Reinforcement Learning Lab Session 6

Benjamin Vandersmissen

April 2, 2020

General Information

This lab session will cover all the Reinforcement Learning material from courses 1 to 4. It will be graded lab sessions. You are free to use any content you want regarding the questions, but please refrain from using outside code beyond the previous lab sessions.

It should be doable in 4 hours, but the deadline will be set to Sunday 29th. As usual, for there on, each day of delay will remove 2.5/10 points from your grade.

Submission – You will have to submit both a report and code through Blackboard. Use the code available on the git at http://github.com/Louis-Bagot/RL_Lab/lab_session6 or https://github.com/Rajawat23/RL_Lab/lab_session6.

Make a copy of this LaTeX document from folder **report**, and fill it according to the instructions below. Please to not alter this base document, only add your content.

Question – *Questions will look like this. For questions, write the answer just below where the (your answer here) appears.*

Programming – *Programming exercises will look like this. For programming exercises, read the* `instructions.md` *of the corresponding section, and then fill in the corresponding TODO flags in the code. If this text asks for a plot, copy the plot output of your code in the document, in the figure space provided (in addition to leaving it in the plots folder in the code). If the text asks for an explanation of the results, write it as you would answer a question.*

Contents

1 Bandits

Question 1 – *Explain, in a few lines, the k-armed Bandit problem. Give an example of a real-world application, detailing actions and rewards.*

The k-armed Bandit problem is a problem where we want to determine the winning rate for k different one-armed bandits. These are gambling machines with each a different pay-out rate and we want to optimise our pay-out, by finding the machine with the highest value / pay-out rate.

A real world example could be in a soccer club, with 3 different keepers, playing the different keepers on the pitch. The actions would then be selecting a keeper, the reward would be (minus) the goals conceded. Of course this makes the assumption that each match is exactly as difficult et cetera, but this is a massive simplification.

Question 2 – *Derive the incremental updates of the Sample Average method.*

$avg\_reward(t_{i+1}) = (R_1 + ... + R_{i+1})/(i+1)$
$avg\_reward(t_{i+1}) = (i * avg\_reward(t_i) + R_{i+1})/(i+1)$
$avg\_reward(t_{i+1}) = ((i+1) * avg\_reward(t_i) - avg\_reward(t_i) + R_{i+1})/(i+1)$
$avg\_reward(t_{i+1}) = avg\_reward(t_i) + 1/(i+1) * (R_{i+1} - avg\_reward(t_i))$

Question 3 – *Explain, with your own words, the difference between Sample Average and Weighted Average methods*

In Sample average, each sample in time is equally valuable as the other samples. In Weighted average, this is not the case, the samples that are more recent have a heigher influence on the average, because of the weight.

Question 4 – *Explain the impact of the hyper-parameters of the following algorithms: ϵ in ϵ-greedy, c in UCB, and α in Gradient Bandit. Use extreme values as examples.*

ϵ is the parameter that defines the explore rate, if ϵ is very high, we explore a lot and don't exploit very much. For example, if $\epsilon - 1$, we always explore and never exploit.

c in UCB is a constant that determines the influence of the parameter $\sqrt{ln(t)/N_t(a)}$. This parameter is high if the action hasn't been chosen often during the runtime of the algorithm. If we have a very high c, the value of an action will just be dependent on how often it was chosen already and not on the estimated value at the instant.

α in Gradient Bandit is the update rate of for the preference for an action. This means if α is 0, the preference is never updated and if α is very high, $H_t(k+1)$ is not really dependent on $H_t(k)$ anymore.

Question 5 – *Show that the Sample Average method erases the bias introduced by the initialization of the Q estimates. What does this mean for Optimistic Greedy? Show that, despite this, Optimistic Greedy does not work on non-stationary problems.*

Assume the bias is b and the optimal value is a, then after the first iteration, $Q = (b+a)/2$, after n iterations, $Q = (b + (n-1)a)/n$. If we have n going to infinity, we can simplify this equation to $Q = a$.

This means for Optimistic greedy, that the algorithm will automatically converge to a higher optimum than if we started with $Q = 0$. This will still not work for the non-stationary problems, because if the distribution changes after a specific amount of time, the new reward has no real influence on the average reward. This is because $avg\_reward(t_{i+1}) = n/(n+1) * avg\_reward(t_i) + 1/(n+1) * new\_reward$.

Programming – *Implement a Sample Average and Weighted Average version of ϵ greedy on a Non-Stationary k-armed Bandit problem. In order to see results, run the experiment for $10k$ steps, and paste here the resulting performance plot in the Figure **??** below. Explain your results below.*

As seen in the plot, the weighted average is rising faster than the sample average. This is because the sample average is worse at adapting to a non-stationary k-bandit problem.

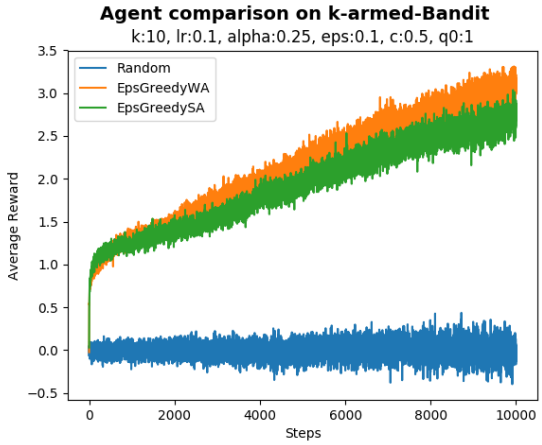

Figure 1: Comparison: ϵ-greedy algorithm with Sample Average versus Weighted Average updates.

2 Markov Decision Processes

For questions where a drawing (MDP or diagram) is required, you can use whichever of the following methods:

- a (properly cropped and clear) photo of a drawing on paper. Make sure everything is readable.
- a tikz diagram, i.e. the plotting tool for LaTeX (if you know how it works. Don't learn for this report otherwise, tikz takes an eternity)
- Mathcha, which can generate tikz or pngs. (recommended)

Question 1 – *Define a Markov Decision Process, and the goal of the Reinforcement Learning problem.*

A Markov Decision Process is a schema consisting of states and transitions. A transition goes from one state to another state following an action and has a reward from using the transition. The goal of Reinforcement learning is to optimise the total reward.

Question 2 – *Show that the MDP framework generalizes over Bandits by drawing the Bandits problem as a MDP with reward distributions r_a for each action a. Paste your drawing on Figure* **??**. *Shortly explain your submission.*

A bandit problem can be generalised by having one state and a transition for each action to the same state, with the reward of such transition being the reward one would get when using the corresponding bandit.

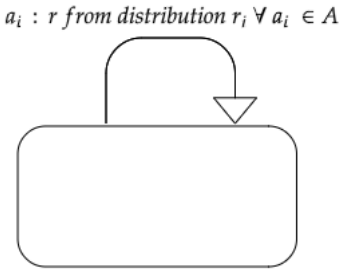

Figure 2: The MDP corresponding to the Bandit problem with reward distributions r_a, $\forall$ actions a

.

(your explanation here)

Question 3 – *Turn the following statement into a MDP, with states and transitions with actions (named), probabilities and rewards. Paste the graph on Figure* **??***; pick real values for both rewards and probabilities (no unknowns). Shortly explain your submission after the statement and plot.*

Statement:

`You go to the university using Velo - Antwerp's shared bikes. There are three stations where you can drop off your bike on the way: the park, furthest from the university; the cemetery, second furthest; and the university station, right in front of your destination. You want to take the least time possible to go to the university, and you take much longer walking than biking.`
`At any station, you can either decide to drop off your bike and walk to the university, or continue to the next station. However, it sometimes happens that the stations are full - you cannot drop off your bike there. You can either go back, or, if possible, continue. You notice that the amount of free spots in the first stations often aligns with the amount of free spots in the following stations - or is less. In order to decide whether you should drop off your bike or not, you take note of the last station's number of free spots - it can either be a lot, or a few, or none.`
`When you have to go back, we assume that people could've come to pick or drop bikes, so the transition doesn't depend on the previous state of the station.`

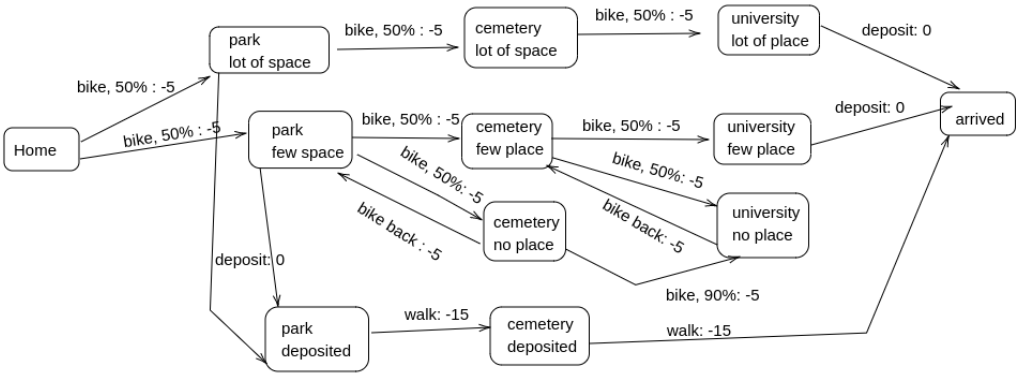

Figure 3: The MDP corresponding to the statement of Question 3

.

Assume the total reward is time lost, so each reward will be negative and we want to maximize the reward or minimize the time lost. Lets also assume each station is 5 minutes bike time or 15 minutes walking time. We assume there is always at least some place free in the first station. If a station is full, 90% chance the next station is also full. To avoid cluttering in the MDP, we omitted some transitions: a station with a lot of place has 25% chance the next will have few place and 25% chance the next has no place. from each station, you can also bike back to the previous station, but because this is only relevant if there is no place in the current station, this is omitted as well. Also cemetery lot and cemetery few have a transition to cemetery deposited with action deposit, reward 0 but this would have cluttered the MDP even more.

Question 4 – *RL has been widely popular lately because of its combination with Deep Learning (using Neural Nets as policy or value function approximators), leading to incredible performances on difficult environments like video games. One such game is the first Mario World. Show how the MDP can look like for the frames of the game. Where can stochasticity be involved?*

A state of the mdp can consist of a grid, the screen, with all the objects and enemies and the player in. The actions could be the controls a scoring function could be like in mario, each transition gets -1, a death -1000, a powerup, a coin and a kill get a positive score. The stochasticity can be the goombas movements, they randomly move in 2 directions.

3 Control

In lab session 2 and 3, the Value Iteration algorithm was stopped after a maximum number of iterations. However, this is not the natural stopping condition of the algorithm: it should stop when the value estimates have converged: $V_k = v*$. When implementing this, we define convergence of the V_{k-1}, V_k, V_{k+1}.. stream of vector values as

$$||V_{k+1} - V_k||_2 < \delta$$

Where δ is an arbitrary small constant ($\delta = 0.01$ in the code). The number of iterations of Value Iteration to convergence is a measure of the algorithm's performance.

Policy Iteration alternates evaluating a policy π until convergence to V_π, and updating the policy to be greedy over the new values, $\pi' = greedy(v_\pi)$. We define *convergence in policy* as $\pi' = \pi$ (same action in all states), and *convergence in value* as

$$||V_{\pi'} - V_\pi||_2 < \delta$$

Value Iteration only converges in value, as there is no explicit policy. When comparing convergence speed in value of Value Iteration vs Policy Iteration, make sure to compare the number of single sweeps over the state space! (iterations)

Programming – *Implement Value Iteration on the course's diamond/pit Gridworld environment (course and Lab session 2). You can reuse Environment code from before.*

Programming – *Implement Policy Iteration on the course's diamond/pit Gridworld environment.*

Question 1 – *Discuss and compare the performances of both algorithms. Under what circumstances can one come on top of the other?*

In the current gridworld, the value iteration algorithm uses 10 iteration to converge on a value with delta = 0.01. Policy iteration uses 30 iterations to converge on the same values. The advantage of policy iteration is that we will get the optimal policy as well. This is especially useful if we have a large action space, then the policy iteration will be faster and we don't need to calculate the optimal policy at the end, like in value iteration.

Question 3 – *Explain the fundamental differences between QLearning and the Value Iteration / Policy Iteration algorithms. Can you see why QLearning is more interesting in the general case?*

Value iteration & policy iteration both need to know the exact probabilities for each transition and action, which means you can't use it when these values are unknown. Q learning doesn't need these probabilities to calculate the q-values, so this approach can be used when the environment is not fully known.

4 Bonus

Programming – ***BONUS, 1.5pts*** *Implement the Gridworld drawn on Figure* **??***: a river crossing. The actions are up, down, left, right and "do nothing". The agent needs to cross a river from the lower left corner (state S) to the upper right corner (state G). This* 3×5 *Gridworld is divided on the 2nd row by a river that might push the agent right upon entering by one, two or three squares, with corresponding probabilities* $0.2, 0.5$ *and* 0.3*. If the agent is pushed off to the far right in the river, the episode ends with reward* -1*. If the agent reaches the goal, the reward is* $+1$*. Note that it is the transition to the state (i.e. "right" from (0,3) to G=(0,4)) that yields the reward, and states G and red (1,4) are terminal.*

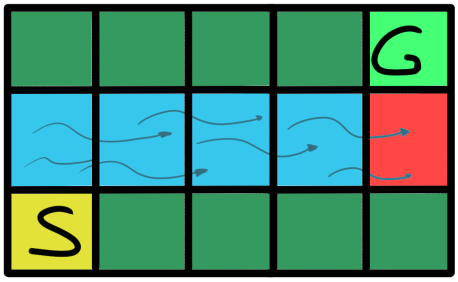

Figure 4: The River Crossing MDP

.