Coursework (Assessed): Foundations of Reinforcement Learning

Thomas Aujoux
Paul Wattellier

February 2024

Exercise 1

(a)

Initially, one might assume that an algorithm playing against itself would resemble battling a mirror image, leading to minimal learning due to the mirroring of strategies. However, this overlooks the critical roles of probability, exploration, and exploitation in the learning process, which collectively ensure this scenario is far from simple mirroring. Self-play, in fact, fosters a dynamic environment by having both instances of the algorithm adapt to each other's continuously evolving strategies. It refines its move selection policy not just to react but to proactively counter the opponent's strategies. Through this iterative process of learning and adaptation, the algorithm's decision-making is significantly enhanced.

(b)

This approach might initially accelerate learning by exploiting known good moves. However, this strategy is double-edged. On one hand, it might lead to rapid improvements in performance as the algorithm quickly converges to a policy that maximizes immediate rewards (exploitation). On the other hand, the greedy approach risks missing out on potentially superior long-term strategies that require exploration of less immediately rewarding moves. The primary pitfall here is the potential for the algorithm to get stuck in local optima, where it repeats strategies that seem best based on limited experience but are suboptimal in the broader scope of the game. This contrasts with non-greedy players who, by exploring a variety of moves, including suboptimal ones, might discover more effective strategies over time.

(c)

Here are the states equivalent to the original state: $a_{11}a_{12}a_{13}a_{21}a_{22}a_{23}a_{31}a_{32}a_{33}$

- 90-egree Rotation: $a_{31}a_{21}a_{11}a_{32}a_{22}a_{12}a_{33}a_{23}a_{13}$
- 180-degree Rotation: $a_{33}a_{32}a_{31}a_{23}a_{22}a_{21}a_{13}a_{12}a_{11}$
- 270-degree Rotation: $a_{13}a_{23}a_{33}a_{12}a_{22}a_{32}a_{11}a_{21}a_{31}$
- Horizontal symmetry:$a_{31}a_{32}a_{33}a_{21}a_{22}a_{23}a_{11}a_{12}a_{13}$
- Vertical symmetry: $a_{13}a_{12}a_{11}a_{23}a_{22}a_{21}a_{33}a_{32}a_{31}$
- First diagonal symmetry: $a_{11}a_{21}a_{31}a_{12}a_{22}a_{32}a_{13}a_{23}a_{33}$
- Second diagonal symmetry: $a_{33}a_{23}a_{13}a_{32}a_{22}a_{12}a_{31}a_{21}a_{11}$

(d)

We can create a set that contains unique states in the sense of the symmetry and a function that would transform each state to its representation using the math concept of equivalent classes.

(e)

Leveraging symmetries in tic-tac-toe with reinforcement learning simplifies learning by treating symmetric states as identical. This reduces state space complexity, making learning more efficient and helping the algorithm converge to optimal strategies faster by generalizing learning across equivalent states.

(f)

The inherent value of symmetrically equivalent positions, in theory, should be the same because they represent the same strategic situation with just a different orientation or reflection. In very specific cases it is possible that the opponent is biased and don't see some symmetries so it could improve the performance of the algorithm to take this into account, not reducing the state space, but more often it remains advantageous for a reinforcement learning algorithm to use these symmetries.

(g)

One key method is enhancing exploration techniques, such as using ϵ-greedy or Upper Confidence Bound (UCB) algorithms, to balance exploration with exploitation more effectively.

(h)

A better approach to solving the noughts and crosses problem, as outlined in question, might involve integrating more sophisticated AI techniques such as deep reinforcement learning.

(i)

This could lead to a situation where the student is incentivized to focus on addition, which is easier, and neglect subtraction, which is more challenging yet crucial for balanced mathematical proficiency. The uniform reward structure fails to account for the varying effort and skill development required for different operations, potentially skewing the learning process towards tasks that yield rewards more easily but are less educationally valuable in the long run.

(j)

The answer is available on an attached file named question_j.py

Exercise 2

(a)

We are in the setting of a k-armed bandit problem with k = 4 different actions using the ε-greedy algorithm. In an ε-greedy algorithm, with probability 1-ε we select a greedy action based on the highest estimated action value and with probability ε we select a random action. The initial choice for the action values at time 1 is $Q_0(a) = 0$ for all actions. We have the following sequence given by the exercise:

$$
\begin{array}{llllll}
A_1 = 1, & R_1 = 1, & Q_1(1) = 1, & Q_1(2) = 0, & Q_1(3) = 0, & Q_1(4) = 0 \\
A_2 = 2, & R_2 = 1, & Q_2(1) = 1, & Q_2(2) = 1, & Q_2(3) = 0, & Q_2(4) = 0 \\
A_3 = 2, & R_3 = 2, & Q_3(1) = 1, & Q_3(2) = 3/2, & Q_3(3) = 0, & Q_3(4) = 0 \\
A_4 = 2, & R_4 = 2, & Q_4(1) = 1, & Q_4(2) = 5/3, & Q_4(3) = 0, & Q_4(4) = 0 \\
A_5 = 3, & R_5 = 0, & Q_5(1) = 1, & Q_5(2) = 5/3, & Q_5(3) = 1, & Q_5(4) = 0
\end{array}
$$

First Action = Possibly ($A_1 = 1$): All different actions start with an equal value 0. The choice can be seen as random or based on exploration. We cannot distinguish between the two possibilities.

Second Action = Definitely ($A_2 = 2$): We can see that $Q_1(1) = 1$ and $Q_1(2) = 0$, so because the argmax is not to choose action 2 this must be the result of some random exploration.

Third and Fourth Actions = Possibly (A_3, A_4): They are two possibilities, the two actions can be the result of finding the argmax but it can also be the result of a random choice. We cannot distinguish between the two possibilities.

Fifth Action = Definitely ($A_5 = 3$): Since $Q_4(3) = 0$ the action 3 has been chosen due to exploration.

(b)

Our goal is to find an equation that updates the estimate of the action value $Q_t(a)$ using the new reward R_t received after taking action a at time t. We have:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{N_t(a)},$$

where

$$N_t(a) = \sum_{i=1}^{t-1} 1_{A_i=a},$$

and $1_{\text{predicate}}$ is an indicator function that is 1 if the predicate is true, and 0 otherwise.

We aim to derive the formula for

$$Q_{t+1}(a) = Q_t(a) + \alpha_t(a)[R_t - Q_t(a)],$$

where

$$\alpha_t(a) = \frac{1}{N_{t+1}(a)}.$$

1. Current estimation of $Q_t(a)$:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{N_t(a)}.$$

2. After taking action a at time t and receiving reward R_t:
- The new total reward for action a becomes $\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a} + R_t$.
- The new count of times action a has been taken becomes $N_t(a) + 1$. ‘

3. New estimation of $Q_{t+1}(a)$:

$$Q_{t+1}(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a} + R_t}{N_t(a) + 1}.$$

4. Rewrite $Q_{t+1}(a)$ using $Q_t(a)$:

Since $Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{N_t(a)}$, we can rewrite the equation for $Q_{t+1}(a)$ in terms of $Q_t(a)$ and R_t:

$$Q_{t+1}(a) = Q_t(a) + \frac{1}{N_t(a) + 1}(R_t - Q_t(a)).$$

5. Identify $\alpha_t(a)$:

We can see that $\frac{1}{N_t(a)+1}$ is the step-size parameter $\alpha_t(a)$ in the incremental update formula. Therefore, $\alpha_t(a) = \frac{1}{N_{t+1}(a)}$.

6. Final equation:

$$Q_{t+1}(a) = Q_t(a) + \alpha_t(a)[R_t - Q_t(a)],$$

where

$$\alpha_t(a) = \frac{1}{N_{t+1}(a)}.$$

(c)

We are in a setting of weighting each reward, in the general case where the step-size parameters α_n is not constant. The formula will be different from when α is constant.

In the general case when α_n are not constant we can use this formula:

$$Q_{n+1} = Q_n + \alpha_n[R_n - Q_n],$$

Using a recursive formula for i = 1, 2, 3, 4, we can generalise a formulae with the weight w_i on reward R_i in the estimate Q_n can be formulated as:

$$w_i = \alpha_i \prod_{j=i+1}^{n} (1 - \alpha_j),$$

This expression indicates that the influence of each reward R_i on the current estimate Q_n decreases over time.

(d)

We must show that the softmax distribution used in gradient bandit algorithms becomes equivalent to the logistic function when applied to a scenario with only two actions. We can recall different definitions before showing the result.

The softmax distribution for action selection in gradient bandit algorithms is defined as:

$$P[At = a] = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}}$$

where $H_t(a)$ is the preference for action a at time t, and k is the total number of actions.

The logistic function, often used in statistics and artificial neural networks for binary outcomes, is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

We have $H_t(1)$ and $H_t(2)$ for actions 1 and 2. The softmax probabilities for these two actions are:

$$P[A_t = 1] = \frac{e^{H_t(1)}}{e^{H_t(1)} + e^{H_t(2)}}$$

$$P[A_t = 2] = \frac{e^{H_t(2)}}{e^{H_t(1)} + e^{H_t(2)}}$$

We can compute the ratio of choosing action 1 over action 2, and then transform it into the logistic function form. The ratio of choosing action 1 over action 2 is given by the ratio of their probabilities:

$$\frac{P[A_t = 1]}{P[A_t = 2]} = \frac{e^{H_t(1)}}{e^{H_t(2)}} = e^{H_t(1) - H_t(2)}$$

If we say that $x = H_t(1) - H_t(2)$. The probability of choosing action 1 can then be expressed in terms of x:

$$P[A_t = 1] = \frac{e^x}{1 + e^x}$$

This form is the logistic function of x, $\sigma(x)$, where x is the difference in preferences between the two actions. Thus, for two actions, the softmax selection probability for an action is equivalent to the logistic function of the difference in their preferences.

(e)

You can see the Notebook.

(f)

You can see the Notebook.