Reinforcement Learning: Homework Assignment 1 (Semester 2 - 2016/17)

Subramanian Ramamoorthy and Svetlin Penkov

7 February 2017

Instructions:

- This homework assignment is to be done *individually*, without help from your classmates or others. Plaigarism will be dealt with strictly as per University policy.
- Solve all problems and provide your **complete** solutions (with adequate reasoning behind each step, and citations where needed) in a computer-printed form.
- This assignment will be marked out of a 100 points, and will count for 10% of your *final* course mark. It is due at 4 pm on 27 February 2017.

1 Multi-armed Bandits [30 points]

1. Show that in the case of two actions, the softmax operation using the Gibbs distribution becomes the logistic, or sigmoid, function commonly used in artificial neural networks. What effect does the temperature parameter have on the function?
2. Consider the optimistic initial value example, (fig. 2.4 in Sutton and Barto's book - based on numbering in the first edition). This represents averages over 2000 individual, randomly chosen 10-armed bandit tasks, so the result should be reliable. How do you explain the oscillations and spikes in the early part of the curve for the optimistic method? What makes this method perform differently on particular early plays?

Figure 1: Screenshot from the Enduro game.

2 Value Functions [20 points]

1. In the grid world example (figure 3.8 in Sutton and Barto's book - based on numbering in the first edition), the optimal value of the best state of the gridworld is given up to one decimal place (as 24.4). Use your knowledge of the optimal policy and the definition of the discounted return (equation 3.2, same version of the book) to express this value symbolically, and then to compute it to three decimal places.

3 Q-Learning to play the Enduro game [50 points]

You are asked to design and implement a Q-learning based agent to play the game of Enduro within the Arcade Learning Environment. Enduro is a racing game based on participation in an endurance race and the objective is to pass a certain number of cars by manoeuvring and avoiding collisions. You should implement a learning agent based on the standard Q-learning algorithm, modelling the agent's environment (i.e., the road and other cars) as a grid world, a discrete state space with a corresponding discrete action set.

3.1 Setup

For this question you will be using a Python package which you can find at `www.github.com/ipab-rad/rl-cw1`. It should run out of the box on a DICE machine, however, would you like to install it on your own computer, then follow

the instructions in the repository `README.md` file. Download the package with:

```
$ git clone https://github.com/ipab-rad/rl-cw1.git
```

The package enables you to run the Enduro game and extract a discretised grid from the game. In order to familiarise yourself with the game run the `KeyboardAgent`, which will allow you to control the racing car from the keyboard using the 'WASD' keys:

```
$ cd rl-cw1
$ python keyboard_agent.py
```

On each key press the game will advance to the next state and you will also see the updated environment grid. The total reward obtained by the agent in the current episode is also printed in the terminal. You will notice that whenever you take over a car there is a $+1$ reward and if you are taken over by an opponent then you receive -1 reward.

If you explore the `keyboard_agent.py` file you will see that that the package provides the base class `Agent` which you can derive from in order to create agents that play the game. Make sure you read the entire `README.md` file and understand how the `KeyboardAgent` works before you proceed with the question.

3.2 Discretisation

The underlying dynamical system you are controlling is a continuous one, as mentioned above. Discuss the effect of your modelling assumptions on the design and performance of your agent.

3.3 Random agent

Using the code skeleton in `random_agent.py` implement an agent which follows a uniformly random policy. Run the agent for 100 episodes and plot the total reward obtained for each episode, as well as the resulting distribution. Report the mean and the variance of the total reward obtained per episode.

3.4 Q-learning Agent

Using the code skeleton in `q_agent.py` implement an agent based on the Q-learning algorithm. Compare the performance of the Q-learning agent to the baseline random agent.

3.5 Time horizon

Vary the look-ahead horizon, hence dimensionality of your state space, and present a comparative analysis of the effect of this change by plotting and discussing the differences between learning curves in these settings.

3.6 Additional features

If your model could include features beyond the immediate neighbourhood on the road, what would be the effect on learning and task performance? Suggest at least one such feature and write down the complete specification of the MDP based on this design. *Optionally*, reimplement your learning algorithm to include this feature and comment on the resulting performance based on learning curves.