Evolving Neural Networks for a Flatland Agent - Project Report

Mathias Ose & Øyvind Robertsen
Norwegian University of Science & Technology
mathiabo@stud.ntnu.no, oyvinrob@stud.ntnu.no

Abstract

This report describes a solution to Project 3 in the subject IT3708 at NTNU. The purpose of this project is to use an evolutionary algorithm to tune the weights of an artificial neural network, using the networks performance as an agent in a 2D world with simple rules as a fitness measure.

I EA design

I.1 EA parameters

Table 1 gives an overview of the parameters with which we achieved the best performance. It should be noted that some of the rate parameters were tweaked slightly throughout our experiments, but the table is representative of the type of selection pressure we applied.

These parameters were found by applying our experiences from the previous exercise onto this problem domain. We knew that some degree of elitism would be required. To achieve this, we chose generational mixing and tournament selection as adult and parent selection strategies respectively.

We experimented with various values for tournament selection bracket size, crossover and mutation rates to ensure that we weren't too elitist and to keep mutations at a productive level. Population size and number of generations to simulate were then adjusted to make sure that a reasonably intelligent solution could be found within a few minutes.

Population size	200
Generations	100
Crossover rate	0.5
Mutation rate	0.01
Adult selection	Generational mixing
Adult to child ratio	0.5
Parent selection	Tournament selection
Bracket size	16
Epsilon	0.15
Crossover operator	One point crossover
Mutation operator	Per genome component

Table 1: *Table of EA parameters*

I.2 Fitness function

$$s_{food} = \frac{f_{eaten}}{f_{total}}$$

$$s_{poison} = \frac{p_{eaten}}{p_{total}}$$

$$f = s_{food} - s_{poison}$$

The equations above describe the fitness function we implemented. The fitness f is the share of the total available food eaten, s_{food}, minus the share of the total available poison eaten, s_{poison}. So an optimal agent that eats all the food and avoids all the poison will receive a score of 1.0, while a catastrophically bad agent that eats all the poison while avoiding all food receives a score of -1.0.

II ANN implementation

II.1 ANN design

Figure 1 shows the ANN design with which we achieved the best results, a relatively simple perceptron. There are six input neurons, three representing the food sensors and three representing the poison sensors. Each input neuron is connected to each of the output neurons. The three outputs control the agents movement. We used a single bias node, outputting 1.0 to the output layer.

We used a standard sigmoid for our activation function. To some initial surprise on our part, we achieved the most consistent results using binary weights.

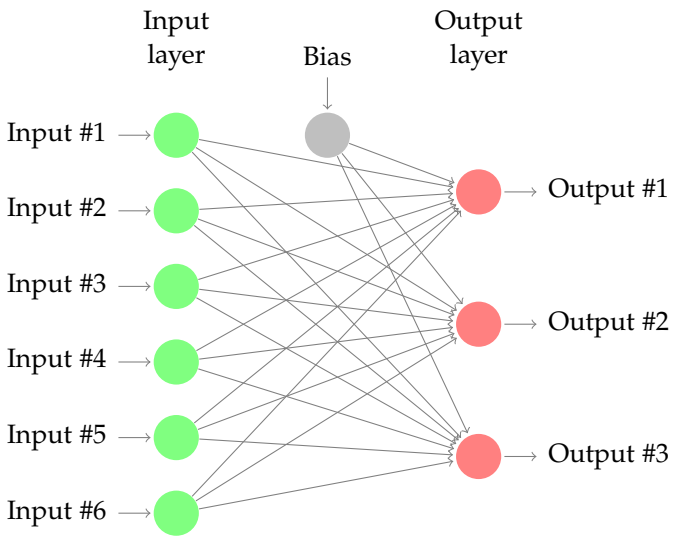

Figure 1: *ANN layout*

II.2 Design process

As we had no prior experience designing ANNs, we initially approached the problem by "blindly" trying different configurations. We started out using 8-bit weights, and a single hidden layer in addition to the design depicted in figure 1. Through several evolutionary runs, with varying parameters used for the runs (increasing selection pressure), the best individuals using this design achieved fitness values between 0.35 and 0.6. Observing these agents moving about the flatland, we saw that they decreased their score by making bad decisions, e.g. moving into a neighboring empty cell instead of a cell with food.

Continuing our experiments, we tried simplifying our network by removing the hidden layer, which did in fact produce a more fit population.

On a whim we also changed the bit count per weight to just 1. To our surprise, the very first run with this weight-encoding was also the first run in which an agent achieved a fitness of 1.0.

However, after giving it some thought, we realised that this simple design restricts the search space for our EA. This means there are fewer local maxima, and the jumps in the search space needed to find a better maxima are shorter. A near-optimal agent for the Flatland problem does not require a lot of complexity. It is a simple mapping of $3 * 3 * 3$ possible input states to 3 possible output states, and could be coded with just a sequence of if-else statements. By simplifying the neural network we make it more likely to find a configuration which acts this way.

III EA PERFORMANCE

III.1 Static, single scenario

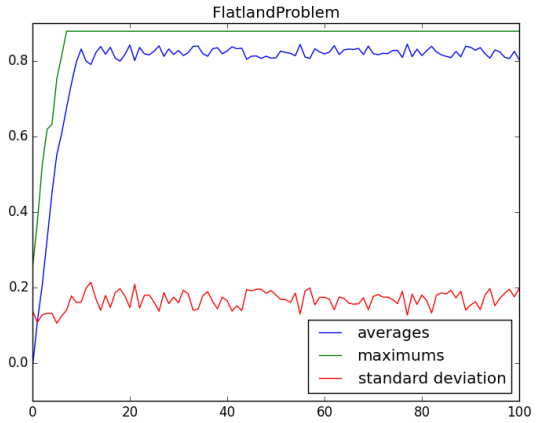

Figure 2: *Fitness plot for static, single scenario run.*

While no optimally performing agent was found, the best agent ate no poison and left only one piece of food in the static scenario. This resulted in a fitness value of ~0.89. Testing that same agent on a freshly generated scenario yielded a fitness value of ~0.44. Since we are "training" the network only on a single scenario, it is to be expected that it will not be a good general solution. This explains the drop in performance on the new scenario.

III.2 Static, five scenarios

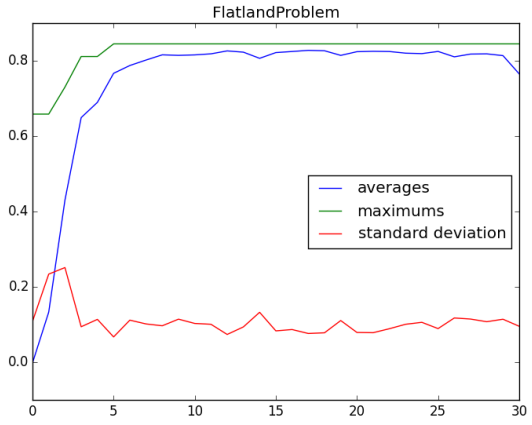

Figure 3: *Fitness plot for static, five scenario per generation run.*

Testing each generation on a static set of five scenarios does not yield a higher top fitness (~0.85). Applying the best individual on a new scenario resulted in

a fitness of ~0.77, which means this a slightly better strategy for breeding a general solution to the flatland-problem. Since the population is trained on the same set of scenarios each generation however, the generality of the resulting network depends on whether the five scenarios represent the breadth of the problem domain. If they are simillar then this training will not be much better than the single static scenario training.

III.3 Dynamic, single scenario

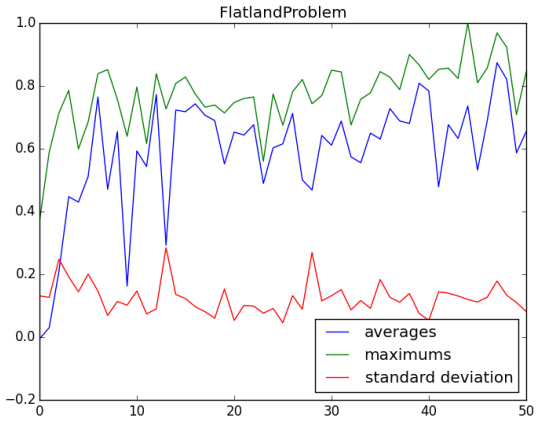

Figure 4: *Fitness plot for dynamic, single scenario run.*

The dynamic, single scenario run has a very fluctuating finess plot. The "environment" changes too much for the selection pressure applied throughout the EA run have much value.

The best performing individual scored a fitness value of ~0.98 during the run, and ~0.57 when retested on a new scenario.

III.4 Dynamic, five scenarios

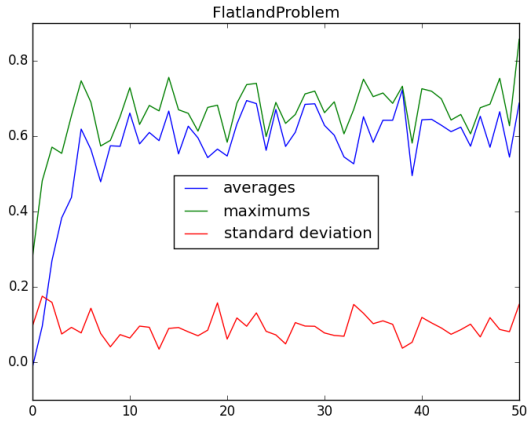

Figure 5: *Fitness plot for dynamic, five scenarios per generation.*

As we can see from the figure, using a dynamic set of five scenarios yields a slightly less chaotic fitness plot. The best individual from this run scores a fitness value of ~0.86. Retesting on a fresh board resulted in ~0.82

This is the best strategy for breeding agents that perform well in any instance of a flatland-problem.

III.5 Notes on behaviour

The following are some observations we noted while experimenting with the configurations outlined above.

- The difference between using one or five scenarios in the static case is large. Due to the randomness involved in creating five scenarios, those five boards will more often than not be different enough that the resulting agent will be fairly general.
- In the dynamic case, the agents produced are often functioning completely optimal, but receive penalties because they run out of time.