**Backpropagation**

**Justification**

For my implementation of the back propagation algorithm I utilized the structures provided, converted the 2D arrays into vectors and added 2 classes to the overall solution. A sigmoid class, containing the sigmoid function and its derivative, and a Node class, containing all the relevant structures and process for a given node including its weights and inputs. I did this to simplify the overall implementation of the solution.

Method parameters:

* Training set size 2500
* Number of nodes in:
  + Input layer 2
  + Hidden layer 5
  + Output layer 2
* Learning rate 0.075
* Momentum factor 0.9
* MSE cut-off 0.04

The number of nodes in the inputs and outputs could not be changed. The choice of using 5 hidden nodes was based on the fact the having too many hidden nodes increases processing time and may result in over fitting and having too little and the network may suffer from under fitting and may never learn due to the low number of hidden nodes not being able to accommodate all the training data.

Since I use a momentum factor, which accelerates learning by encouraging weight changes to continue in the same direction with larger steps, it requires a small learning rate and if used in conjunction with a large momentum factor.

I used an MSE cut-off of 0.04 simply because it was the lowest value I could use without taking too long to process. Occasionally some of the training data entries do not converge to the MSE cut-off and as a result if the MSE cut-off is too low some of these sets of training data the MSE for them starts to rise instead of drop.

The sweepers speed are set based on the distance of the closest object and a multiplier which allows them to speed up when they are far from an object and slow down as they approach the object.

**Method**

1. For each training data
   1. Feed inputs forward, from input layer to output layer, using the Sigmoid function as the neural networks activation function. , where is the sum of the products of each weight value and its corresponding input value.
   2. Propagate the error backwards, from output layer to input layer:
      1. Calculate error for each node in the output and hidden layer
      2. Adjust the weights in the output layer, and then the hidden layer. , where is the weight between nodes is the learning rate, is the error signal at the node, is the input for the node (also the output of the node from the previous layer), is the delta weight for the previous node in the same layer and is the momentum factor which was set at 0.9.
2. Calculate global error
3. Repeat from step 1 until global error is below the cut-off value

**Results**

|  |  |  |
| --- | --- | --- |
| Environment | Average Mines Collected | Average Deaths |
| Environment 1 | 10.75 | 4.5 |
| Environment 2 | 6.15 | 8.9 |
| Environment 3 | 2.65 | 10.65 |

Results for the 20 iterations for each environment can be found in the root solution folder in the following files:

* results – env1
* results – env2
* results – env3

They are essentially csv files the order is:

* Average mines gathered for that iteration
* Number of deaths for that iteration
* Average mines gathered until that iteration
* Average number of deaths until that iteration