IT3708: Project 1

Magnus Moan

January 31, 2017

Task 1
I've chosen to implement my project in Java. My implementation is devided into three parts; model, controller and view. The model consists of 2 classes (Board, Network). The Board class defines a 10-by-10 playing board. The Network class defines a general neural network with inputs, outputs and weights. The controller part of my implementation contains mechanisms for altering the model. The most important classes in the controller part are Game, NeuralAgent, BaselineAgent and MainApp. The Game class controls the game by keeping count of number of rounds, number of games, score, etc. The most important procedures in this class are *play* and *play_one_game_without_gui*. The agent classes defines the behavior of the different agents we are asked to implement. The NeuralAgent class defines the behavior for all the agents associated with a neural network. Finally the MainApp class starts the entire application. The view part of the implementation is only considered with the graphical user interface of the application. A sample of the GUI can be seen on the next page.

My baseline agent have the following line of priority: food, empty, poison, wall. If my agent can choose between two or more options with the same item it will move forward before left and left before right. My baseline agent recieves an average score of 20.579 on 1000 trials.

Task 2
See *delta_supervised* for code snippet where δ_i is calculated for supervised learning. Comments in the code explains how the code implements equation 3. The argument *output_index* is the value i from the equation.

Task 3
See *delta_reinforcement* and *get_best_output_from_direction* for code snippet where δ_i is calculated for reinforcement learning. Comments in the code explains how the code implements equation 5. The argument *output_index* is the value i from the equation.

Task 4
See plot on next page.

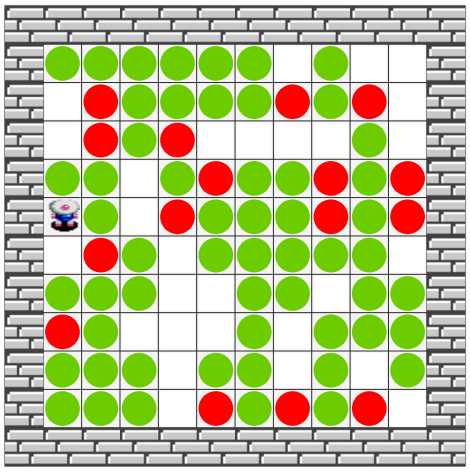
Task 5
Weights for the neural network after 100 rounds of training with the reinforcement neural network from task 3:

	Input: Left				Input: Forward				Input: Right			
	Empty	Wall	Food	Poison	Empty	Wall	Food	Poison	Empty	Wall	Food	Poison
Left	2.2397	-2.7783	3.5745	-1.4320	0.9383	0.4935	-0.7075	0.8791	0.3978	0.5686	0.0988	0.5373
Forward	0.63078	0.1547	0.6952	0.7466	2.3432	-2.7790	3.8455	-1.1820	0.6278	0.1805	0.7055	07136
Right	0.3765	0.6160	0.4656	0.4675	0.7977	0.3024	-1.808	1.0066	2.3271	-2.7834	3.8259	-1.4445

We observe that the weights for each output is at its highest if there is food in the input corresponding to the same direction as the output. There are lesser values for the other options in the inputs in the same direction as the output. We notice that the weights corresponding to inputs from the other directions are of lesser significance. Below we see a sample of inputs and corresponding outputs (the direction corresponding to the highest output is chosen).

Input			Output			Chosen
Left	Forward	Right	Left	Forward	Right	
Empty	Wall	Food	2.8321	-1.4428	4.505	Right
Food	Food	Food	2.9658	5.2462	4.1108	Forward
Poison	Poison	Empty	-0.1550	0.1924	3.8011	Right
Wall	Wall	Poison	-1.7476	-1.9107	-0.5261	Right
Empty	Empty	Empty	3.5758	3.6017	3.5013	Forward

The agents in task 1, 2 and 3 performed close to the same average score. This is as expected as they recieve the same information from the sensors. The agent in task 4 recieved more information and did therefore perform better compared to the others. See the graph showing average scores in the bottom of this page.



```
private double delta_supervised(Board board, int output_index) {
    // Get the current outputs
    double[] curr_outputs = this.neural_network.get_outputs();
    double[] exp_outputs = new double[3];

    // Get the index (0,1 or 2) corresponding to the current move direction
    // (left, forward, right) chosen by baseline agent
    int correct_choice = Helpers.direction_to_index(this.trainer.choose_direction(board));
    double exp_output_sum = 0.0;

    // Calculate exp(output) for each output and also the sum of these values
    for(int i = 0; i < neural_network.get_no_outputs(); i++) {
        exp_outputs[i] = Math.exp(curr_outputs[i]);
        exp_output_sum += exp_outputs[i];
    }

    // If the output we are looking at is the same as the output chosen by
    // the baseline agent we set the variable chosen to 1
    int chosen = 0;
    if(correct_choice == output_index) { chosen = 1; }

    // Calculate equation 3
    return - (exp_outputs[output_index] / exp_output_sum) + chosen;
}
```

```
private double delta_reinforcement(Board board, Integer output_index) {

    // Chosen is the index of the output chosen
    MoveDirection a = Helpers.index_to_direction(neural_network.get_best_output_index());
    int chosen = Helpers.direction_to_index(a);

    if(output_index == chosen) {

        // Find Q(s,a) by getting the output value of the chosen direction
        double Q = this.neural_network.get_outputs()[chosen];

        // r is found by getting the value of the item in the field moved to
        double r = Helpers.get_item_value(board.get_item_from_direction(a));

        // Q_new = Q(s',a')
        double Q_new = get_best_output_from_direction(board, a);

        // Equation 5
        return r + 0.9*Q_new - Q;
    } else {
        return 0.0;
    }
}
```

```
private double get_best_output_from_direction(Board board, MoveDirection direction) {

    // Saving current state
    SightDirection curr_sight_dir = board.get_sight_direction();
    BoardItem curr_item = board.get_item_from_direction(direction);
    int[] new_x_y = board.get_new_x_y(direction);
    if(curr_item.equals(BoardItem.WALL)) {
        return 0.0;
    }

    // Moving agent
    board.move_agent(direction);
    set_inputs(get_sensor_info(board));
    update_outputs();
    double Q_new = this.neural_network.get_best_output_value();

    // Restoring state
    board.move_agent(MoveDirection.BACKWARD);
    board.set_sight_direction(curr_sight_dir);
    board.add_item(new_x_y[0], new_x_y[1], curr_item);
    set_inputs(get_sensor_info(board));
    update_outputs();
    return Q_new;
}
```