Q-Learning Cart Pole

Francesco Paolo Carmone
s308126
Computer Engineering in Automation and Cyber-Physical Systems
Politecnico di Torino
francescopaolo.carmone@studenti.polito.it

Task 3.1

This task revolves around the exploration-exploitation trade-off at training time. This trade-off is necessary, because even though choosing the most greedy action (*exploiting*) assures us of getting the highest *return* (i.e. cumulative reward), *exploring* enables the improvement of the estimate of the non-greedy action's values, which could lead in finding even better actions that leads to greater rewards. A ε-greedy policy looks like this:

$$\pi(a|s) = \begin{cases} \epsilon/m & \text{non-greedy action} \\ \epsilon/m + (1-\epsilon) & \text{greedy action} \end{cases} \tag{1}$$

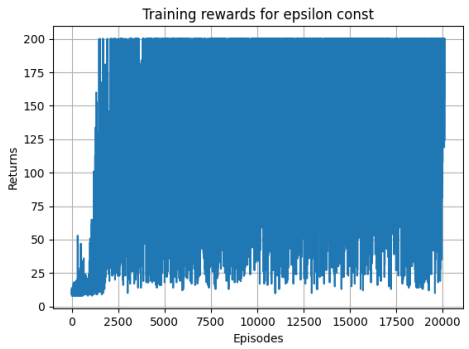
where m is the number of actions, in our case $m = 2$

Choosing an ε-greedy policy with a fixed $\varepsilon = 0.2$ means that twenty percent of the times, the algorithm will not choose the greedy action, but will instead explore. This is important especially in the early iterations when the algorithm still has taste the ground, but it becomes unnecessary once all the states have been explored. One way to improve upon that policy is to decrease ε as the time goes on. We define an ε that behaves like that *greedy in limit with infinite* exploration, or, in short, *GLIE*.

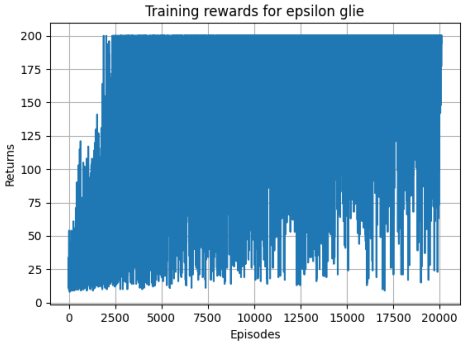
To tune b to the nearest integer such that epsilon reaches 0.1 after $k = 20000$ episodes means choosing $b = 2222$.

$$\varepsilon_k = \frac{b}{b+k} \Rightarrow b = \left.\frac{k}{1/\varepsilon_k - 1}\right|_{k=20000, \varepsilon_k = 0.1} = \lfloor 2222.\bar{2} \rfloor = 2222 \tag{2}$$

The reward of the training process using the two different εs are



(a) $\varepsilon = 0.2$



(b) $\varepsilon_k = \frac{b}{b+k}$

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

The results are overall very similar as expected, since the number of actions is merely two, and the algorithm will rapidly exhaust exploring all the possible configurations. The initial episodes and the ending ones show a sightly lower return using a constant ε if compared to ε-*GLIE*, because the algorithm explores with an higher probability the non-greedy action, and keeps doing it throughout the whole training.

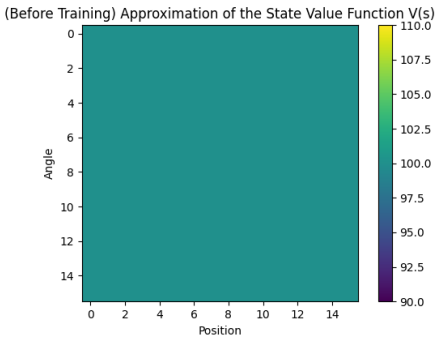
Task 3.2

In the previous task we trained a model using a greedy policy, which under the hypothesis that the model is stationary and the action space is finite, is an an optimal policy. The training provided us an *estimated action-value function*, $Q(S, A)$ which for the reason stated earlier will converge to $q_*(s, a)$, the *optimal action-value function*. Through the Bellman Optimality equations we're able to calculate $v_*(s)$, the *optimal state value function* as follows:

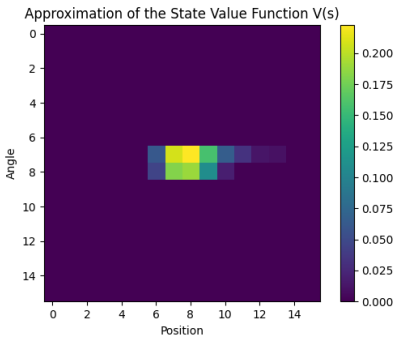
$$v_*(s) = \max_{a \in \mathcal{A}} q_*(s, a) \tag{3}$$

Before the training, all values in $Q(S, A)$ have been initialized with an initial condition, and plotting a heat map of the optimal state value function at that stage will result in a monochromatic flat map, as shown in figure 2a.

If we decide to plot it during the iterations, the grid gets updated with the rewards. Those are higher at the center of the map, attributing greater returns to the cart pole when it balances the pole ($\theta = 0$) near the central position ($x = 0$).

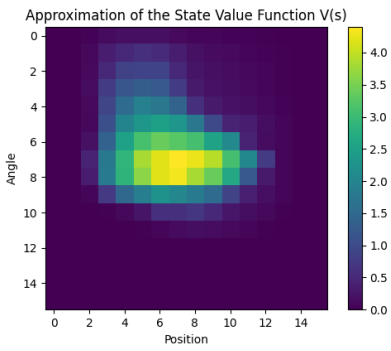


(a) All states have been initialized to 100

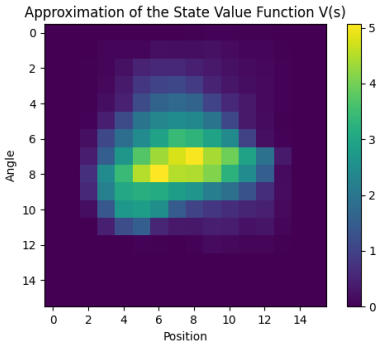


(b) After 600 iterations, null initial condition

By increasing the number of episodes in which the algorithm trains, it will keep exploring and refining the heat map. The heat maps produced with the two different ε-s are similar for reasons stated earlier.



(a) $\varepsilon = 0.2$, null initial condition



(b) $\varepsilon_k = \frac{b}{b+k}$, null initial condition

Task 3.3

By setting $\varepsilon = 0$ the policy (1) will entirely become a greedy policy, in other words it will always choose the action that has the highest return. Training a model like this, will result in a model that does not explore its surroundings, and will not be able to achieve high returns.

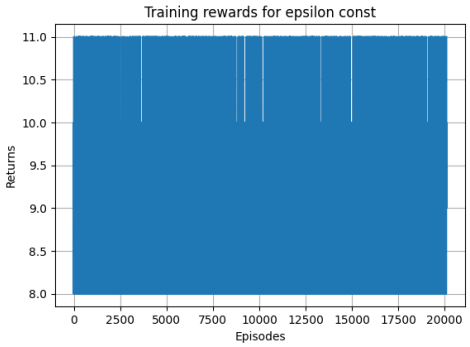
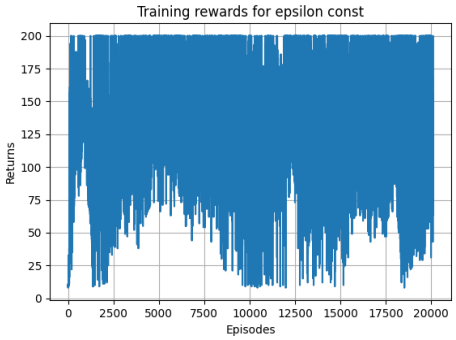
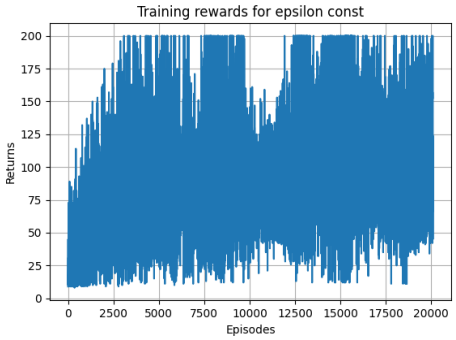


Figure 4: Model trained with an entirely greedy policy ($\varepsilon = 0$), starting from null conditions

A non null, but biased, initial value can be set to encourage initial exploration. This is well summarized in (): *We call this technique for encouraging exploration optimistic initial values.[...] Whichever actions are initially selected, the reward is less than the starting estimates; the learner switches to other actions, being "disappointed" with the rewards it is receiving. The result is that all actions are tried several times before the value estimates converge. The system does a fair amount of exploration even if greedy actions are selected all the time.*



(a) $\varepsilon = 0$. Initial condition 50



(b) $\varepsilon = 0$. Initial condition 100

Choosing 100 as initial condition with this model yields a worse result than initialising it with 50, assuming the same amount of episodes. Increasing that number would improve the returns on that case, as the algorithm has more time to explore the space.

Question 3.4

Can Q-learning be used in environments with continuous state spaces, i.e. with function approximation?

Yes, Q-learning can be used in environments with continuous state spaces as long as function approximation methods are employed, such as linear function approximation or neural networks are be used to estimate the Q-function for continuous state spaces.

Question 3.5

Can Q-learning be used in environments with continuous action spaces? If any, which step of the algorithm would be problematic to compute in the continuous action space case

The computational cost required for solving the maximization problem when computing the Q-Learning target become too demanding when working with continuous spaces. If your tool is a hammer, then everything is a nail. Discretize your action space.

References

[1] "Reinforcement Learning: An introduction (Second Edition)" by Richard S. Sutton and Andrew G. Barto http://incompleteideas.net/book/RLbook2020.pdf