IT3708 Project 4

Evolving Neural Networks for a Minimally-Cognitive Agent

Iver Jordal

# Implementation

## Genotype/phenotype

The genotype is a bit string. Each “chunk” of 8 bits represents an integer. The number of bits in the genotype is determined by the number of weights needed. For example, if 28 weights are needed, then the genotype will consist of 224 bits.

Conversion to phenotype: For each chunk of 8 bits, calculate the sum of 1s and scale it to the desired range. For example biases will be in the range . One alternative technique I tried was regarding the bit chunk as binary number in the range [0, 255]. I didn’t use that because bit flips could shake up the weights too much and the crossover operation didn’t work so well.

## CTRNN implementation

The CTRNN is implemented as a class in Python. It can be initialized with a given number of input nodes, hidden nodes and output nodes. Then its weights can be set (based on the phenotype described above). The CTRNN has an *activate* method that takes in sensor data, runs it through the network and returns the output values of the output layer.

In more detail: First the input buffer is populated with sensor data, then the internal states of the hidden layer is computed. Based on this the output buffer of the hidden layer is computed. Finally the values for the output nodes are computed. The math that is used to compute the internal states and output values is based on the explanation in part 3 of the assignment text.

I’ve found that recurrence in the output layer generally made the agent performance worse. I’ve been told that it was okay to customize the topology of my CTRNN and that it was okay to have recurrence only in the hidden layer. So, adhering to Occam’s razor, I’ve decided to have only have recurrence in the hidden layer. One other customization is that I have five hidden nodes instead of two, because that improves the agent.

The Agent class, which owns the CTRNN instance, is responsible for interpreting the output from the CTRNN. The maximum motor output value is chosen to be the current movement direction, and the magnitude of that value is quantified to a number of steps in the range . In the pull scenario case there is a separate output node for the pull action. The pull action is preferred to movement if the pull node output is over the pull threshold which is

# Performance of the EA

## Standard scenario

By default the agent is moving 2 steps to the left. Whenever it occurs an item of size 1, 2 or 3, it starts jiggling, but stays below the item until it is fully obtained. Whenever the item is of size 4 the agent becomes more sceptical because it looks more like those large objects that should be avoided. Sometimes it stops to catch the item, but on other occasions it misjudges the item and moves past it. For large items (size 5 or 6) the agent jiggly scans both ends of the objects for a few time steps until it decides to quickly flee to the left. After it has safely escaped the item, it moves a bit to the right again. This is good because it delays the agent so it won’t rediscover the same item after wrapping around. In some cases, when the agent discovers a large item late, it is only able to partially avoid it.

Fitness function:

## Pull scenario

TODO

Fitness function:

## No-wrap scenario

TODO