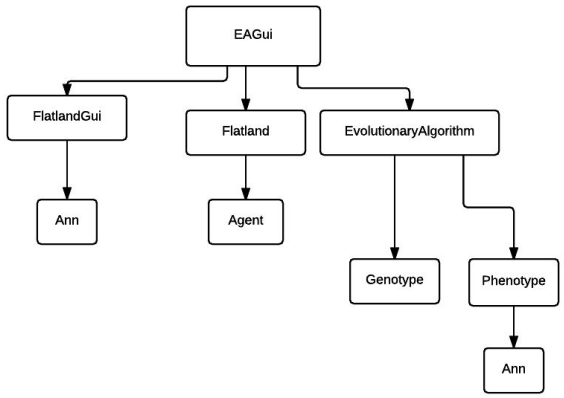
# Project 3

Subsymbolic Methods in AI
Håkon Gimse

**Architecture and implementation**
My system is started using the EAGui class. This is a somewhat modified version of the gui used for specifying parameters for the EA in project 2. It is modified to allow specification of parameters for the neural network as well. This class spawns an EvolutionaryAlgorithm instance, which is the same as used before. This class is in charge of running the basic EA steps and keeping a population of solutions. These solutions are Genotype and Phenotype objects, and behave exactly as they did in project 2. The only difference is that the specific implementation of the Phenotype base class is called PhenotypeAnn. The fitness of this phenotype solution is calculated as specified below. The EAGui class is also in charge of keeping and producing new environments for the agent. This environment is instances of the class Flatland, and contains the board. Each Flatland instance has an agent instance of the class FlatlandAgent. When the training of the neural network is complete, and we want to visualize the efficiency of the agent, the EAGui class spawns a new gui class. This class is called FlatlandGui. It has a neural network, configured to use the best phenotype found as weights, and uses this to visualize an agent navigate in a given environment.

Class diagram




This table shows how I experimented with different parameters to arrive at the best configuration of the EA. All runs are with 5 dynamic environments. The neural network here is as simple as possible. It has no hidden layers and uses a sigmoid activation function on the output layer. The comparison is done by doing 10 runs, each with 100 generations, and collecting the best solutions from each. The second and third column contains the average of these solutions fitness value. The most significant indicator for the fitness of a solution is the second column, as a dynamic environment eliminates most of the randomness in the fitness given by the environments configuration. Some environments are harder to achieve a high fitness on, and with a static environment, an unlucky environment can affect the score and average a lot. A dynamic environment evens this out.

| Nr | Avg best fitness after 100 dynamic generations | Avg best fitness after 100 static generations | Genotype pool size | Adult pool size | Elitism | Mutation rate | Crossover rate | Symbol set size | Tournament size | Tournament slip- through | Adult selection protocol | Parent selection protocol |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.8844 | 0.9145 | 100 | 100 | 1 | 0.55 | 0.85 | 10 | | | Full | Fitness proportionate |
| 2 | 0.8990 | 0.9214 | 100 | 100 | 1 | **0.35** | 0.85 | 10 | | | Full | Fitness proportionate |
| 3 | 0.8986 | 0.9164 | 100 | 100 | 1 | **0.15** | 0.85 | 10 | | | Full | Fitness proportionate |
| 4 | 0.8955 | 0.9177 | 100 | 100 | 1 | 0.35 | **0.5** | 10 | | | Full | Fitness proportionate |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 0.8872 | 0.9162 | 100 | 100 | 1 | 0.35 | **0.95** | 10 | | | Full | Fitness proportionate |
| 6 | 0.8903 | 0.9210 | 100 | 100 | 1 | 0.35 | 0.85 | 10 | | | Full | **Sigma-Scaling** |
| 7 | 0.8992 | 0.9169 | 100 | 100 | 1 | 0.35 | 0.85 | 10 | 10 | 0.65 | Full | **Tournament** |
| 8 | 0.8965 | 0.9218 | 100 | 100 | 1 | 0.35 | 0.85 | 10 | 10 | **0.25** | Full | **Tournament** |
| 9 | 0.9021 | 0.9229 | 100 | 100 | 1 | 0.35 | 0.85 | 10 | 10 | **0.10** | Full | **Tournament** |
| | 0.8350 | 0.9110 | 100 | 100 | 1 | 0.35 | 0.85 | 10 | 10 | **0.90** | Full | **Tournament** |
| 10 | 0.8986 | 0.9116 | **90** | **110** | 1 | 0.35 | 0.85 | 10 | 10 | 0.10 | **Over production** | Tournament |
| 11 | 0.8962 | 0.9058 | **70** | **130** | 1 | 0.35 | 0.85 | 10 | 10 | 0.10 | **Over production** | Tournament |
| 12 | 0.9042 | 0.9155 | 100 | 100 | 1 | 0.35 | 0.85 | 10 | 10 | 0.10 | **Mixing** | Tournament |
| 14 | 0.9036 | 0.9159 | 100 | 100 | **5** | 0.35 | 0.85 | 10 | 10 | 0.10 | Mixing | Tournament |
| 15 | 0.9014 | 0.9134 | 100 | 100 | **0** | 0.35 | 0.85 | 10 | 10 | 0.10 | Mixing | Tournament |

The green rows show where an experiment improves the system. When this happens the parameter i experimented with follows into the next round of experimentation. The bold cell are the parameter value being tweaked in this run.

**Fitness function**

My fitness is determined by how well the agent does it in 60 time steps. It receives a bonus for every food eaten, and a penalty for every poison eaten. All the 60 moves are determined by the ANN, using the weights represented in the phenotypes components. The optimal score is 1, and is received when the agent eats all the food, without eating any poison. The following equation describes the fitness:

$$Fitness = \left( \sum_{i=0}^{nr\ of\ scenarios} (x_i + (Y_i - y_i))/(X_i + Y_i) \right)/nr\ of\ scenarios$$

Where X and Y is the total amount of food and poison in the environment *i*. The amount eaten of food and poison is denoted by *x* and *y* in the same given environment *i*.

I found that this was very effective in getting the agent to explore and eat most of the food, and only sometimes having to eat poison. I tried doubling the negative effect, as it is now weighted equal, but then the agent often became too passive. I think this is a fitting fitness function for this problem, as it performs reasonably well given enough time to search for a good network configuration.

**Description of the complete ANN design**

When deciding on the configuration of the neural network, I used the same incremental experiment approach, and document how i arrived at a specific design. I started out with the best found configuration of the EA found above. This network had simply a input layer with 6 nodes and an output layer with 3 nodes. The input node values were determined as shown in below. The resulting output nodes is also shown. Here I selected the output node with the highest activation, and only did that action.

$Inputs = [Food\_front, Food\_left, Food\_right, Poison\_Front, Poison\_left, Poison\_right]$

The values of the input vector is either 0 or 1, depending on the sensory inputs from the agent.

$Outputs = [Move\_front, Move\_left, Move\_right]$

I incrementally increased the complexity of the network in the following table.

| Nr. | Avg best fitness after 100 dynamic generations | Hidden layers | Nr of nodes in each layer | Activation functions | Symbol set size |
|---|---|---|---|---|---|
| 1 | 0.9053 | 0 | | Hyperbolic tangent | 10 |

| | | | | | |
|---|---|---|---|---|---|
| 2 | 0.9077 | 0 | | Rectify | 10 |
| 3 | 0.9099 | 0 | | Softmax | 10 |
| 4 | 0.8792 | 1 | **4** | **Sigmoid, softmax** | 10 |
| 5 | 0.9025 | 1 | 4 | **Hyperbolic tangent, softmax** | 10 |
| 6 | 0.9023 | 1 | 4 | **rectify, softmax** | 10 |
| 7 | 0.9051 | 1 | 4 | **softmax, softmax** | 10 |
| 8 | 0.8765 | 1 | **2** | rectify, softmax | 10 |
| 9 | 0.8977 | 1 | **6** | rectify, softmax | 10 |
| 10 | 0.9090 | 0 | | Softmax | **20** |
| 11 | 0.8991 | 0 | | Softmax | **5** |

My best found configuration of the neural network was actually using no hidden layers. I had expected some increase in performance when I increased the complexity, but this was disproven. I also experimented with the activation function, and found that the best working one was the softmax function on the output layer. In experiment 4-7 I experimented on hidden layer activation functions with a hidden layer with 4 nodes. This experiment did not yield any new configuration surpassing the previously found in fitness score. The best combination found was rectify and softmax. I used these two and tried to increase and decrease the number of nodes in the hidden layer in experiment 8-9. This also failed to yield better results. The last experiment i did was change the number of different valid values of the weights (symbol set size in the phenotype). Using a symbol set size of 10 the weights can have the integer values [-5, 4]. For this experiment i used the best found configuration. Either increasing or decreasing this value made the network any stronger. The **best** found ANN configuration was nr 3, using the EA parameters nr 12. This design has proven to work very well, and the agent appears as intelligent as expected. The reason why no complex network is needed may be that the problem is fairly simple.

The activation functions works as shown below.
**Sigmoid:** S-shaped function above 0. Given by the formula:

$$S(x) = \frac{1}{1 + e^{-x}}$$

**Hyperbolic tangent:**

$$S(x) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

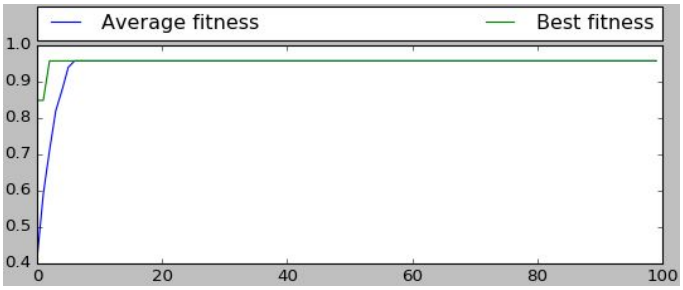
**Rectify:** A simple ramp function, pulling the input in a neuron up to above 0. Given by the function:

$$S(x) = Max(0,\ x)$$

**Softmax:** Softmax function, given by the function:

$$S(x) = \frac{exp(x - max(x))}{sum(exp(x - max(x)))}$$

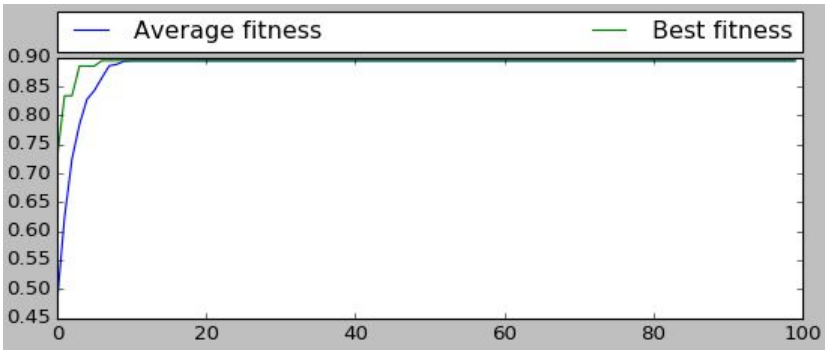
**Performance of the EA**
Static run on a single scenario




The agent performs very well on this scenario. IT appears highly intelligent, almost as it has more knowledge about the environment than it actually has. This is a result of being tested and evaluated in the same specific environment configuration. When I ran the agent with the same solution on a new

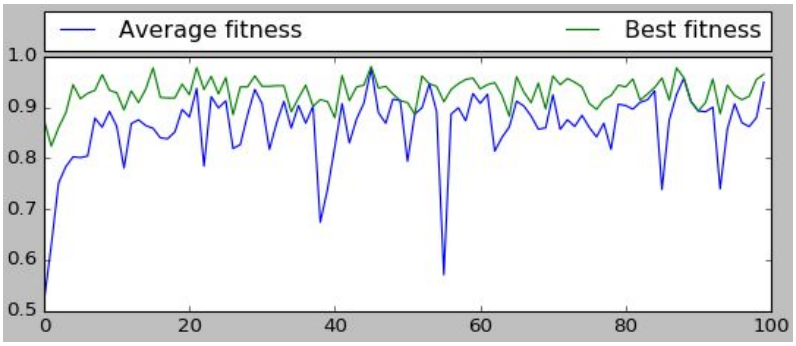
environment it performed a lot worse. It ate 3 pieces of poison and did not collect as much food as it did in the first run. It was still intelligent enough to collect most of the food and avoid most of the poison.

Static run on 5 different scenarios



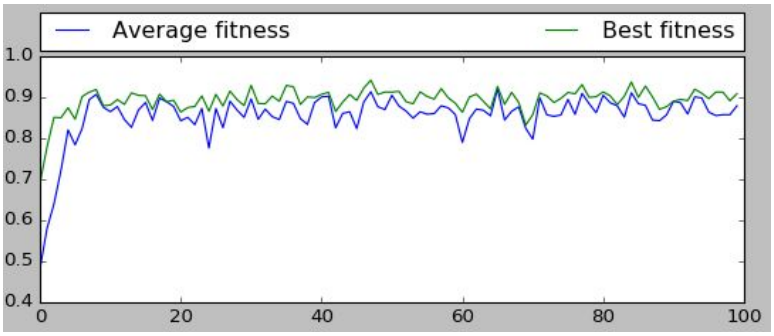

All the agents appear highly intelligent, and performs reasonably well. Still, the average is somewhat lower than what it was using only one environment. It is harder to specialize an agent on five different scenarios than just one. When i reset the environments here, the agents performance drops, but not as much as it did when using only one scenario. No poison is eaten by any agent, but the amount of food eaten is significantly less. This agent seems more robust to dynamic environments then the first.

Dynamic run on a single scenario




The agent performed very well. It did not eat any poison, and only left one piece of food. The system often solved the problem (eating all the food and leaving all the poison), but this was dependant on the randomly configured environment. As seen in the graph, the solutions fitness is very dependant on the configuration of the current environment.

Dynamic run on 5 different scenarios




The advantage of testing each solution on five different scenarios is that the outlying scenarios does not affect the solution selection as much, resulting in a more even fitness curve. All the agents have clear intelligence and performs as good as they can in their given environments. The solution seams to be very adaptive to new environments.