IT3708 Spring 2017 – Project 1

Supervised and Reinforcement Learning of Neural Agent Controllers

20% of final grade. Individual project—no groups allowed.

Purpose: Implementing simple neural networks and using them to control agents. Training the neural networks using approaches based on supervised learning and reinforcement learning.

Introduction

In this project you will get acquainted with simple versions of neural networks and using them to control agents. The agents will be situated in the "Flatland" environment, which is a simple 2D environment where the agent needs to respond to sensory inputs in order to avoid dangers and to seek out food. The neural networks used to control the actions of the agent will be trained using supervised learning and reinforcement learning. The neural networks will receive the agent's sensory readings as input values, and the network outputs will then be used to decide the direction for the agent to move in.

For this project you will be required to implement all of your program's features from scratch, which includes simulating the Flatland environment, visualizing the agent's actions in the Flatland world, calculating outputs from neural networks and training these networks using supervised learning and reinforcement learning. You may use any programming language you wish, as well as general packages for drawing your visualizations and plots. You may *not* use libraries to handle tasks such as training and evaluating your neural networks—an important point of the project is for you to write this code yourself.

The next section describes the Flatland environment and the sensors and actions available to the agent. The remaining sections of the document describes the various tasks that this project consists of. First you will implement a simulator for the Flatland environment, as well as a "baseline" agent against which to compare the rest of the agents. This baseline agent will not use neural networks, but will be written in regular code—the agent will later be used as a "teacher" for the supervised learning neural network. Your simulator must be able to visualize the Flatland world and the agent's actions in it.

The next task is to implement an agent controller where the agent's actions are decided by a neural network. The weights in the neural network will be trained using a gradient descent learning rule, where the training signal to the network will be determined by using the baseline agent as a teacher. This is thus a supervised learning paradigm. The third task is to adapt your supervised learning agent to instead use a reinforcement learning paradigm. The agent will no longer have the teacher available, but will have to learn based on rewards and punishments received from the environment. As we will see later, the gradient descent part of the learning rule stays intact, so that the only change necessary is in the calculation of the network's error signal.

Up until now, the neural agents will likely have produced results similar in performance to the baseline agent. A major use-case for neural networks is, however, in solving problems where we don't already have a solution prepared in a regular programming language. The fourth task explores whether the reinforcement learning agent can find ways to exploit additional sensory data, when we increase the range of the agent's sensors so that it can look farther ahead. Finally, in the fifth task we ask you to analyze one of your trained neural networks and to reflect upon the results you have gotten.

Each task specifies the number of points obtainable from that task. If you find that you will not be able to finish all of the tasks in time, please keep in mind that you can still submit your project for partial credit.

The Flatland environment

The Flatland environment is a 2D world consisting of 10x10 squares arranged in a 2D grid. Each square is connected to its four neighboring squares up, down, left and right. A square is either empty, occupied by food or occupied by poison. Outside of the 10x10 world, all possible cells are occupied by a solid wall.

The specific configuration of a Flatland world is generated randomly. For each of the 10x10 squares, there is a 50% chance that it could spawn with food in it. If a particular square doesn't receive any food, there is a 50% chance that it could spawn with poison in it. Otherwise, the square remains empty. The agent starts in a random square, facing in a random direction.

Whenever the agent enters a new square, it receives a score as follows:

Empty:	0 points	**Food:**	4 points
Wall:	-100 points and the simulation ends	**Poison:**	-1 point

A square's content is consumed when the agent enters it, so the square will be empty if the agent later re-visits the same square.

A good agent will want to eat the food and avoid the poison, while at all costs staying clear of the walls. To help enable this behavior, the agent is equipped with three sensors that face forward, left and right relative to the agent's current heading. Each sensor is able to detect the content of the neighboring square in its particular direction. An illustration of the Flatland world along with a schematic of the agent is shown in Figure 1a.

For each step of the simulation, the agent has to choose one of the following three actions: Move forward, move left, or move right. The agent will then rotate to face in the given direction and move into the square ahead. Note that there is no "stop" action, nor any "move backward" action—the agent might therefore find itself in a situation where it is forced to eat poison in order to proceed.

The agent is allowed 50 steps to accumulate as many points as possible.

Visualization

Your system should be able to produce visualizations of the Flatland world and a given agent's behavior in it. Whether you want to produce this visualization by using a separate tool after an agent has been fully trained, or provide a periodically updating visualization during the training process itself, is up to you. You need to visualize the contents of the squares in the Flatland world, as well as clearly showing how the agent behaved throughout the 50 steps of the simulation.

It is up to you whether you want to show the agent's behavior by animating between the different steps, or rendering the full trajectory of the agent as a line in a single, static plot. An example of the latter is shown in Figure 1b. If you choose to animate, the speed of the animation must be easily configurable. If you choose to make a static visualization, keep in mind that you must somehow indicate the locations where there used to be food/poison that has now been consumed by the agent.

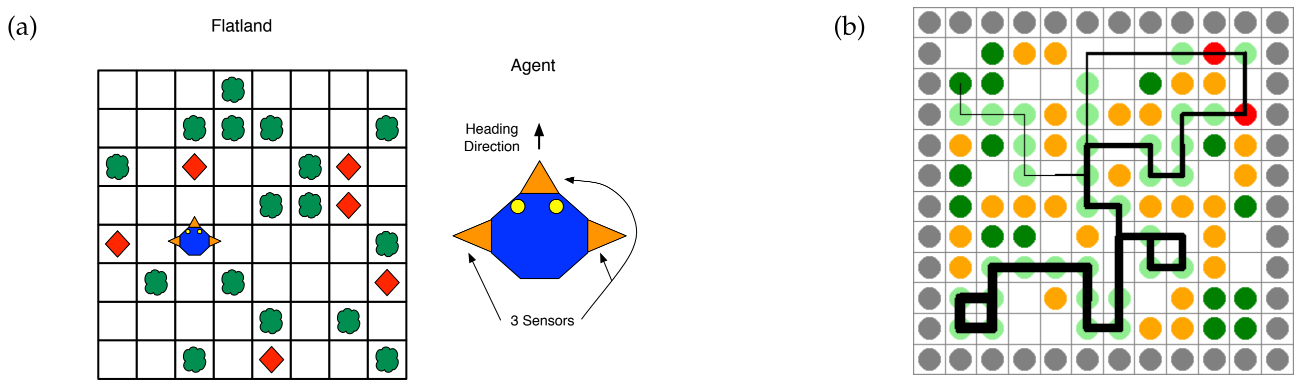

Figure 1: (a) An example Flatland board and a schematic of the agent, showing its left, forward and right sensors. (b) An example visualization of a Flatland board, with the agent's behavior drawn as a black line. The line's thickness increases to indicate later steps. The colors are as follows: ● Food ● Eaten food ● Poison ● Eaten poison ● Wall ● Agent's path

Task 1: Simulation and visualization of Flatland + baseline agent (5 points)

Implement a simulator for the Flatland environment according to the rules described above. The Flatland worlds should be generated randomly as outlined above.

Implement a “baseline” agent which will later be used as a teacher for the supervised learning agent. This agent will not use neural networks, but will be a piece of “normal” computer code that makes a decision about whether to move left, right or forward based on the sensory inputs available to the agent. The details of how this agent makes its decision is up to you, but it should strictly be a function of the current sensory inputs and not use e.g. randomness. A sketch of the algorithm could be as follows: *Never move into a wall, always move towards food, and only ever move towards poison if no other options are available.*

Implement a visualization of the Flatland world and the agent’s behavior in it, as described above. During the demo, we may ask you to repeatedly produce visualizations of new randomly generated Flatland worlds. Your program must calculate and output the total sum of points received by the agent after the 50 steps.

As a guide, if your baseline agent is properly implemented you should be able to get an average score around 20-21 points (calculating the mean score value after running many separate trials). Note that the variance is quite high, so you should run e.g. 1000 separate trials before averaging.

Deliverables: For this task you may be awarded up to 3 points for your report answers and up to 2 points during the demo session. Include the following in your report:

- (1 point) Provide a brief overview of your implementation—programming language, important files, classes and methods, etc. Include a screenshot of your Flatland visualization.
- (1 point) Describe how your baseline agent decides whether to move left, forward or right.
- (1 point) What is the average score achieved by your baseline agent over many trails, e.g. 1000 trials?

Task 2: Supervised learning of a neural agent controller (5 points)

You will now implement a neural network to use as your agent controller, and then train this neural network using supervised learning to replicate the behavior of your baseline agent from task 1. The structure of the neural network is quite simple: There is an input layer that conveys information from the sensors, and an output layer that decides the agent’s action. The two layers are fully connected. See Figure 2 for an illustration of the network structure.

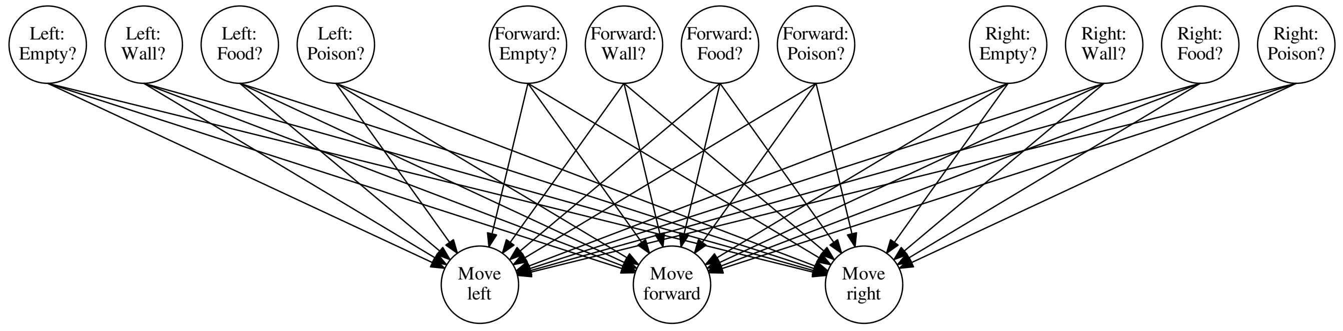

Figure 2: The neural network structure that will be used throughout this project.

Input layer

For each square observed by the agent’s sensors (left, forward and right), there are *four* input neurons. These four neurons together describe the state of the corresponding square using *one-hot encoding*. That is, for each possible state of the square (empty, food, poison, wall), there is a dedicated neuron which activates only in that situation. Active and inactive neurons output values of 1 and 0, respectively.

Output layer

There are three output neurons—one for each possible movement direction. The neurons sum up their incoming activation signals—which is the output value of each input neuron multiplied by the specific weight for that connection—and uses that sum as their own output values. In other words we do not use any activation functions in these neurons. Written as an equation, the output value y_i of output neuron i is calculated as

$$y_i = \sum_j w_{ij} \cdot x_j, \tag{1}$$

where j iterates over all input neurons, x_j is the value of input neuron j, and w_{ij} is the weight between output neuron i and input neuron j. The agent selects its action according to which output neuron has the strongest output signal.

Weights

The input layer and the output layer are fully connected, so in total there are $12 \cdot 3 = 36$ weights in the network. All of these weights must be trained. The initial value for each weight should be set to a random value of low magnitude (e.g. up to 0.001).

Learning rule

Because we don't use any hidden layers and no activation functions, we can use the *delta rule* (also known as the *Widrow-Hoff rule*) to train the weights in the network. After each step made by the agent, the weights are increased by Δw_{ij} (in other words, $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$), where the Δw_{ij} value is calculated as follows:

$$\Delta w_{ij} = \eta \cdot \delta_i \cdot x_j. \tag{2}$$

Here, η is the *learning rate*, which can be set to e.g. 0.01, x_j is the current output value from input neuron j, and δ_i is the *delta* value for output neuron i. This delta value should convey the amount of error in the current output from neuron i. Typically this is calculated as $\delta_i = t_i - y_i$, i.e. simply the difference between the desired output value t_i and the actual output value y_i. In our case, we don't have any particular requirements for what the specific output value t_i should be—all we care about is that the network "makes the correct choice", i.e. gives the *highest* output value to the correct choice. For that reason, we will therefore use the following formula for our delta values: [1]

$$\delta_i = -\frac{e^{x_i}}{\sum_k e^{x_k}} + CorrectChoice(i), \tag{3}$$

where k iterates over all the output neurons. $CorrectChoice(i)$ is 1 if the "teacher" (i.e. the baseline agent) chose action i, and 0 otherwise.

Training procedures

Each training round should consist of a large number of randomly generated Flatland worlds, such as 100. The Flatland agent is controlled by the neural network (not by the teacher). However, for each step evaluated by the neural network-based agent, *the same sensory inputs are also fed to the teacher*. The teacher is not otherwise "connected" to the Flatland environment, so it does not control the agent directly, but whatever decision is made by the teacher will be used to determine the $CorrectChoice(i)$ values during training. One round of weight updates is performed for each step made, and this continues for the usual 50 steps allowed in the Flatland environment.

After each training round, your program should output the mean score achieved across all of the Flatland worlds trained on during that round. Your supervised learning agent should be able to consistently reach the performance level of the baseline agent after a sufficient number of training rounds.

Deliverables: For this task you may be awarded up to 2 points for your report answers and up to 3 points during the demo session. Include the following in your report:

- (1 point) Include a few lines of code from the part of your program where the delta values (δ_i) are calculated and explain how this code implements Equation 3.
- (1 point) Plot how your agent's performance develops as the number of training rounds increases. The score value plotted for a given training round should be the average score over all of the Flatland worlds trained on during that round (which should be a large number of Flatland worlds such as 100).

[1] This is the "cross-entropy loss" for a "softmax classifier". For more information, interested readers should check out `http://cs231n.github.io/linear-classify/#softmax` and `http://cs231n.github.io/neural-networks-case-study/#grad`.

Task 3: Reinforcement learning of a neural agent controller (5 points)

You will now adapt the supervised learning agent to learn without using the teacher, but only by observing the rewards and punishments it receives from the environment (i.e. the points received by the agent according to whether the visited square is empty, wall, food or poison). The updated agent will therefore be using a variant of reinforcement learning.

The specific learning rule to be used by the agent is that of *Q-learning*. Q-learning can be formulated as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{4}$$

In the above, $Q(s,a)$ represents the agent's estimate of the expected future reward from performing action a in state s, α is a learning rate, r is the reward received after performing action a in state s and γ is a discount factor that can be set to e.g. 0.9. Reinforcement learning occurs by gradually refining the agent's estimates of the $Q(s,a)$ values—this takes place in the formula by adjusting $Q(s,a)$ to get closer to the expression $r + \gamma \max_{a'} Q(s',a')$. In other words, the full bracketed expression is an error value for $Q(s,a)$—let's call this the "Q error".

We will use the neural network to represent the $Q(s,a)$ function—the Q value for a particular state s and action a will be the output value from the output neuron corresponding to action a when the sensory readings corresponding to state s is presented at the input neurons. We can train the neural network to do this by using the Q error as the delta value:

$$\delta_i = r + \gamma \max_{a'} Q(s',a') - Q(s,a) \tag{5}$$

We only wish to change the weights for the output neuron that corresponds to action a, so the rest of the δ_i values will be 0 (for the output neurons that represent other actions than a).

In Equation 5, $Q(s,a)$ is the current output value of the output neuron under consideration, i.e. equivalent to y_i. The $\max_{a'} Q(s',a')$ expression evaluates to the maximum output value from the three output neurons in the updated state of the agent. In other words, to get the correct value for $\max_{a'} Q(s',a')$ you need to update the Flatland environment to reflect the fact that the agent actually did execute action a in state s (reaching state s'), determine the new sensory inputs for that location, feed that to the neural network and then determine which output neuron is the most strongly activated under these new conditions. The output value of that neuron is then your final value for the expression $\max_{a'} Q(s',a')$. *However*, you then need to reset the activation values in the neural network back to the conditions of state s before you apply the delta learning rule (Equation 2). An outline of the steps needed for each iteration is thus as follows:

1. Initially, the agent is at location A. Calculate the sensory inputs to the neural network—this is s.
2. Evaluate the output values for the three output neurons of the neural network when s is the input to the network. The maximally activated output neuron will be the action chosen by the agent—this is a. The output value of that output neuron is $Q(s,a)$.
3. As a result of the agent choosing action a, it will move to a new location B. Update the Flatland environment accordingly. The agent obtains a reward according to the contents of the square at location B—this reward is r. The agent will experience new sensory inputs when it is now located at location B—this is s'. Calculate the new output values from the neural network when s' is used as input. Determine which output neuron is the most strongly activated—whatever action corresponds to that output neuron is then a', and the output value from that neuron is equivalent to $\max_{a'} Q(s',a')$.
4. You now have all the values necessary for computing the delta value for the output neuron that corresponds to action a (by using Equation 5). The delta values for the rest of the output neurons are all 0. Next you must *re-evaluate* the neural network using the old state s as the input—this is needed in order to reinstate the old activation values in the network, which will be used by the delta learning rule. Only after doing this, can you then perform weight updates by using Equation 2.

Ideally, by properly structuring your code, you should be able to support the supervised and the reinforcement learning agents by using a common implementation for both and only differing in the code that corresponds to evaluation of respectively Equation 3 and Equation 5.

Your reinforcement learning agent should be able to consistently reach the performance of level of the baseline agent and/or the supervised learning agent.

Deliverables: For this task you may be awarded up to 2 points for your report answers and up to 3 points during the demo session. Include the following in your report:

- (1 point) Include a few lines of code from the part of your program where the delta values (δ_i) are calculated and explain how this code implements Equation 5.
- (1 point) Plot how your agent's performance develops as the number of training rounds increases.

Task 4: Extending the sensor range of the reinforcement agent (2 points)

An advantage of using reinforcement learning is that we now can feed the agent more sensory information that was not used by the baseline/supervised agents, and hopefully see that the reinforcement learning algorithm is able to learn to exploit this extra information for improved performance.

Specifically, extend the range of the sensors from 1 square to 3 squares, so that the total number of squares observed by the agent is now 9 rather than 3—it can now see 3 steps ahead in each of the 3 directions left, forward and right. The number of input neurons thus increases from $3 \cdot 4 = 12$ to $9 \cdot 4 = 36$, and the number of weights in the network increases from $12 \cdot 3 = 36$ to $36 \cdot 3 = 108$. Train the network in otherwise the same fashion as in task 4 and plot the resulting score curve. Do you see any improvement in the agent's performance level?

Deliverables: For this task you may be awarded up to 1 point for your report answers and up to 1 point during the demo session. Include the following in your report:

- (1 point) Plot how your agent's performance develops as the number of training rounds increases.

Task 5: Analysis (3 points)

As part of your report, you will now analyze one of the neural networks you trained. Answer the following items:

- (1 point) List all of the weights in the neural network. Make sure this is readable, e.g. by grouping the weights by output neuron and input sensor. You may also visualize the network in a graph (optional).
- (1 point) Select a few (at least 5) input situations and describe how the network responds (output values and which action is selected). Account for how the network generates its behavior (e.g. any structure you might find in the weights, etc.).
- (1 point) How did your agents' results progress from task 1 (baseline agent) to task 2 (supervised), task 3 (reinforcement) and task 4 (reinforcement with extended sensor range)? Was this as expected?

Report, demo and delivery

Report: You should write a report answering the points listed throughout above. The report can give a maximum of 11 points. **Your report must not exceed 2 pages in total.** You can e.g. combine the plots into one in order to save space. Overlength reports will result in points being deducted from your final score. Bring a hard copy of your report to the demo session. **Print on both sides of the sheet** to avoid having to staple the report.

Demo: There will be a demo session where you will show us the running code and we will verify that it works. This session can give a maximum of 9 points.

Delivery: You should deliver your report + a zip file of your code on itslearning. The deadline is given on the assignment on itslearning. The 20 points total for this project is 20 of the 100 points available for this class. **For this project you must work alone.**