COMS W4701: Artificial Intelligence, Spring 2022

Homework 3

Instructions: Compile all solutions to the written problems on this assignment in a single PDF file (typed, or handwritten *legibly* if absolutely necessary). **Please show your work by either writing down the relevant equations or expressions for each problem, or by explaining any logic that you are using to bypass known equations**. Coding solutions may be directly implemented in the provided Python file(s). When ready, follow the submission instructions to submit all files to Gradescope. Please be mindful of the deadline, as late submissions are not accepted, as well as our course policies on academic honesty.

Problem 1: UCB Bandits (12 points)

In this problem you will be running several instances of the provided UCB bandit Python script. You will not be writing or turning in code, although you will be asked to show certain plot outputs in your PDF. The script simulates a n-armed UCB bandit, where the reward for each action follows a normal distribution. The arguments to the UCB bandit function are a list of n distribution means and a list of n distribution variances. The third argument is the exploration parameter c, and the fourth (optional) argument is the number of iterations for which to simulate the bandit.

Three plots are shown when the script is run. The first two show the action values Q_t and upper confidence interval (first and second terms of the UCB expression) of each arm at each iteration. The third plot shows the arm (action) taken in each iteration. Actions are represented as integers starting from 0.

(a) Simulate two 3-armed bandit experiments with means `[0, 0, 0]` and variances `[1, 1, 1]`. For experiment 1, select a value of c such that all Q_t values converge to Q^*; for experiment 2, select c such that at least one Q_t value does not converge to Q^*. Show the three plots for each experiment. Briefly explain why your c values lead to the two different outcomes. Contrast the trend of the confidence intervals and distribution of actions taken in each.

(b) Now suppose our three arms have very different means: `[5, 0, -5]`. Simulate two bandit experiments with these means, and variances `[1, 1, 1]`, once with $c = 1$ and once with $c = 10$. Again show the plots for each scenario. Comment (for each scenario) on how the Q_t values of each action update over time (if at all), how the confidence intervals evolve, and the distribution of the actions taken.

(c) Let's consider a scenario in which the means are different but very close, while variances are very large. Simulate bandit experiments with means `[1, 0, -1]`, variances `[10, 10, 10]`, and $c = 1$. Do so until you see an outcome in which the "dominant" action taken most often is not action 0 with the largest mean. Show the plots for this result and explain why we ended up converging on a suboptimal action. Briefly explain the importance of c and exploration when we have bandits with large variances.

Problem 2: Mini-Blackjack (12 points)

We will model a mini-blackjack game as a MDP. The goal is to draw cards from a deck containing 2s, 3s, and 4s (with replacement) and stop with a card sum as close to 6 as possible without going over. The possible card sums form the states: 0, 2, 3, 4, 5, 6, "done". The last state is terminal and has no associated actions. From all other states, one action is to **draw** a card and advance to a new state according to the new card sum, with "done" representing card sums of 7 and 8. Alternatively, one may **stop** and receive reward equal to the current card sum, also advancing to "done" afterward.

(a) Draw a state transition diagram of this MDP. The diagram should be a graph with seven nodes, one for each state. Draw edges that represent transitions between states due to the **draw** action only; you may omit transitions due to the **stop** action. Write the transition probabilities adjacent to each edge.

(b) Based on the given information and without solving any equations, what are the optimal actions and values of states 5 and 6? You may assume that $V^*(\text{done}) = 0$. Then using $\gamma = 1$, solve for the optimal actions and values of states 4, 3, 2, and 0 (you should do so in that order). Briefly explain why dynamic programming is not required for this particular problem.

(c) Find the largest possible value of γ that would possibly lead to different optimal actions in both states 2 and 3 (compared to those above). Compute the values of states 3, 2, and 0 for the discount factor that you found. Briefly explain why a lower value of γ decreases the values of these states but not those of the other states.

Problem 3: Dynamic Programming (12 points)

Let's revisit the mini-blackjack game but from the perspective of dynamic programming. You will be thinking about both value iteration and policy iteration at the same time. Assume $\gamma = 1$.

(a) Let's initialize the time-limited state values: $V_0(s) = 0$ for all s. Find the state values of V_1 after one round of value iteration. You do not need to write out every calculation if you can briefly explain how you infer the new values.

(b) Coincidentally, V_0 are also the values for the (suboptimal) policy $\pi_0(s) = \textbf{draw}$ for all s. If we were to run policy iteration starting from π_0, what would be the new policy π_1 after performing policy improvement? Choose the **stop** action in the case of ties.

(c) Perform a second round of value iteration to find the values V_2. Have the values converged?

(d) Perform a second round of policy iteration to find the policy π_2. Has the policy converged?

Problem 4: Reinforcement Learning (12 points)

Let's now study the mini-blackjack game from the perspective of using reinforcement learning to learn an optimal policy. Again, assume $\gamma = 1$. We initialize all values and Q-values to 0 and observe the following episodes of state-action sequences:

- 0, draw, 2, draw, 4, draw, done (reward = 0)
- 0, draw, 3, draw, done (reward = 0)
- 0, draw, 2, draw, 5, stop, done (reward = 5)
- 0, draw, 3, draw, 5, stop, done (reward = 5)
- 0, draw, 4, draw, 6, stop, done (reward = 6)

(a) Suppose that the above episodes were generated by following a fixed policy. According to Monte Carlo prediction, what are the values of the six states other than the "done" state? Explain whether the *order* in which we see these episodes affects the estimated state values.

(b) Suppose we use temporal-difference learning with $\alpha = 0.5$ instead. Write out **each** of the updates that changes a current state value. Again explain whether the *order* in which we see these episodes affects the estimated state values.

(c) Suppose we are interested in learning a better policy and start allowing for exploration. The values computed using TD learning above are assigned to the relevant Q values, while Q values that were not updated are 0. Consider using Q learning versus SARSA to update the Q values. For which state-action pairs will the converged Q values differ between the two methods and why (assuming exploration is persistent)?

Problem 5: "Real" Blackjack (52 points)

We will now extend mini-blackjack from 6 to 21 and add in a few other twists. We have 22 numbered states, one for each possible card sum from 0 to 21, in addition to the "done" terminal state. From each non-terminal state we can either **stop** and receive reward equal to the current card sum, or we can **draw** an additional card.

When drawing a card from the deck, any of the standard set of cards may show up, but the jack, queen, and king cards are treated as having value equal to 10 (aces will just be treated as 1s). We will still be drawing cards with replacement, so the probability of **drawing** a card with a value 1 through 9 is $\frac{1}{13}$ each, while the probability of obtaining a 10 value is $\frac{4}{13}$. Lastly, we will add in a constant living reward that is received with every **draw** action.

Given this information, we can thus model the problem as a Markov decision process and solve for the optimal policy and value functions. You will be implementing several of the dynamic programming and reinforcement learning algorithms in the provided blackjack Python file.

5.1: Value Iteration (12 points)

Implement `value_iteration` to compute the optimal values for each of the 22 non-terminal states. The first argument is the initial value function V_0 stored in a 1D NumPy array, with the index corresponding to the state. The other arguments are the living reward, discount factor, and stopping threshold. As discussed in class, your stopping criterion should be based on the *maximum* value change between successive iterations. Your value updates should be **synchronous**; only use V_i to compute V_{i+1}. When finished, your function should return the converged values list V^*.

5.2: Policy Extraction (10 points)

An agent would typically care more for a policy than a value function. Implement `value_to_policy`, which takes in a set of values, living reward, and discount factor to compute the best policy associated with the provided values. Return the policy as a NumPy array with 22 entries (one for each state), with value 0 representing **stop** and value 1 representing **draw**.

5.3: DP Analysis (9 points)

Now that you can generate optimal policies and values, we can study the impact of living reward and discount factor on the problem. For each of the following experiments, you can simply use an initial set of values all equal to 0.

(a) Compute and plot the values V^* and policy π^* for living reward $lr = 0$ and $\gamma = 1$. You should see that V^* consists of three continuous "segments". Briefly explain why the discontinuities between the segments exist, referring to the optimal policy found and game rules.

(b) Experiment with decreasing the discount factor, e.g. in 0.1 decrements. For sufficiently low values of γ you should see that the three segments of V^* merge into two. Show the plots for an instance of this effect and report the γ value you used. Briefly explain the changes that you observe in V^* and π^*.

(c) Reset γ to 1 and experiment with changing the living reward, e.g. using intervals of 1 or 2. Which segments of V^* shift for negative living rewards or slightly positive living rewards? In which direction do these values shift in each case? How does π^* change as these state values shift? Find approximate thresholds of the living reward in which π^* becomes **stop** in all states, and alternatively **draw** in all states.

5.4: Temporal-Difference Learning (12 points)

You will now investigate using reinforcement learning, and in particular the Q learning algorithm, to learn the optimal policy and values for our blackjack game purely through self-play. We will need to keep track of Q values; we can use a 22×2 NumPy array `Q`, so that we have `Q[s,a]`$= Q(s, a)$. `a=0` corresponds to the **stop** action and `a=1` corresponds to the **draw** action. To allow for exploration, we will use ε-greedy action selection.

The `Qlearn` function takes in an initial array of Q values, living reward lr, discount factor γ, learning rate α, exploration rate ε, and the number of transitions N. In addition to updating `Q`, your procedure should keep track of the states, actions, and rewards seen in a $N \times 3$ array `record`. After initializing the arrays and initial state, the procedure within the simulation loop is as follows:

- Use the ε-greedy method to determine whether we explore or exploit.
- If **stopping**, the reward is $r = s$, where s is the current state. If **drawing**, the reward is $r = lr$; also call the `draw()` function so that you can compute the successor state s'.
- Update the appropriate Q value, as well as `record` with the current state, action, and reward. You should use 0 for the "Q value" of the "done" state.
- Reset s to 0 if we either took the **stop** action or $s' > 21$, else set $s = s'$.

After finishing N transitions, return `Q` and `record`.

5.5: RL Analysis (9 points)

After you implement `Qlearn`, you should ensure that it is working correctly by comparing the learned state values (the maximum values in each row of `Q`) with those returned by value iteration. A suitable default set of learning parameters would be to have a low α (e.g., 0.1), low ε (e.g., 0.1) and high N (e.g., 50000).

Once you are ready, call the provided `RL_analysis` function (no additional code needed) and answer the following questions using the plots that you see. Your `Qlearn` **must** return the two arrays as specified above in order for this to work properly.

(a) The first plot shows the number of times each state is visited when running `Qlearn` with $lr = 0$, $\gamma = 1$, $\alpha = \epsilon = 0.1$, and N ranging from 0 to 50k in 10k intervals. Explain how the value of N is important to ensuring that all states are visited sufficiently. You should also note that the two most visited states are the same regardless of N; explain why these two states attract so much attention.

(b) The second plot shows the cumulative reward received for a game using the same parameters above, except with $N = 10000$ and ε ranging from 0 to 1 in 0.2 intervals. Explain what you see with the $\varepsilon = 0$ curve and why we need exploration to learn. Looking at the other curves, how does increasing exploration affect the overall rewards received and why?

(c) The third plot shows the estimated state values using the same parameters above, except with $\varepsilon = 0.1$ and α ranging from 0.1 to 1. While the set of curves may be hard to read, each should bear some resemblance to the true state values V^*. What is a problem that arises when α is too low, particularly with less visited states? What is a problem that arises when α is too high? Try to compare the stability or smoothness of the different curves.

Submission

You should have one PDF document containing your solutions for problems 1-4, as well as the information and plots for 5.3 and 5.5. You should also have a completed Python file implementing problem 5; make sure that all provided function headers are unchanged. **Please save and upload your code as a `.py`, not `.ipynb`, file.** Submit the document and code file to the respective assignment bins on Gradescope. For full credit, you must tag your pages for each given problem on the former.