IT3708 Project 3

Evolving Neural Networks for a Flatland Agent

Iver Jordal

EA choices

Parameters

Adult selection method	Over production
Parent selection	Fitness proportionate
Adult pool size	25
Population size	50
Number of generations	100
Crossover rate	0.5
Mutation rate	0.5

The process of finding these parameters was basically just experimentation and looking at fitness plots, in the same fashion as in project 2. I tuned one parameter at a time and went forward with the parameter value that was found to be best. One might call it a naïve, manual *grid search*. For the sake of brevity, and because I described this kind of process in detail in the previous assignment, I will leave out the details about the parameter tuning process in this report.

Fitness function

Each agent keeps track of how many items of food (x) and items of poison (y) it has consumed. The fitness of an agent (after its journey of t timesteps) is computed this way:

$$f(x, y) = x - 0.5 * y$$

In other words, there's a reward for eating food and a penalty for consuming poison. I could give more penalty for eating poison, but I didn't do that because a smaller penalty allows the agent to explore more and potentially find and eat more food. This way, we avoid agents that come to a complete stop in the face of poison in all three directions, for example.

ANN structure

Because of the simple nature of the problem, I decided to try an ANN structure without any hidden nodes first. This is also in accordance with Occam's razor principle:

> *Among competing hypotheses, the one with the fewest assumptions should be selected.*

When you think about it, it makes sense that inputs can be directly connected to outputs. If there's food to the left, you want to go left. If there's poison to the right, you want to NOT go right, but rather choose a forward or left if there's no poison there. In other words, there's a direct link between observations and the chosen action, given that the agent has limited sensor data and does not have any memory from previous time steps.

However, there is one edge case that needs to be dealt with: When the agent does not sense any items, it should move *somewhere* in order to keep searching for items (preferably food). That's how I figured that it

would be smart to add a bias node which can be thought of as an input node with constant value 1. This way, the agent can learn that a weight from the bias node to the forward motor node would be nice to have for those cases where no food nor poison is sensed.

The activation function in the output layer is tanh and the threshold value for actually moving at all is 0. That is, if no output values are above 0, the agent won't move. The ANN is fully connected and the weights are in the range $[-2, 2]$.

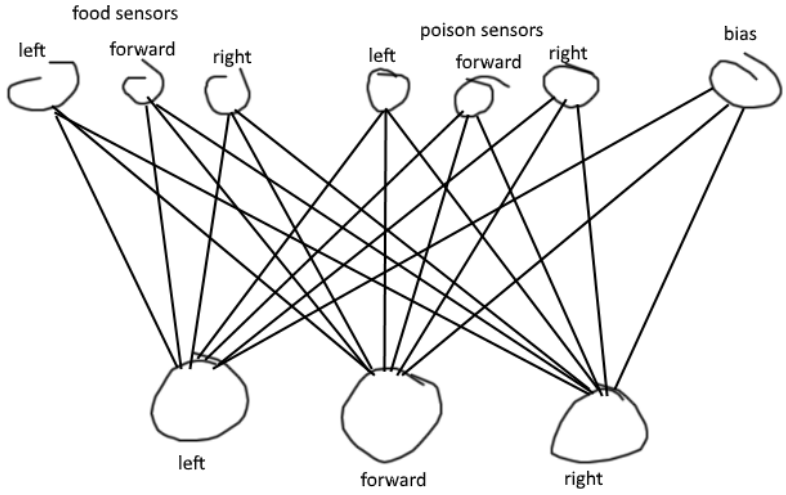

Figure 1: The final structure of the artificial neural network (please disregard my limited drawing abilities)

I did not do experiments with many different neural network structures, because:

- This simple structure immediately proved to be effective
- According to Occam's razor, one should prefer simple structures over more complex structures
- A simpler structure makes the agent easier/faster to evolve because there are fewer weights to tune
- My thought experiments as described earlier did not reveal any cases where hidden nodes would be useful, given the simple nature of the problem. However, if the agent would get memory and/or better sensors, hidden nodes would become useful because it would make the decisions harder to make. In other words, there would be need for more sophisticated pattern recognition.

Performance of the EA

Static run, single scenario

The agent seems to be kind of overfit to run well on the single scenario it was optimized for. For example, it has a positive weight from poison_forward to motor_forward, which is generally bad, but proved to be useful in this specific scenario. The agent generally likes to move forward regardless of what item is in front of it. If there's food to the left or to the right, it will sometimes change direction.

Testing it on a random scenario yields a bad result. In some random scenarios the agent would eat more poison than food, because it had learned that moving forward is good, no matter what is there.

Static run, 5 scenarios

With 5 scenarios instead of 1, the agent generally makes better decisions. Its poison evading abilities have improved. It also runs better on random scenario (which it was not tested on during evolution). It's still not perfect though, because in some cases it decides to consume poison when there's a better option.

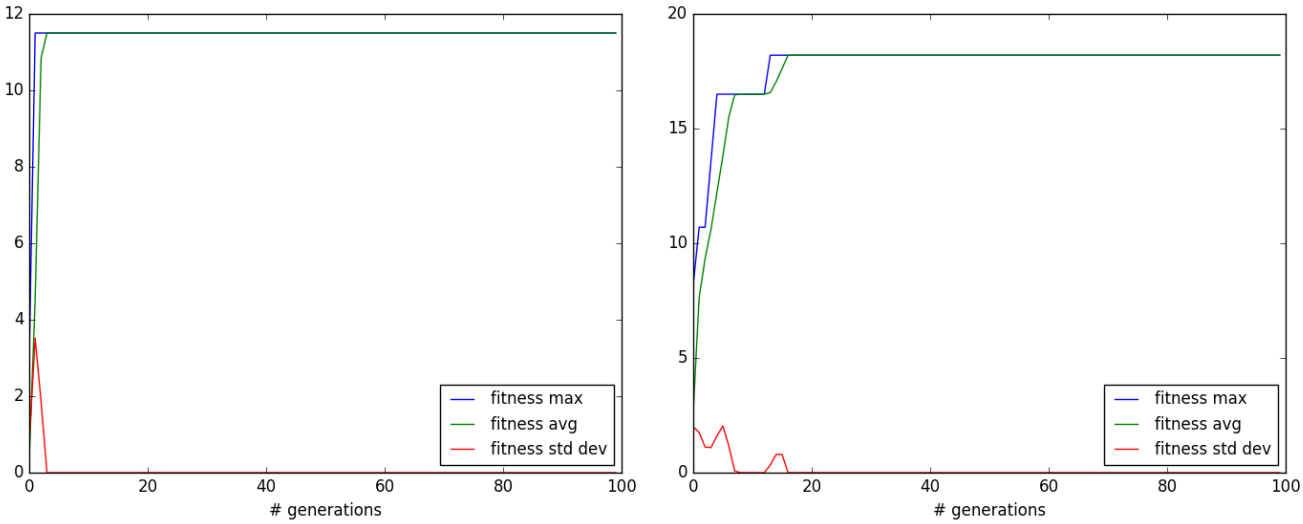

Figure 2: Left: 1 scenario per fitness evaluation. The best fit agent is quickly found. Right: 5 scenarios per fitness evaluation. It takes more generations to evolve the best fit agent because it needs to be able to handle more scenarios.

Dynamic run, 1 scenario

Behavior of best agent after this run: It really likes moving towards food, and it avoids most of the poison. Still, it sometimes eats poison while it could have been avoided. To sum it up: It is pretty good, but there's room for improvement.

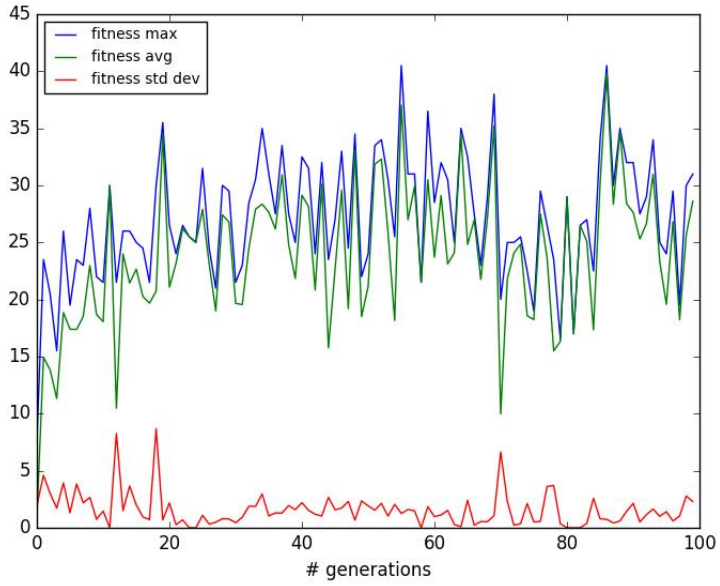

Figure 2: Fitness plot. The max fitness varies much between generations. This is because there's a single new, randomly generated scenario for each generation. Interestingly, there's a significant dip around generation 70. I suspect that there was a "difficult" scenario where suboptimal agents proved to be better, and consequently they ruled for the following generations before generally good agents started taking the top spots again.

Dynamic run, 5 scenarios

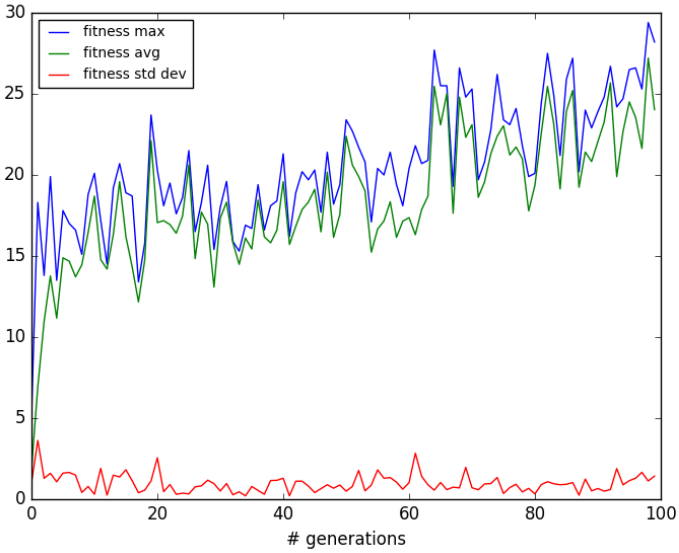

Figure 3: Fitness plot. With 5 scenarios instead of 1 there's less variance. The general trend is that the fitness increases more steadily than what was the case for the single scenario run.

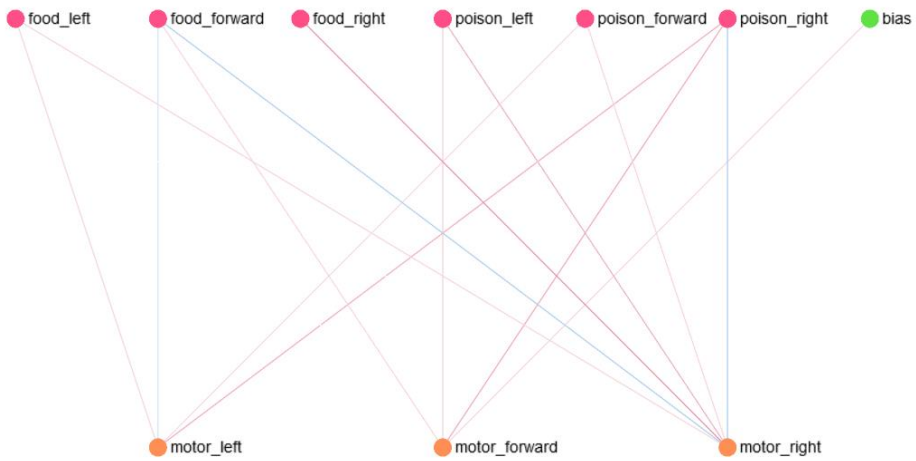

Figure 4: Visualization of the neural network of the best agent. Light color means small absolute value while strong color means large absolute value. Red means positive weight and blue means negative weight. Zero weights are invisible. One can see that most weights correspond with common sense. For example, food_right has a strong positive connection to motor_right and poison_right has a negative connection to motor_right. Also, the only connection from bias is to motor_forward, which makes sense because that allows the agent to explore more. If the default action would have been right or left, the agent would just get stuck going in circles and finding less food.

Generally, the dynamic run with many scenarios for each fitness evaluation proved to yield the best agent. Static runs result in agents that are good on only one or a few scenarios. They are overfit and thus do not generalize well. Dynamic runs are better because only agents that are good in all kinds of scenarios survive. More scenarios per fitness evaluation is good because it prioritizes agents that are good on *multiple* random scenarios.