MLMI7 Reinforcement Learning and Decision Making: Coursework

David Krueger and Carl Edward Rasmussen

Due: Thursday March 10th 2022 at 12:00 noon via moodle

In this assignment, you will implement the basic building-blocks of Reinforcement Learning in Python for a discrete grid-world model. You need to write a report of upto 2000 words and include the wordcount. We provide code for constructing the worlds and visualising them.

Your answers should contain the key commands implementing the algorithm, and discussion of them. You should not include a full listing of your code including all the necessary bookkeping details etc, only the key steps necessary for supporting your discussion. You must also give an interpretation of what the numerical values and graphs you provide *mean* – why are the results the way they are? Each question should be labelled and answered separately and distinctly.

We provide the python code `model.py` and `world_config.py` which are used to create grid worlds including transition probabilities and rewards. Three specific worlds we will use are `small_world`, `grid_world` and `cliff_world`. You can use `plot_vp.py` to visualise values and policies, although note that you may need to look at `world_config.py` to understand the dynamics of the environments (e.g. in `cliff_world` when the agent steps off the cliff, it returns to the start state). We also provide `policy_iteration.py` which illustrates how the model and plotting code can be used. Familiarise yourself with this code and the environments before you start.

a) 20%: Implement Value Iteration, and verify that you get the same policy as when using Policy Iteration. What is a suitable stopping criteria for Value Iteration? Contrast this with Policy Iteration. Compare synchronous and asynchronous Value Iteration, which is more efficient and why? Compare the computational efficiency of Policy Iteration and Value Iteration (for the computational cost use the algorithmic cost, rather than the actual time your code takes to run).

b) 20% : Implement the SARSA algorithm and run it on `small_world`. Discuss how to set the learning rate α, the exploration parameter ϵ, the maximum number of iterations per episode and the number of episodes; elaborate on what happens if these are too small or too large, and how you find reasonable values. Discuss the learned policy. Implement Expected SARSA; which algorithm is more efficient?

c) 20% : Implement Q-learning algorithm and run it on `small_world`. How do you set the parameters? Compare the learned policy and the efficiency to SARSA.

d) 20% : Modify the SARSA and Q-learning algorithms to save the accumulated reward per episode. Run these algorithms using the `cliff_world` model and plot the rewards obtained in each episode throughout the learning process. This should look similar to Figure 6.13 from the Sutton and Barto text (`http://incompleteideas.net/book/ebook/node65.html#fig:cliff`). Your plot may look slightly different (and likely less smooth) due to the values you choose for ϵ and α. Describe any significant differences between the learning behavior and performance of SARSA and Q-learning. Also describe the policies learned by both algorithms at the end of training and estimate their performance. Explain any significant differences in the policies' behavior and/or performance.

e) 20% : Consider a variant of `small_world` where the 3 barrier cells only *sometimes* contain barriers. In each episode, whether each of the 3 cells contains a barrier is sampled at independently at uniform random and remains fixed throughout the episode. The new state space is $\bar{\mathcal{S}} = \mathcal{S} \times \mathbb{Z}_2^3$, so that agent knows not only its current location, but also which cells contain barriers. Explain why function approximation could be useful in this environment. Would linear function approximation suffice? Why or why not?