Imperial College London

November 11, 2019

Department of Computing

BBB – Bits, Brain & Beahviour– Dr Aldo Faisal

Assessed coursework 2 (Version 1.0)

To be returned as online submission.

The final document should be submitted in **PDF** format, preferably in Latex. If you need to program your answers, please paste the completed annotated source code in the appendix of your submission.

Your answers should be yours, i.e., written by you, in your own words, showing your own understanding. You must produce your own code and submit it when appropriate. Figures should be clearly readable, labelled and visible. Your coursework should not be longer than 4 single sided pages with at least 2 centimetre margins all around and 12pt font (again, 4 pages is maximum length, shorter courseworks are desirable, code appendix does not count towards page limit). You are encouraged to discuss with other students, but your answers should be yours, i.e.,written by you, in your own words, showing your own understanding. You should produce your own code and you can reuse code you developed in the lab assignments. If you have questions about the coursework please make use of the labs or Piazza, but note that GTAs cannot provide you with answers that directly solve the coursework

Marks are shown next to each question. Note that the marks are only indicative.

Please submit this coursework on Blackboard. All coursework will be marked together by all GTAs to ensure consistency of marks agnostic of department/course/etc.

The goal of this assignment is to apply reinforcement learning methods to a simple card game that we call *Easy21*. This task is similar to the Blackjack example in Sutton and Barto—please note, however, that the rules of the card game are different and non-standard.

- The game is played with an innite deck of cards (i.e. cards are sampled with replacement)
- Each draw from the deck results in a value between 1 and 10 (uniformly distributed) with a colour of red (probability 1/3) or black (probability 2/3).
- There are no aces or picture (face) cards in this game
- At the start of the game both the player and the dealer draw one black card (fully observed)
- Each turn the player may either stick or hit
- If the player hits then she draws another card from the deck
- If the player sticks she receives no further cards
- The values of the player's cards are added (black cards) or subtracted (red cards)

- If the player's sum exceeds 21, or becomes less than 1, then she "goes bust" and loses the game (reward -1)
- If the player sticks then the dealer starts taking turns. The dealer always sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome — win (reward +1), lose (reward -1), or draw (reward 0) — is the player with the largest sum.

Implementation of Easy21

Write an environment that implements the game Easy21. Specially, write a function `step` which takes as input a state s (dealer's first card 1-10 and the player's sum 1-21), and an action a (hit or stick), and returns a sample of the next state s' (which may be *terminal* if the game is finished) and reward r. We will be using this environment for model-free reinforcement learning, we recommend that you do not explicitly represent the transition matrix for the MDP. There is no discounting ($\gamma = 1$). You should treat the dealer's moves as part of the environment, i.e. calling *step* with a *stick* action will play out the dealer's cards and return the final reward and terminal state. For those that choose to implement the task in Python we have provided a file `easy21.ipynb` that provides a skeleton environment and useful functions stubs.

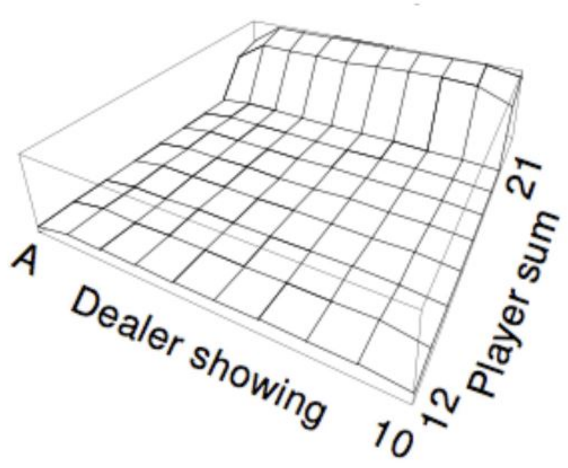
Question 1: Monte-Carlo Control in Easy21 [20]

Use a form of Monte-Carlo control to learn to play Easy21. Initialise the value function to zero.

a. Briefly explain your implementation and design choices.

b. Plot the learning curve (reward vs episodes) showing the mean and standard deviation for an appropriate number of repeated runs.

c. Plot the optimal value function $V^*(s) = \max_a Q^*(s, a)$ using similar axes to the following figure taken from Sutton and Barto's Blackjack example.



Question 2: TD Learning in Easy21 [30]

Use SARSA to play Easy21. Initialise the value function to zero.

a. Briefly explain your implementation and design choices.

b. Plot the learning curve (reward vs episodes) showing the mean and standard deviation for an appropriate number of repeated runs. How does this curve depend on your strategy for handling α?

c. Plot the optimal value function $V^*(s) = \max_a Q^*(s, a)$.

Question 3: Q-Learning in Easy21 [30]

Use Q-Learning to play Easy21.

a. Briefly explain your implementation and design choices.

b. Plot the learning curve (reward vs episodes) showing the mean and standard deviation for an appropriate number of repeated runs. How does this curve depend on your strategy for handling ϵ-greedyness?

c. Plot the optimal value function $V^*(s) = \max_a Q^*(s, a)$.

Question 4: Compare the algorithms [20]

Compare the learning curves of your 3 implementations and briefly discuss how these are explainable by differences in the algorithms.