IT3708 Project 4

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 parameter range.

CTRNN implementation

The CTRNN is implemented as a class that can be initialized with a specific topology and a set of parameters (weights, time constants etc). It has an *activate* method that takes in sensor data, runs it through the network and returns the output values of the output layer. The math and procedures used are based on the explanation in part 3 of the assignment text. Its topology is different from the suggestion though. My variant has more hidden nodes and no recurrence in the output layer. For more details, see the code. When it comes to choosing action, the movement direction is based on the maximum motor output value and the magnitude of that value is quantified to a number of steps.

Performance of the EA

Standard scenario

Default action: 2 steps to the left. Whenever it senses 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 sceptical. 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.

Fitness function: $1 * small\ captures + 1.5 * large\ misses - 3 * partial\ captures - 3 * small\ misses - 3 * large\ captures$

Pull scenario

Default action: 3 steps to the right. When it starts sensing shadow on its rightmost cells, it spends a few time steps moving back and forth, trying to learn whether the item is large or not. Then, if the item is found to be large, the agent moves past the item. Otherwise it decides to pull the object.

Fitness function: $1 * small\ captures + 1.5 * large\ misses - 3 * partial\ captures - 3 * small\ misses - 3 * large\ captures + 1 * good\ pulls - 1 * bad\ pulls$

Where a $good\ pull$ is defined as $(small\ capture \,||\, large\ miss)$ and a $bad\ pull$ is defined as $(partial\ capture \,||\, small\ miss \,||\, large\ capture)$

No-wrap scenario

The agent has learned to move in the opposite direction when it hits a wall. When it bounces from the right wall, it correctly keeps moving left until it either hits the left wall or senses an item. Sometimes it fails to move all the way to the right when moving away from the left wall. When it finds an item, it reacts accordingly. It is able to catch most small objects (size 1-3). It has some difficulties with distinguishing between objects of size 4 and larger objects, though.

Fitness function: $2 * small\ captures + 1.5 * large\ misses - 3 * partial\ captures - 3 * small\ misses - 3 * large\ captures + 1 * placement\ reward$

Where $placement\ reward$ is defined to be $15 / \left(1 + 5 * \left|0.5 - \frac{x_l}{x_l + x_r}\right|\right)$ given that x_l is the number of timesteps the agent was in the left half of the world and x_r the same thing but for the right half of the world.

Analysis of an evolved CTRNN

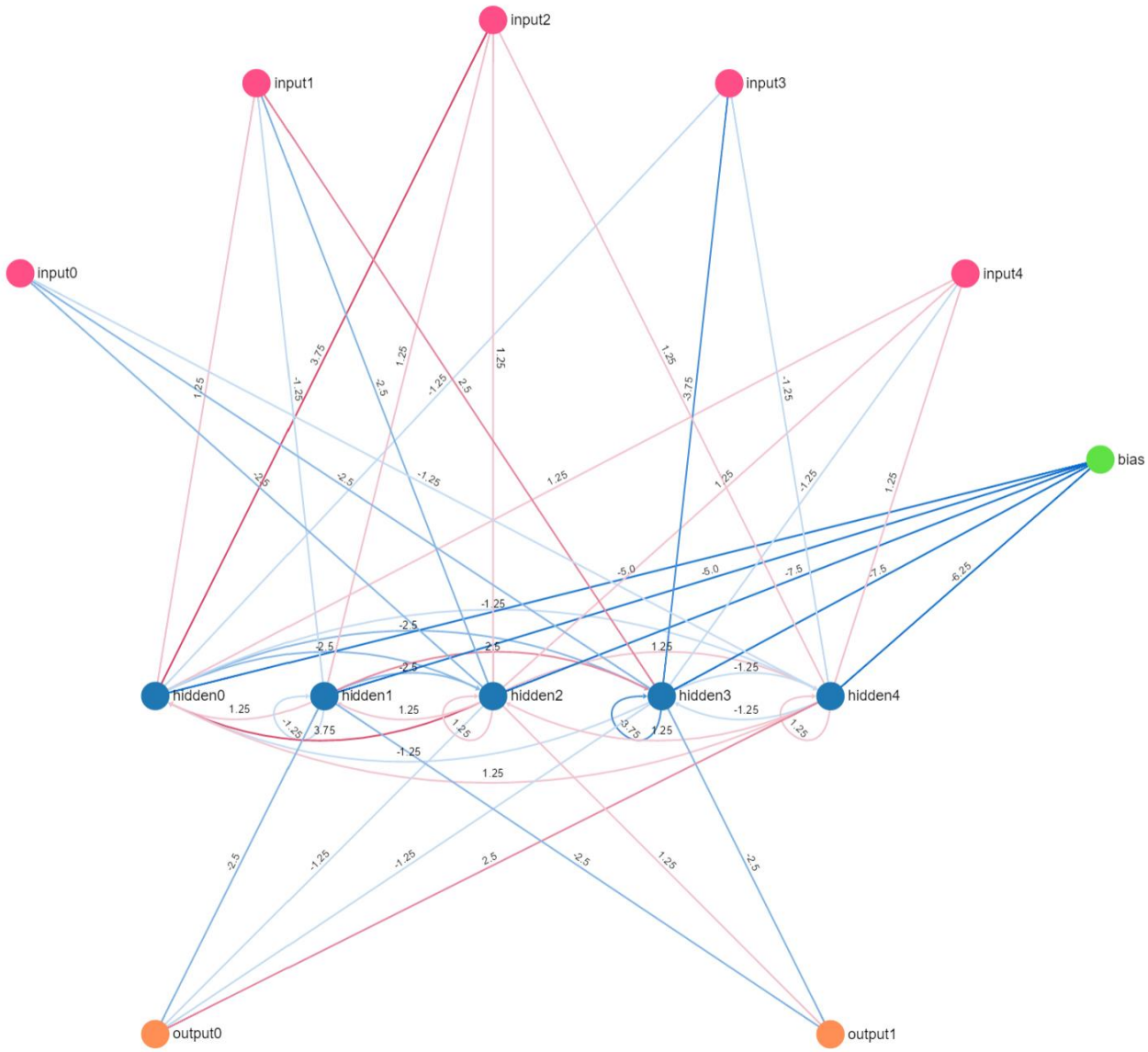

Figure 1: Visualization of an evolved CTRNN for the *standard* scenario

Hidden0 doesn't send any signal directly to the output nodes. It acts as a memory node that strongly feeds a representation of its state to hidden2 (amongst others), which is the only hidden node that sends positive signals to the right motor output node. Hidden0 probably helps remembering whether there is a large object or not. **Hidden1, hidden2 and hidden3** act as inhibitors. Whenever they send out signals, leftwards movement is inhibited. In case of high activation levels in hidden3, the agent might move right rather than left. **Hidden4** has a strong weight to output0, which moves the agent to the left. This is probably the node that by default moves the agent to the left when it is looking for items.

Here are some consequent inputs and resulting output values from the CTRNN evolved in the *pull* scenario.

```
[0, 0, 0, 0, 0] -> [0.498805, 0.500812, 0.497859]  # move right by default
[0, 0, 0, 1, 0] -> [0.499881, 0.499664, 0.500003]  # pull that tiny object
[0, 0, 0, 0, 0] -> [0.499984, 0.500001, 0.499979]  # move right
[1, 1, 1, 1, 0] -> [0.499801, 0.499116, 0.499957]  # not sure if large, go left
[1, 1, 1, 1, 0] -> [0.499289, 0.492673, 0.501247]  # pull
```

The two last activations have the same input values, but the last output has changed from the second last. This shows how the memory of the network can give different output for the same input based on earlier input.