Project 4 - Beer Tracker

Aleksander Skraastad

April 14, 2016

1 Implementation details

The implementation uses the following settings, which are almost identical to the ones used from the previous project, except that the pool sizes and mutation rates are significantly higher. This was a result of some experimentation, and the realization that I would require the system to perform a lot of exploration compared to exploitation, in order to cover as much of the search space as possible. This was due to there being several extra parameters than I had used in previous EA + NeuralNetwork implementations. The system uses a regular integer genotype, which is ruled by a granularity setting of 0-2048. When being converted into a phenotype, these values are scaled to fit the ranges presented in the exercise document. The phenotype is then represented as a real value matrix, with one parameter vector for each neuron.

Mutation Rate: 0.75, Component mutation rate: 0.35, Crossover Rate 0.9, Tournament Selection K: 10, Tournament Selection E: 0.3, Child pool size: 200, Adult pool size: 150.

The CTRNN is implemented with one hidden layer. Using sigmoid activation functions, and additional weights for the incoming recurrent signals on each neuron. In essence, each time a signal is fired, a delta scaled by a time constant is added to each neuron's internal state. Based on the gain levels and time constants, this changes the state of the neuron over time, and effectively gives it memory. One can view the neuron as having a residual signal from previous activations. An action is determined by simply assigning each of the output values to a movement direction or pull action. The relative difference of the left and right signals are then used to determine how many steps in either of the direction the agent should take.

2 EA Performance analysis

2.1 Standard Scenario (Wrap)

Most of the time, depending on how well the agent is able to train in the given generations, it will oscillate below all "green" targets. It quickly counteracts it's residual spike train signal when a capturable object is above, and inverts direction repeatedly until capture. For avoidable objects, it will, depending on the strength of the weights, scurry away. Normally to the side from which it came. It does however happen that it will instead carry on forward with more momentum. This however is more often correlated with the occasional skipping of capturable targets of large size. The fitness function for the standard wrap scenario I used was (C=Captured, A=Avoided, M=Missed, S=Struck by):

$$\frac{C+A}{1+M+S*1.5}$$

This will quickly reduce an agents fitness if it misses too many capturables, or fails to avoide too many large objects.

2.2 Pull Scenario

This one was a bit more tricky. It is in my opinion quite sensitive, and the provided weight ranges allow it to sometimes not use the pull action, and sometimes pull too much. But most often than not, it will utilize the pull action in a somewhat intelligent fashion. On a couple of interesting runs, the agent managed to learn that all greens were pull on sight, while after completely passing a large object, a pull could safely be initiated. I very rarely experience that though. For the most part it pulls small greens and ignores the "reds".

The fitness function used was:

$$\frac{C^2+A}{M*2+S*37}$$

2.3 No-Wrap Scenario

This scenario is what really shows the effectiveness of memory in the network. Otherwise, the network will tell the agent to oscillate on the edge, or be completely locked until something falling might trigger a movement change. The agent will have a varying degree of oscillation from one direction to the other when it hits the wall. Depending on the rate of state change, and gain, this might be a slow transition from side to side, or in some cases it only travels across half the board before either returning, or hunting for "green" objects. It often hovers under the objects, sizing them up and evaluating if they can be captured on its travels from side to side.

The fitness function used was:

$$\frac{C*5+A}{M*2+S*3.5}$$

3 Evolved CTRNN analysis

```
--- Hidden Layer{3} ---
Neuron{Gain: 3.2207, Bias: -5.6542, Time: 1.239746, Weights: [4.87792, 0.141601, -3.671875, -4.956054, -1.92382, 2.65625, 4.477539, 1.96289]}
Neuron{Gain: 1.84960, Bias: -0.600585, Time: 1.86572, Weights: [3.65722, 3.83789, -3.93554, 3.0859, -2.651367, 1.860351, -0.85, -2.10]}
Neuron{Gain: 3.14648, Bias: -3.7304, Time: 1.7397460, Weights: [0.053710, 1.206054, 4.301757, 3.110351, -3.725585, -0.703125, -0.288085, 2.993164]}
---
--- Output Layer{2} ---
Neuron{Gain: 3.48242, Bias: -4.633789, Time: 1.5053710, Weights: [4.301757, 3.110351, -3.725585, -0.703125, -0.288085]}
Neuron{Gain: 1.25976, Bias: -2.006835, Time: 1.0161132, Weights: [3.7890625, -3.920898, -0.278320, -2.72460, -2.407226]}
---

Input: [0.0, 1.0, 1.0, 1.0, 0.0]

Fire one:   [0.0011284632113274637, 0.0070280572816751735]
Fire two:   [2.257496192360311, 0.9918998740849742]
Fire three: [0.7582144374685603, 1.195517785896403]
Fire four:  [2.5032393958438592, 1.2542734083004992]
```

Here we see how the evolved weights are, with individual bias and rate of change (time constant). We see two main gain values distributed among the neurons as well. These two, in combination, will dictate how fast the agent will react to stimuli. A rapid rate of change will make the agent turn on a dime when encountering objects it should flee from, or potentially turn away from (the wall). The net relative difference in output strength also determines how many of the 4 possible steps the agent will take. Attempting to send the same signal (a 3 block capturable object on center) successively reveals how the output signal changes with time (memory). The desire to move left oscillates quickly, while the desire to move right increases much more gradually.