MACHINE LEARNING: NeuroEvolution Algorithm

Justification

The aim of this assignment was to implement and develop a neuro-evolution algorithm. Initially minesweeper has a neural network associated with it. The minesweepers would be trained by their respective neural network by classification of the sensory inputs that are obtained from the minesweeper. The structure of the neural network was fixed to 2 input nodes, 4 hidden nodes and 2 output nodes. The choice of fewer nodes implied that the dimensionality of the weight space would be low thus allowing us to statistically sound and reliable results.

The sensory inputs were based on the dot products the look vector of the sweeper 0made with the sweeper's closest super mine and its target mine. The target mine is a unique mine that each sweeper is assigned to approach. In the case of few mines than minesweepers, the target mine assignment is not unique. Mine targeting reduces the chances of circling of minesweepers about one fixed point in the environment. The dot products would be clamped to ensure that the sensory inputs are focussed on a particular range of inputs and not just the whole range of -1 to 1.

```
Clamp(dot_supermine, 0.0, 0.5);
Clamp(dot_target_mine, 0.5, 1);
//Classify the output
double dots[2] = { dot_target_mine, dot_supermine };
uint response = genomes[i]->classify((const double*)&dots);
```

After each iteration, the fitness value of the neural network would be computed, the parents selected and the selection operators (crossover and mutation) performed to remove the worst performing minesweepers.

Fitness Function

The fitness function selected for this assignment was simply considering the number of mines gathered and the number of ticks the sweeper survived. The function is illustrated below:

```
double liveliness = (sweeper->getTimeOfDeath() > 0) ?
       (sweeper->getTimeOfDeath() / CParams::iNumTicks) : 1.5;

if (CParams::iNumMines > 0)
       result.fitness = sweeper->MinesGathered() * ((sweeper->MinesGathered() /
              CParams::iNumMines) * MAX_FITNESS + liveliness * MIN_FITNESS);
```

The above fitness function ensures that more emphasis is taken to the mines gathered than the number of ticks the sweeper survived per iteration. As a result, lazy sweepers who manage to survive an iteration without collecting mines are considered highly unfit (fitness=0).

Selection

For the selection process, the genomes were sorted based on their fitness values. The sole purpose for the selection process as implemented, was to remove the weak genomes from the population.

The first implementation of this was based on removing the genomes with the minimum fitness value. These genomes would be replaced by a mutated crossed-over offspring from two of the elite parents. Pseudo code is shown below:

```
vector<ParentAndFitness> sortedParents; // max at the beginning
for i in 0 ... sortedParents.size():
    if sortedParents[sortedParents.size() – i – 1].fitness == minFitness:
        // check if there are any valid parents to recombine with
        if (sortedParents[i].fitness == minFitness or
            sortedParents[i+1].fitness == minFitness)`:
            break;
        crossover (sortedParents[i], sortedParents[i + 1],
            sortedParents[sortedParents.size() – i – 1]);
        mutate (sortedParents[sortedParents.size() – i – 1]);
    else: break;
```

This selection method ensured that the worst performing genomes were removed and converted into "better" genomes at the same time it ensured that crossover was not done by the worst performing parents.

Crossover

The crossover method performed was a 1-point crossover where the point of crossover was selected at random. The method took in two const NeuralNetworks (parent genomes) and the offspring neural network which is the new neural network to be obtained. Furthermore, to avoid recalculation of the network size, the network size parameter was give. Pseudo code shown below:

```
void crossover(const CNeuralNet& a , const CNeuralNet b,
                CNeuralNet offspring, const int networkSize) {
    int selectedPoint = RandInt (0, networkSize);
    foreach neuronLayer nl in a:
        foreach neuron n in nl:
            foreach weight w in n:
                if (selectedPoint > 0):
                    offspring.neuronLayer.neuron[w] =
                        a.neuronLayer.neuron[w];
                else
                    offspring.neuronLayer.neuron[w] =
                        b.neuronLayer.neuron[w];
                selectedPoint--;
}
```

Mutation

For the mutation operator, the weights of an offspring's neural network were either randomized to a new weight value or remained the same. This mutation was based on a random RandBool() function which can either return a true or false. Pseudo code

```
foreach weight in offspringNetwork :
	if (RandBool()):
		weight = -1 + 2 * RandInt (0, 10000)/ 10000;
```

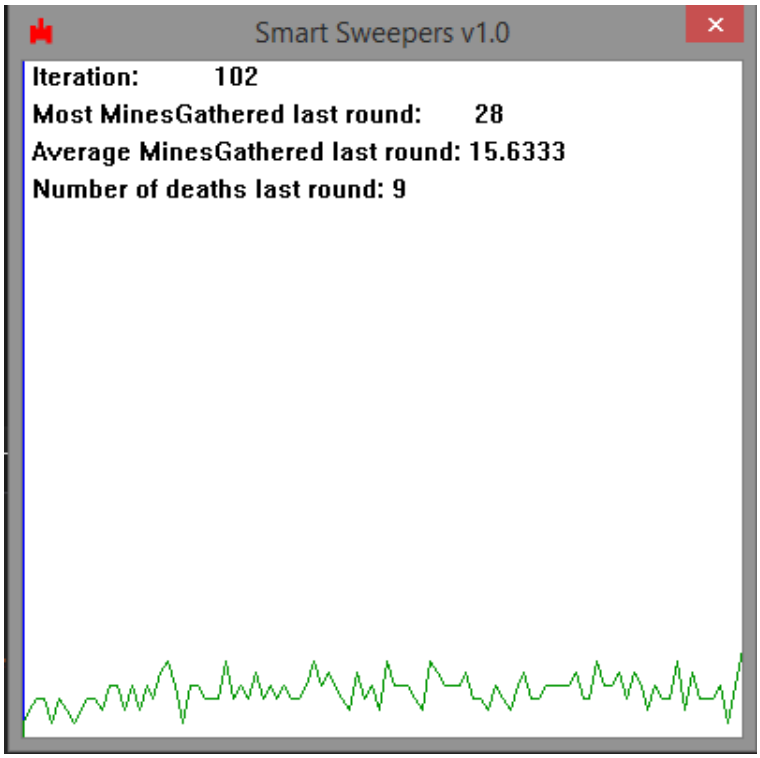
Overall Pseudo code

```
Initialize genomes
for each iteration:
	if end of iteration:
		for each genome:
			computeFitness();
		sort_genome_parents();
		select_parents();
		crossover_and_mutate();
	else
		for each sweeper:
			process input sensors
			classify input by associated ANN (genome)
			respond to classification
```

Results

Environment1

	Average Mines Collected	Sweeper Deaths
1	15.4333	10
2	13.3333	11
3	14.3333	9
4	12.3333	11
5	14.3667	10
6	14.6	8
7	14.1333	10
8	15.4667	9
9	13.9	9
10	14.7667	12
11	14.8	8
12	15.5667	11
13	14	9
14	15.4667	9
15	15.2	9
16	15.3	10
17	16.2333	7
18	14.8333	10
19	13.1	13
20	14.6333	10
Average	14.59	9.75

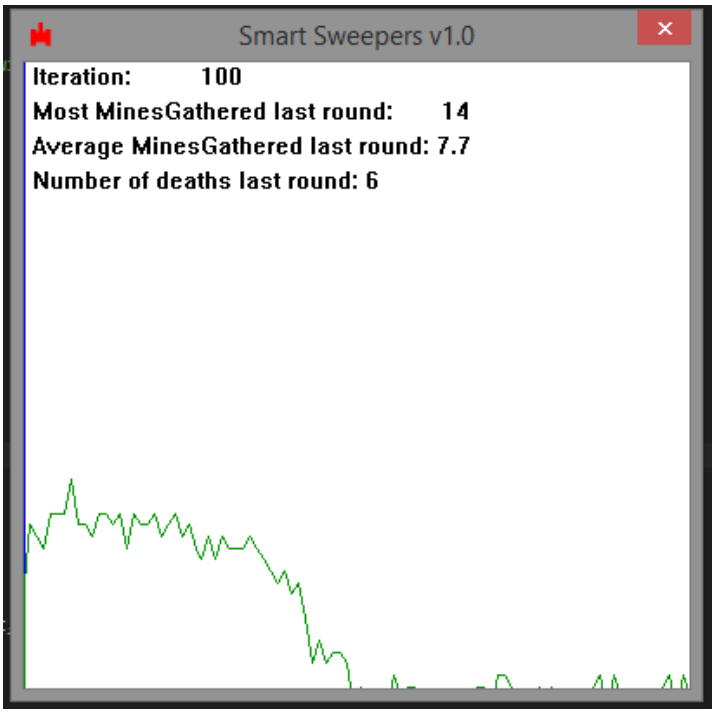


The results for all the environments were run for 100 iterations and their behaviour recorded to env1_log.txt, env2_log.txt and env3_log.txt. The tables shown in this section illustrate the last twenty iterations.

Please note that the green line on the graph is the percentage number of deaths.

Environment2

	Average Mines Collected	Sweeper Deaths
1	7.4	4
2	8.3	5
3	7.66667	6
4	7.03333	7
5	6.86667	4
6	7.1	7
7	7.66667	5
8	7.76667	3
9	6.63333	4
10	8.26667	4
11	7.23333	5
12	7.5	5
13	7.03333	6
14	6.93333	7
15	7.06667	4
16	7.66667	7
17	7.4	5
18	7.36667	5
19	7.7	6
20	7.5	3
Average	7.4050005	5.1



Environment 3

	Average Mines Collected	Sweeper Deaths
1	1.7	2
2	3.03333	6
3	2.66667	1
4	2.76667	3
5	3.06667	2
6	1.8	3
7	2.73333	4
8	2.1	4
9	2.73333	3
10	1.83333	4
11	2.36667	4
12	2.3	5
13	2.16667	4
14	2.26667	3
15	1.86667	1
16	2.5	4
17	1.6	6
18	2.13333	1
19	3.03333	3
20	2.4	1
Average	2.353334	3.2