Practical session: Hebbian learner

First, before you begin get a feeling for how NNs work: http://playground.tensorflow.org/ and https://github.com/cazala/synaptic/wiki.

In this assignment, you will modify your existing code (either Lemmings or Didabots) to create an agent that learns from the environment. You will be working on this assignment until the last practical session of the year, and report your findings in a short paper (deadline December 22).

The settings and tasks are similar to the DAC example described in Chapter 5 from Understanding Intelligence (Pfeifer & Scheier), which you should read. For details refer to the lecture on artificial neural networks given by Beata.

In short, we have a robot with distance sensors that should 1) wander around and 2) learn how to avoid objects. Unlike the DAC example, we are not using real Khepera robots nor an array of coupled proximity-collision sensor pairs, so you will need to exercise creativity to translate the problem to the simulation world. Probably the simplest approach is to use two binary "touch" sensors (e.g. modify the distance sensor) and two distance sensors. Your challenge is to implement Hebbian learning in the simulator such that the robot learns weights that enable it to avoid objects during 20,000 simulation steps. Objects to avoid include walls, blocks, and other robots in the arena. We then want you to test whether this learned avoidance behavior produces similar clustering behavior to the behavior seen in the Didabot assignment. For effective clustering you may need to modify some simulation settings (indicate these in the report).

Assignment

First, to test whether clustering occurs, create a 4x4 grid of boxes (similar to the Didabot assignment) and run your Hebbian learner in this environment with three different learning rates, for example {0, 0.001, 0.01}. Be sure to include zero as a baseline. Second, vary in three levels, another property of the simulation (you can test all learning rates or just the one which worked best) which you believe will affect the development of the weights (for example, the number of robots, number of boxes, a forgetting rate). Make sure to always include reasoning behind your decisions.

Data collection

For both experiments store the development of the weights, and the number and size of the clusters formed. Run at least 10 simulation per condition. Consider to write automatic counting of quantities like percentage of boxes in clusters during the simulation, to reduce manual work. As before, statistical tests are not needed but you can include them if you wish.

Report

- Write a scientific paper reporting the results of your experiments.
- Include a concise description and reasoning behind the design of your simulation, your chosen factor and your algorithms.

- Report and plot the results of both experiments. This should include a) how the weights change with learning rate and b) your chosen factor. In addition also describe and plot how the properties of the clusters, a) the number of clusters and b) the percentage of boxes in heaps, change with the learning rate and your factor.
- Discuss the influence of learning rate and your factor on weight development and heap formation.

Technical notes

- To describe your algorithms concisely and completely (e.g. including values of named constants) it is beneficial to include precise pseudocode or well documented JavaScript code.
- It is important to provide parameter settings and initial conditions in the report, to improve reproducibility. Consider examples screenshots and tables provided in an appendix.
- For your initial weights it is useful to use a weight of zero for your distance sensors and a weight of one for your touch sensors.
- In order to avoid your robot getting stuck at a wall in the beginning, you need to implement a method of turning away from the walls depending on the touch sensor value. Behavior depending on the distance sensor should be purely learned.
- Note that previous versions of the code included a bug in the `senseDistance()` function, that you should fix. Specifically, the inner function `getEndpoint()` should look like this:

```
function getEndpoint(rayLength) {
  return {x: startPoint.x + rayLength * Math.cos(rayAngle),
          y: startPoint.y + rayLength * Math.sin(rayAngle)};
};
```

Deliverables

With your report please include the following in a single zip file named `Hebbian_report_NAME.zip`, where `NAME` stands for the person handing it in.

- Screenshots of one exemplary simulation result for experiment 1 and experiment 2 (you do not need to hand in all screenshots of all simulation runs).
  Consider adding them as figures into your report.
- All code files. Clearly label in your code what needs to be changed in order to switch between experiment 1 and 2 and the different conditions.