Sub symbolic AI methods: Project 3

Evolving neural networks for a flatland agent

Bjørnar Walle Alvestad

2016

Norwegian University of Science and Technology

NTNU

EA parameters and choices

The EA-configuration is presented in the table to the right. Multiple configurations of the EA-system were tested, such as different combinations of adult- and parent selection strategies, mutation rates and population sizes. Especially high mutation rates caused the agent performance to vary greatly, not converge to a stable fitness. Because of this, the mutation probability is set to 0.5 per genome. That is, 50% chance to flip 1 bit in a genome.

Parameter	Value
Adult selection:	Generational mixing
Parent selection:	Tournament selection
Tournament size:	20
Population size:	100
Mutation mode:	Per genome
Mutation probability:	0.5
Crossover probability:	0.7
Elitism:	2
Max generations:	1000

The genotype is implemented as a binary vector, where each connection weight occupy 24 bits (3 bytes). The programmer can specify the number of bytes allocated to each weight. Each connection weight is calculated by extracting the corresponding 24 bits from the binary genotype vector, and dividing it with the biggest number representable with the specified number of bytes. The logic uses two's complement binary, which allows for both positive and negative weights. This calculation results in a real valued number between -1.0 and 1.0, which is multiplied by 10 to form the final connection weight.

Fitness function

The fitness of an individual is calculated by taking into account the number of foods and number of poison eaten by the agent. The mathematical expression for the fitness function is:

$$fitness = \frac{\sum foodEaten - \sum poisonEaten}{\sum foodInScenario}$$

The fitness is calculated by taking the sum of foods eaten in all scenarios, minus the sum of poison eaten in all scenarios, divided by the total number of food in all scenarios. This implies a perfect score of 1.0 for an agent that eats all the food in all scenarios, while not eating any poison in any scenario. It also allows for negative fitness if the agent eats more poison than food.

ANN implementation

The artificial neural network used in this system is a 3-layered network, with 6 input nodes, 10 hidden nodes and 3 output nodes. The number of sensory input and motor actuators of the agent reflects the number of input- and output nodes of the neural network. The network accepts a binary vector of length 6 as input, and produces a real-valued vector of length 3 as output. Each component of the output vector represents the activation of the respective motor actuator. The agent action (left, right or forward) is determined by the component with the highest activation. The agent will always move in each timestep, as there is no activation threshold for the actuators. This is deliberately done to avoid the agent getting stuck by not moving at all.

Parameter	Value
Number of hidden layers:	1
ANN configuration:	6, 10, 3
ANN activation functions:	Linear (all layers)

Each neuron in the network implements a bias and bias unit with constant activation of -1, and a connecting weight. This weight is specified by the EA and is subject to evolution, just as all the other weights in the network. The range of accepted connection weights is -10 to 10, as described in the section above.

This design was chosen after testing the agent performance using different combinations of hidden layer count, number of neurons and the activation function. At first, the sigmoid and tanh activation functions provided acceptable agent performance, but was significantly slower than the linear function. Using different activation functions in each layer did not increase agent performance, and so the linear activation function is used in all layers.

Adding multiple hidden layers did not increase the performance of the agent, but only made the execution slower. 10 hidden nodes proved to serve a good performance to speed ratio of the system.

By mutation and crossover operations, the evolutionary algorithm should be able to produce neural connection weights that produces reasonable intelligent behavior in the agent. This problem is a form of classification task, in which artificial neural networks is known to be powerful.

Agent performance analysis

Static run, 1 scenario

The system is run with a single static scenario, for 1000 generations. The result plot only includes the first 100 generations, as it quickly converged to an acceptable fitness of 0.806, with a final fitness of 0.839 after 1000 generations. The agent performance is (31, 5) (31 foods eaten, 5 poisons eaten). This is not a perfect result, but shows that the system can differentiate between food and poison and shows an acceptable level of intelligence.

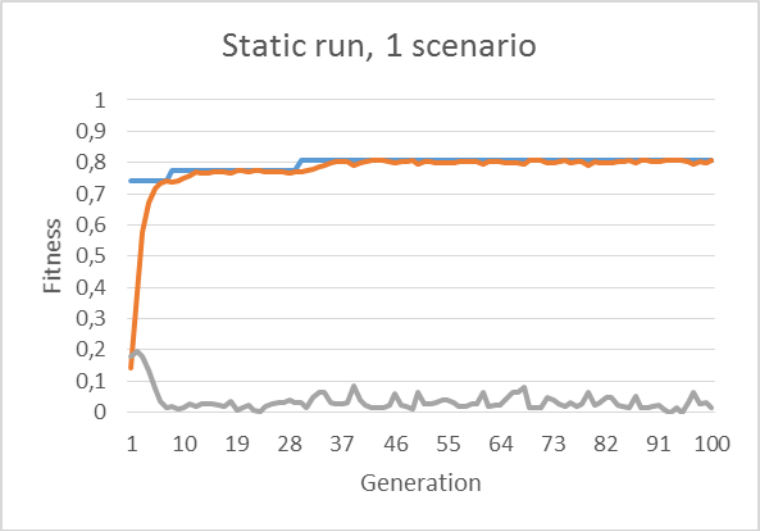

Figure 1: Fitness plot of the static, 1 scenario run. The plot shows best fitness (blue), average fitness(orange) and standard deviation(gray).

Testing the same agent on a new, randomly generated scenario, yielded mediocre results. The agent got stuck in a loop pattern, going in circles. Its performance was (14, 4).

Static run, 5 scenarios

In this run, the fitness evaluation is based on five different scenarios. It produces a marginally lower fitness result than the first run, but should still present an acceptable level of intelligence. In this plot, all 1000 generations is shown. The best fitness after 1000 generations is 0.755. The agent performance is (14, 4), but that is still a good result, as there were only 15 foods on the board, along with 18 poison. The complete agent performance is presented in the table to the right. The agent got stuck in a circle pattern early in the fifth random cenario, which explains the somewhat poor result. It also achieved a perfect score of (24, 0), where all food entities were consumed, but no poison.

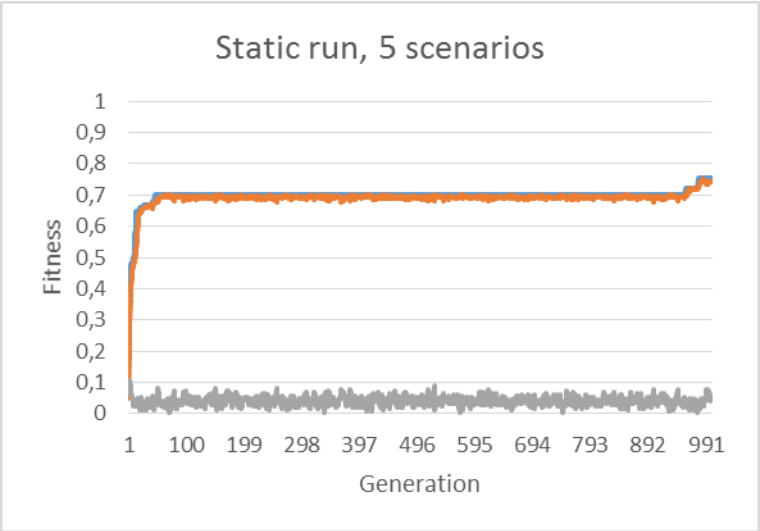

Figure 2: Fitness plot for the static, 5 scenario run.

Static scenarios	Random scenarios
(14, 4)	(20, 2)
(18, 5)	(15, 3)
(24, 1)	(24, 0)
(14, 3)	(15, 4)
(16, 2)	(5, 2)

Dynamic run, 1 scenario

In a dynamic run, a newly generated scenario is used in every generation during fitness testing. In this run, an agent achieved a fitness of 1.0 after 482 generations, where the simulation is configured to stop.

The agent performance is (22, 2) on a random scenario, which is a strong result. The agent consumed most of the food while avoiding the majority of poison.

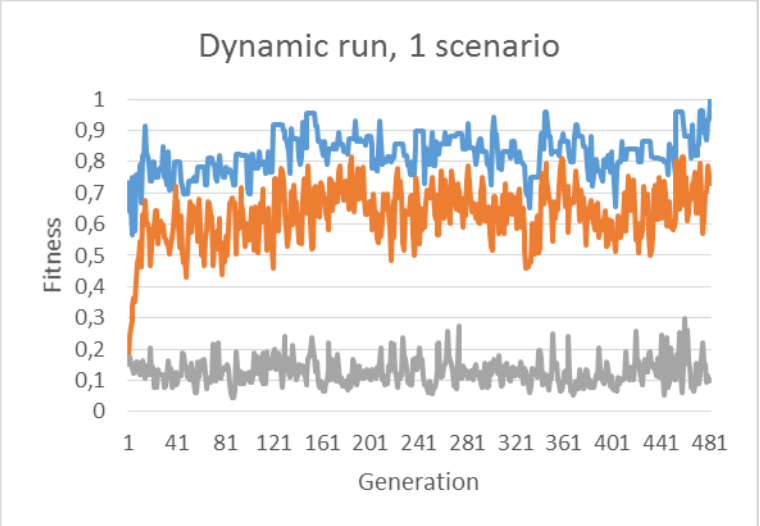

Figure 3: Fitness plot for the dynamic, 1 scenario run.

Dynamic run, 5 scenarios

This run is again based on 5 different scenarios during fitness evaluation. The best fitness value after 1000 generations is 0.752. This agent produces the following results when presented with 5 randomly generated scenarios:

Random scenarios
(7, 1)
(20, 5)
(4, 0)
(15, 9)
(21, 5)

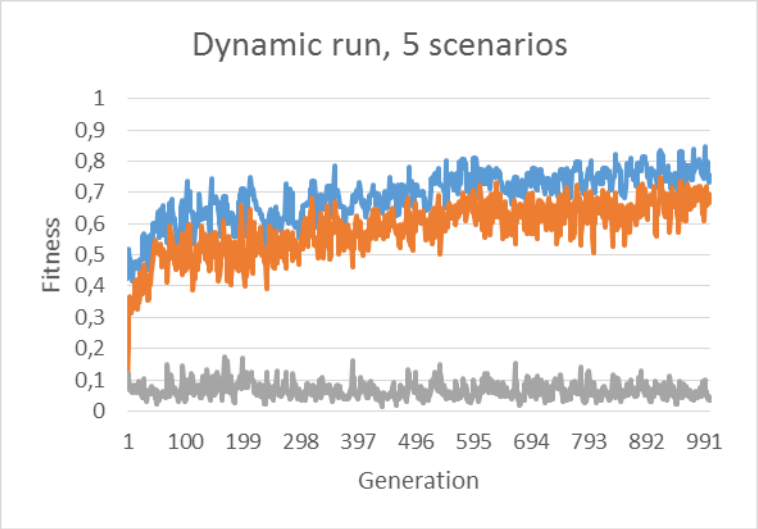

Figure 4: Fitness plot for the dynamic, 5 scenario run.

In scenario 3, the agent got stuck in an infinite pattern where it walked in a circle, and was only able to eat 4 food in the 60 timestep limit. In scenario 4 the agent performed poorly; (15, 9), while achieving an acceptable result in the other scenarios.

There are some distinct differences in the four runs documented above. The plots shows clearly that in a static run, the fitness of the population grows faster, and stabilizes at a given level. In the dynamic runs, both the best- and average fitness levels varies greatly throughout the run, while slowly increasing. One reason for this is that in a dynamic run, the individuals are tested on random scenarios each generation. While an agent can perform exceptional in one scenario, it may achieve a poor fitness value in another because of randomly positioned poison that is hard to avoid. Some scenarios may also have food that is hard to approach without first eating a poison. In a static run, the same scenario is used for fitness evaluation in all generations, and will not induce this kind of fluctuations in the fitness value.

In the static runs, the fitness value converged faster when running 5 scenarios, compared to only one. This is because 5 scenarios introduce more variation in the test environment, thus requiring fewer generations to incorporate desired agent behavior.

However, there is no obvious run that is superior to the others. All four runs produced agents with approximately the same performance level. Even though the agent reached a fitness of 1 in the single, dynamic run, the agent probably got “lucky” on the single scenario.