Introduction to Machine Learning (IML)
BPINFOR-113
Take-Home Assignment 2 (THA2)
Grading scheme: 20 points (0-20)
Weight for final grade: 10%

Responsible: Decebal Mocanu, Christopher Gadzinski, Patrick Keller

Release date: 3 November 2023.
Deadline: 20 November 2023, 23 : 59.

Preamble

Please submit your solutions (the developed code and a report in PDF format) through Moodle. State clearly in the report the group number and the group members.

The solutions for the Take-Home Assignment 2 can be submitted just by a group. Individual submissions are not allowed. Each group shall have three members.

For each task (e.g. A.2) or subtask (e.g. A.1.a), create a corresponding section in your written report. Make sure to provide an answer to all questions that are asked within a task and/or indicate where the implementation is to be found. When you are asked to plot something, always provide the plot in the report and explain what it shows.

For each code that needs to be implemented, if required, create a separate stand-alone version/file, do not mix multiple task solutions into the same function or code block.

A. Exploration-Exploitation dilemma (10 points)

Problem

The goal of this exercise is to gain insights into how to solve the exploration vs. exploitation dilemma by experimenting with the multi-armed bandit problem.
We consider in the following 10 slot machines (i.e., a 10-armed bandit). Each of the slot machines possesses a fixed winning probability that we do not know (it is accessible from the code but must not be used to implement a strategy!) and will have to be estimated by repeated trials. You are provided with a code template *(ex_A_template.py)* that contains most of what you will need to get started. Adjust the code as you see fit but make sure to maintain the behavior of the slot machines.

We call one experiment a set of 1000 trials (actions) (nb_trials), each trial consisting in selecting a machine, inserting a coin, pulling its arm and collecting the winnings if any. The aim is to maximize the average winning over the 1000 trials by learning which machine is the most profitable.

It is recommended to address the different Tasks in order as they may depend on the previous results.

Tasks

1. **Understanding the template (1.5 points)**

(a) The value returned by *get_reward()* is the winning in euros obtained at a certain trial. What are the possible winnings of a single trial? Give a mathematically precise definition of the image of the *get_reward()* function.
(Hint: "the image of a function is the set of all output values it may produce")

(b) What is the maximum amount of money (BestPossibleWinnings) that can be won in a single trial?

(c) What information does the plot generated by the template show?

(d) Describe the default playing strategy which is implemented in the template?

2. **Repeatability of experiments (1.0 points)**

(a) Is the result of a single experiment representative? Why, why not?
Hint: Run the unmodified template script multiple times to verify your claim!

(b) By default, the implementation template allows to only run a single experiment. Extend the code to run multiple experiments by encapsulating the code for a single experiment into a function called *run_experiment*. Your function should return the relevant experiment data to gather, compute and display statistics about multiple experiments and draw a plot.
Hint: make sure that the machine's winning chances remain constant over multiple experiment runs.

3. **Performance evaluation of the default strategy (2.5 points)**

(a) Plot (using code) the evolution of the average winnings (runningMean) over the trials of an experiment! Do you observe that the average performance is improving over an experiment (i.e. is the algorithm learning to improve its strategy)? Explain why with the help of the graph!

(b) Calculate and plot (using code) the average ratio AverageWinnings/BestPossibleWinnings over at least 10 or more experiments (more is better, choose as many as possible if your PC can run it in reasonable time).
Is this strategy stable over multiple experiments (i.e. does it reliably return the same winnings)? Explain why with the help of the graph!

(c) Plot the winnings per machine as a series of box plots or violin plots in a single figure (see lecture slides). Can you deduce which machine has the highest win chances from the figure? Motivate your answer.

4. **Epsilon-Greedy strategy (4.0 points)**

(a) Determine the best arm so far from history. To do so, complete the code in function *get_best_arm()* after line 23 (step1 and step2).
It is supposed to look at the history of rounds played so far (action-reward) and return which was the best action on average so far. Note that this function is currently not used and is required for implementing the epsilon-greedy strategy. (Hint: it is recommended to use *numpy.where()* and *numpy.mean())*

(b) Implement the epsilon-greedy strategy (replace line 62-64 from the original template) as presented during the lecture and add the epsilon as a parameter to the *run_experiment* function!
Hint: Epsilon can be used to run the initially implemented strategy by setting it to a specific value.

(c) Plot (using code) the evolution of the average winnings (runningMean) over the trials of an experiment! Do you observe that the average performance is improving over an experiment (i.e. is the algorithm learning to improve its strategy)? Explain why with the help of the graph!

(d) Calculate and plot (using code) the average ratio AverageWinnings/BestPossibleWinnings over at least 10 or more experiments. BestPossibleWinnings is the amount we could win if we would get extremely lucky and would win the maximum at each trial.
Is this strategy stable over multiple experiments? Does it work better than the standard policy that was provided by the template? Explain why with the help of the graph(s)!

5. **Advanced strategy - softmax (1.0 points)**
   Read up on the softmax strategy. Explain in your own words how it works. Describe how it differs from the epsilon-greedy strategy and what advantage it has.

B. Reinforcement Learning (10 points)

Problem

The goal of this exercise is to apply reinforcement learning techniques to teach an agent how to play a game in the best possible manner. The game considered, provided as part of the gymnasium platform, is described at `https://gymnasium.farama.org/environments/toy_text/cliff_walking/`.
In that game, we assume that we do not know beforehand neither the possible states nor the rewards but we do know there are four actions that can be taken at each step. We assume the agent is blind and only knows whether he stayed on the hill, fell off the cliff or reached the goal after taking an action. The aim is to find a policy that maximizes the cumulated reward obtained when playing the game.

In the following, an experiment is 500 episodes, each episode consists of a single game of max. 1000 steps (i.e., 1000 actions taken by the agent at each episode). If the agent reaches the goal or falls off the cliff we consider the game (and thus the episode) to be over and a new game(episode) starts.

You are provided with a template *(ex_B_template.py)* that shows you how to use the gymnasium environment and provides a sample implementation of a dumb agent that only moves in circles. To run the template, first install the "gymnasium" package as described at the beginning of the course.

The template also allows to display the game field graphically after every action. To enable it, install the pygame package and set the "RENDER_MODE" variable to "graphic". By default, the game field is rendered as text in the console.

Tasks

- **Understanding the problem (1.5 points)**

  Read the description of the gymnasium "cliff walking" environment at: `https://gymnasium.farama.org/environments/toy_text/cliff_walking/`.

  1. Describe in your own words what the game is about and what are the relevant components (actions, states, goal, traps, etc.)
  2. Describe what the perfect policy would be to solve this game optimally.
  3. What is the best (highest) and worst (lowest) reward we can achieve in this game and why?

- **Baseline strategy: Equiprobable Random Policy (1.5 points)**

  1. Implement a new agent class that uses the ERP. (Hint: you may use the provided DumbAgent class as a template)
  2. Evaluate the ERP agent by running a single experiment of 500 episodes. Plot the reward per episode as a scatter plot. What can you conclude about the effectiveness of this policy from the plot?
  3. Report the average reward that the agent achieves per game, how many times the agent reached the goal and how often the optimal reward was achieved during the experiment.

- **Q-Learning (5.0 points)**

  1. Implement a new agent that uses the Q-Learning algorithm as it was presented during the lecture. Implement the Q-Learning algorithm from scratch, applying the epsilon-greedy policy. Add arguments to the agents constructor to provide the alpha and epsilson values at initialization. If in a given state there are multiple best actions, choose one at random to favor exploration.

2. Evaluate your implementation of Q-Learning with alpha=0.15 and epsilon=0.15 by running 100 full experiments of 500 episodes. For gamma, you can apply 0.9. Plot the average reward per experiment and the best reward per experiment as a scatter plot.
3. Report the average reward that the agent achieves overall, how many times the agent reached the goal and how often the optimal reward was achieved during the experiment.

- **Improving Q-Learning (2.0 points)** Find the best values for alpha and epsilon in the range [0.0, 1.0] (for instance with a granularity of 0.05). Optimally, run 10 or more experiments per (alpha,epsilon)-pair to get stable results. To compare which value pair is better, determine a reasonable metric and explain why you chose it.

Success!