Reinforcement Learning Assignment: Easy21
Deadline: Wednesday April 6

The goal of this assignment is to apply reinforcement learning methods to a simple card game that we call *Easy21*. This exercise is similar to the Blackjack example in Sutton and Barto 5.3 – please note, however, that the rules of the card game are different and non-standard.

- The game is played with an infinite 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.

1 Implementation of Easy21

You should write an environment that implements the game Easy21. First write a *draw* function to draw a card from the deck. Next, write a *step* function 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, and you should 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.

Provide observed output frequencies from these functions in test files. The first file, named *checkDraw*, lists the observed frequencies of each card from 1000 calls of draw. Each line of this file has three numbers, the card value (a number between 1 and 10), the color (use -1 for red and 1 for black), and the observed frequency (a number between 0 and 1). Provide similar test files for the step function, starting from the four different inputs (dealerCard, playerSum, action) of (1,1,hit), (1,10,hit), (1,18,stick), (10,15,stick). Name the files *checkStepDealer#Player#Action#*, using hit=0 and stick=1 as numbers for the actions. (e.g. the last

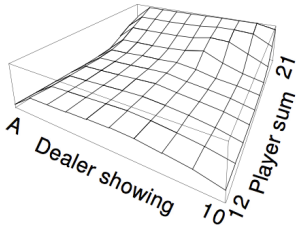
file is *checkStepDealer10Player15Action1*). Each file has the observed output frequencies from 1000 calls to step. Provide one outcome per line with four numbers: the dealer card, player sum, reward, and observed frequency. Represent the terminal state with zeros for both the dealer card and player sum.

10 marks

2 Monte-Carlo Control in Easy21

Apply Monte-Carlo control to Easy21. Initialise the value function to zero. Use a time-varying scalar step-size of $\alpha_t = 1/N(s_t, a_t)$ and an ϵ-greedy exploration strategy with $\epsilon_t = N_0/(N_0 + \min_a N(s_t, a))$, where $N_0 = 10$ is a constant, and $N(s, a)$ is the number of times that action a has been selected from state s. Get a good estimate of Q^* from 1 million episodes, adapting the number of episodes or other parameters as needed. 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.

Provide your estimate of Q^* in a test file named *c*heckQ, with the one state and action estimate per line. The four numbers on each line should be the dealer card, player sum, action (encoded as in the first part), and the action value.



15 marks

3 TD Learning in Easy21

Implement Sarsa(λ) for Easy21. Initialise the value function to zero. Use the same step-size and exploration schedules as in the previous section. Run the algorithm with parameter values $\lambda \in \{0, 0.1, 0.2, ..., 1\}$. Stop each run after 10000 episodes and report the mean-squared error $\sum_{s,a}(Q(s, a) - Q^*(s, a))^2$ over all states s and actions a, comparing the values $Q^*(s, a)$ computed in the previous section with the estimated values $Q(s, a)$ computed by Sarsa. Plot the mean-squared error against λ. For $\lambda = 0$ and $\lambda = 1$ only, plot the learning curve of mean-squared error against episode number.

15 marks

4 Linear Function Approximation in Easy21

We now consider a simple value function approximator using coarse coding. Use a binary feature vector $\phi(s, a)$ with $3 * 6 * 2 = 36$ features. Each binary feature has a value of 1 iff (s, a) lies within the cuboid of

state-space corresponding to that feature, and the action corresponding to that feature. The cuboids have the following overlapping intervals:

$$\begin{aligned} dealer(s) &= \{[1,4],[4,7],[7,10]\} \\ player(s) &= \{[1,6],[4,9],[7,12],[10,15],[13,18],[16,21]\} \\ a &= \{hit, stick\} \end{aligned}$$

where

- $dealer(s)$ is the value of the dealer's first card (1–10)
- $sum(s)$ is the sum of the player's cards (1–21)

Repeat the Sarsa(λ) experiment from the previous section, but using linear value function approximation $Q(s,a) = \phi(s,a)^\top \theta$. Use a constant exploration of $\epsilon = 0.1$ and a constant step-size of 0.01. Plot the mean-squared error against λ. For $\lambda = 0$ and $\lambda = 1$ only, plot the learning curve of mean-squared error against episode number.

15 marks

5 Discussion

Discuss the choice of algorithm used in the previous section.

- What are the pros and cons of bootstrapping in Easy21?
- Would you expect bootstrapping to help more in *blackjack* or *Easy21*? Why?
- What are the pros and cons of function approximation in Easy21?
- How would you modify the function approximator suggested in this section to get better results in Easy21?

5 marks

Total 60 marks

6 Submission

- You should submit a single pdf document containing your plots and discussion, and a single archive containing all your source code and output test files.
- In your writeup, clearly indicate which piece of code was used to answer each section. Describe how to run the code, and any non-standard libraries.
- Send your submission to Zbigniew Wojna (*zbigniewwojna@gmail.com*) with the subject line "RL COMPGI13 Assignment".

- Late assignments will be penalised according to UCL guidelines (https://www.ucl.ac.uk/srs/academic-manual/c4/ug-assessment/penalties#top).
- This assignment should be completed *individually*