Evolving Continuous Time Recurrent Neural Networks

Mathias Ose & Øyvind Robertsen
Norwegian University of Science & Technology
mathiabo@stud.ntnu.no, oyvinrob@stud.ntnu.no

Abstract

This report describes a solution to Project 4 in the subject IT3708 at NTNU. The purpose of this project is to use an evolutionary algorithm to tune the weights of a continuous time recurrent neural network, which then acts as an agent in a simple game in a 2D world and receives a performance score.

I Implementation

A phenotype is the representation of the weights of a CTRNN. A CTRNN requires more weights than the simple ANN from project 3. In addition to matrices that represent arcs crossing between layers, there are also internal arcs between the neurons of a layer, and each neuron requires two additional paramenters, the *gain* and *time constant*.

The genotype is a bitstring of size $8 * \text{number of weights}$. The conversion to phenotype takes each 8 bit sequence and interprets it as a floating point value between 0.0 and 1.0. These values are then fitted into matrices that represent the arc weights, or assigned as `g` or `t` values of neurons, and scaled and adjusted to the appropriate range for that weight type.

The CTRNN implementation is a modified version of the ANN implementation from project 3. Neurons are no longer only simple numerical values where input is equal to output, but are objects that maintain state and computes output based on input and state.

At each tick of the simulation the input nodes get either 1 or 0 from the sensors. They then output some value computed with the input, the previous state, the *gain* and *time constant* parameters and the sigmoid function. These values are propagated along the weighted arcs to the next layer, where the sum of the inputs to each neuron goes through the same calculation. In the CTRNN there are also internal arcs in each non-input layer, including arcs from the neuron to itself. These are also part of the sum of inputs.

Once the sensory inputs have propagated their way through the network and the final layer has new output values, the agent object uses these values to make a decision about how to act in the Beer Tracker World. If the left neuron has a higher value than the right neuron the agent will move left, and vice versa. The magnitude of the output decides the magnitude of the movement, so an output near 1.0 means the agent takes 4 steps and an output near 0.0 means it takes no steps at all.

II Performance

Listing 1: *Awarding points which are used for fitness measure*

```
def capture(self, obj):
    if obj.width >= 5:
        self.points -= 2
    else:
        self.points += 1

def avoidance(self, obj):
    if obj.width >= 5:
        self.points += 2
    else:
        self.points -= 1

def fail(self, obj):
    self.points -= 2
```

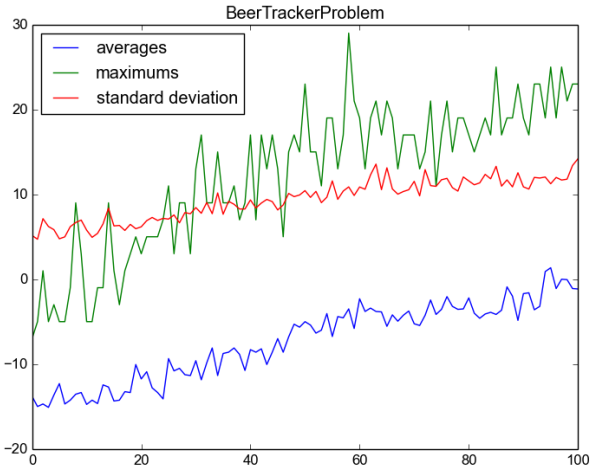

Figure 1: *Standard scenario, 100 generations of evolution*

Listing 1 shows how points are awarded. It is possible to capture an object, to avoid it, or to fail at either. These points are used as the fitness measure, which can be seen evolving in figure 1.

It is more difficult to consistently reproduce the same behavior in this project than it has been in previous projects. This is because both the problem and the CTRNN itself are more complex, with more variable parameters and more randomness.

In the standard scenario the most commonly seen evolved behavior is an agent which moves in only one direction. It stops and stands still to capture most objects, but is usually not very good at avoiding the dangerous objects, and often tries to capture big objects as well.

We have however also seen smarter agents appear sometimes, that are able to avoid most dangerous objects, while also being able to capture most safe objects. These will typically slow down when they find an object, and do small steps to "figure out" the size of the object, then either stand still if it is safe or accelerate away if not. With 1-3 wide objects they stop and wait, with 4-wide objects they move about more because of the lack of long term memory.

The agent in the "pull" scenario typically acts a lot like the standard agent, but uses the pull move instead of standing still and waiting for small objects to drop. Sometimes the time recurrency of the network makes it use the pull ability two times in rapid succession.

In the no-wrap scenario the most commonly seen behavior is for the agent to move to one of the corners and then just sitting there for the entirety of the simulation. We managed to evolve an agent into moving back and forth by increasing the range of the time constant drastically, seen in figure 2. However this agent did not do very well at capturing and avoiding.

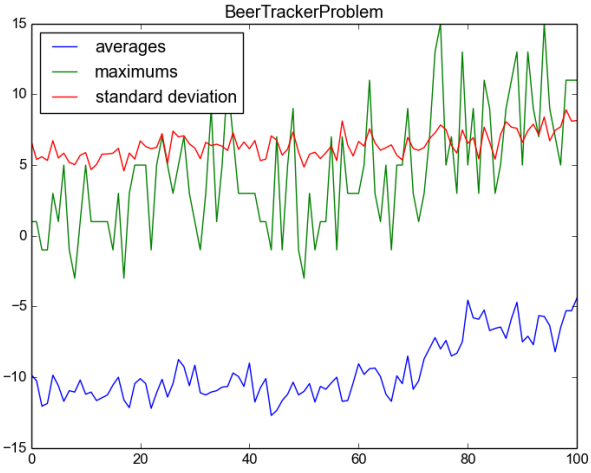

Figure 2: *Nowrap scenario, 100 generations of evolution*

III Analysis

Listing 2: *Evolved phenotype*

```
{'inter': [[[-4.60784314, -4.33333333],
            [ 4.41176471,  4.29411765]],
           [[ 2.25490196,  1.54901961],
            [-2.1372549 , -2.96078431]]],
 'cross': [[[ 4.01960784, -0.7254902 ],
            [-3.74509804, -4.33333333],
            [-0.52941176, -1.78431373],
            [ 1.43137255, -1.43137255],
            [ 2.33333333, -4.68627451],
            [-1.03921569, -3.98039216]],
           [[-0.21568627,  3.94117647],
            [-2.49019608, -1.31372549],
            [-4.96078431, -2.76470588]]],
'ts': [1.2980392156862746, 1.192156862745098,
       1.192156862745098, 1.192156862745098,
       1.1607843137254903, 1.1294117647058823,
       1.9607843137254903, 1.396078431372549,
       1.1686274509803922],
'gains': [3.9019607843137254, 1.0784313725490196,
          3.83921568627451, 3.1333333333333333,
          2.6, 3.164705882352941,
          1.188235294117647, 2.4117647058823533,
          2.7254901960784315]}
```

The evolved phenotype shown in listing 2 is very good at capturing and avoiding correctly when it finds objects, but moves very slowly, so still misses a lot.

Looking at the hidden layer, we can see that one of the two has a much larger gain term (3.8 vs. 1.19). The arc emanating from this neuron are -0.22 to the left motor neuron and 3.94 to the right motor. This means the agent is much more likely to move to the right.

However, the hidden neuron with the lower gain also has a much higher time constant, (1.13 vs 1.96). This neuron outputs negative to both the left and right outputs. So if this neuron gets to output high values (because the memory repeatedly gets reinforced) it is able to counteract the other neuron's initiative to move to the right, and hold the agent in place.

Repeatedly inputing all zeroes makes the agent move 1 step to the right, stand still one tick, then move once again, stand still, and so on.

Inputing $[0, 0, 1, 0, 0]$ outputs low values, so the agent stands still. When repeating the input the left neuron keeps a constant zero value while the right neuron alternates between very low values and values that are near the "step 1 to the right" range.