Exercise 4
IT3708

Simon Borøy-Johnsen
MTDT

April 14, 2016

1 Implementation

1.1 Representation

The phenotypes are represented as the weights in the neural network. In addition to the edges between the neuron, as used in the last exercise, support for edges between neuron in the same layer, and looping edges, also had to be implemented into the phenotype. The time constant and gain had to be implemented. The phenotypes are represented as dictionaries with keys for the *gain*, *time constant*, *neurons*, *looping connections*, and *intra-connections*.

The genotype incorporates the artificial network into a bit-string. The bit-string is built up of several eight-bit sequences and is interpreted as a value in the range appropriate for the node. The value can either be a weight in the network, or the *gain* or *time* value of a neuron.

1.2 CTRNN

1.2.1 Definition

The CTRNN network is based on the simple neural network from the last exercise. The neurons have been modified, to incorporate the *gains* and *time constants*. Instead of only using the input for calculating the output (as was did in the last exercise), the state of the neuron is also used. Both the input, the gain and time constant, and the previous state contribute to the final output value of a neuron.

The neuron now also have connections to other neurons in the same layer, as well as to themselves.

1.2.2 Use

At each step in the run, the input takes on the values from the sensors (either 1 for shadow, or 0 for no shadow). The input nodes then compute their output based on the input, the gain and time constant, and the previous state. These values are fed to the sigmoid function, before travelling though the neuron's connections, both normal, looping, and intra-layer connections.

The values propagate through the network, before ending at the output nodes. The agent uses these values for making a decision on what to do next. If the difference in output values is great enough, the agent makes a move. If not, the agent stands still. The difference in output values decides the size of the step taken.

2 EA Performance

2.1 Fitness function

The fitness value is decided by how many small objects the agent captures, and how many large objects it avoids. For each small object the agent catches, it gets one point. For each small object the agent misses, it looses one point. For each large object the agent avoids, it gets two points. For each large object the agent catches, it looses one point.

2.2 Standard scenario

The most common behaviour seen in the standard scenario is an agent that moves only in one direction, before it stops and waits for any falling items it sees over itself. It usually makes sure to not cover itself completely, in order to avoid loosing points. When the agent cannot see anything above itself, it move on in the same direction, looking for blocks.

Sometimes, the agent is too risk averse, also avoiding covering more than three of its own blocks, missing

out on the four-block items. It also sometimes race past one-block items.

2.3 Pull scenario

The agent for the pull-scenario behaved much like the standard agent, with the exception that it pulled whenever it was covered with one to three-block items. Also here the agent sometimes ignored one-block items.

2.4 No-wrap scenario

The agent for the no-wrap scenario moved to one of the corners, and stood there for the rest of the game.

3 Analysis

3.1 Weights

The agent with phenotype as in Figure 1 was able to capture $\frac{28}{30}$ small items, and avoiding all the large ones.

```
{'looping_connections': [array([[1.58823529, 3.94117647],
       [ 1.98039216, -2.29411765]]), array([[3.03921569, 5.],
       [-0.41176471, -1.58823529]])], 'intra_connections':
       [array([[ 1.50980392, -3.82352941], [ 4.88235294, -3.8627451 ],
       [ 3.74509804, -0.09803922], [ 4.09803922, 2.84313725],
       [-4.52941176, -4.21568627], [ 2.1372549 , 3.47058824]]),
       array([[-1.82352941, -0.33333333], [-1.58823529, 3.47058824],
       [ 0.29411765, -2.05882353]])], 'time': [1.1490196078431372,
       1.3372549019607844, 1.6705882352941175, 1.9490196078431372,
       1.011764705882353, 1.4509803921568627, 1.5568627450980392,
       1.1215686274509804, 1.6431372549019607, 1.2941176470588236,
       1.5294117647058822, 1.8470588235294119, 1.3411764705882354,
       1.4666666666666668, 1.3176470588235294, 1.8470588235294119,
       1.7137254901960786, 1.0784313725490196, 1.0470588235294118,
       1.784313725490196, 1.9098039215686273, 1.4901960784313726,
       1.8745098039215686, 1.1137254901960785, 1.988235294117647,
       1.1176470588235294, 1.6509803921568627, 1.3411764705882354,
       1.4588235294117646, 2.0, 1.803921568627451, 1.2705882352941176,
       1.6980392156862745, 1.8941176470588235, 1.6588235294117646], 'gains':
       [1.9411764705882353, 3.070588235294118, 3.4784313725490197,
       1.4078431372549018, 1.4078431372549018, 2.427450980392157,
       3.070588235294118, 4.670588235294117, 3.6980392156862747]}
```

Figure 1: Derp

3.2 Different inputs

Input	Output	Explanation
$[1, 1, 0, 0, 0]$	$0.996, 0.960$	The agent wants to move slightly to the left to figure out whether the block is small or large.
$[0, 1, 1, 1, 1]$	$0.999, 0.962$	This input was given right after the input $[1, 1, 0, 0, 0]$ was given. Here, the agent have probably found out that the block above is large, and it wants to continue moving to the left.