to train the network with the training data, a learning rate of 0.6 and a maximum of 10000 epochs.

```
BackpropagationTrainer trainer = new BackpropagationTrainer(nn);
trainer.train(data, expected, 0.6, 10000);
```

- Create the following data set required to test if the network is fully trained:

```
int testIndex = 11;
double[] result = nn.process(data[testIndex]);
for (int i = 0; i < expected[testIndex].length; i++){
  System.out.print(expected[testIndex][i] + ",");
}
System.out.println("==>" + (Utils.getMaxIndex(result) + 1));
```

- ***Progressively increase the "number of enemies"*** in the test data by an order of magnitude from {1, 10, 100, 1000, 10000} and examine the output and stability of the network. Normalize the "number of enemies" in both the training and test data and examine the result. You should use the method ***Utils.normalize(double[] vector, double lower, double upper)*** and set the values for low and upper to 0 and 2 respectively.