ASEN/CSCI 5264 Decision Making under Uncertainty
Homework 5: Introduction to POMDPs and Advanced RL

March 9, 2024

1 Exercises

Question 1. (25 pts) Consider the following POMDP that represents cancer monitoring and treatment plan[1]:

$$\begin{aligned}
\mathcal{S} &= \{\texttt{healthy}, \texttt{in-situ-cancer}, \texttt{invasive-cancer}, \texttt{death}\} \\
\mathcal{A} &= \{\texttt{wait}, \texttt{test}, \texttt{treat}\} \quad \mathcal{O} = \{\texttt{positive}, \texttt{negative}\} \\
\gamma &= 0.99 \quad s_0 = \texttt{healthy}
\end{aligned}$$

The **transition dynamics** are designated with the following table. The state stays the same except with the probabilities encoded in the table.

s	a	s': $\mathcal{T}(s' \mid s, a)$
`healthy`	all	`in-situ-cancer`: 2%
`in-situ-cancer`	`treat`	`healthy`: 60%
`in-situ-cancer`	$\neq$ `treat`	`invasive-cancer`: 10%
`invasive-cancer`	`treat`	`healthy`: 20%; `death`: 20%
`invasive-cancer`	$\neq$ `treat`	`death`: 60%

The **observation** is generated according to the following table. The observation is `negative` except with the probabilities encoded in the table.

a	s'	o: $\mathcal{Z}(o \mid a, s')$
`test`	`healthy`	`positive`: 5%
`test`	`in-situ-cancer`	`positive`: 80%
`test`	`invasive-cancer`	`positive`: 100%
`treat`	`in-situ-cancer` or `invasive-cancer`	`positive`: 100%

The **rewards** are defined as follows (one could interpret the reward as roughly quality years of life):

- $R(\texttt{death}, \text{any action}) = 0.0$ (i.e. `death` is a terminal state)
- $R(\text{any living state}, \texttt{wait}) = 1.0$
- $R(\text{any living state}, \texttt{test}) = 0.8$ (because of costs and anxiety about a positive result)
- $R(\text{any living state}, \texttt{treat}) = 0.1$

Create a model of this problem using `QuickPOMDPs` and use Monte Carlo simulations to evaluate a policy that always **wait**s (we will solve this problem in the next homework).

[1]Note that the probabilities are not meant to be realistic. See `https://pubsonline.informs.org/doi/10.1287/opre.1110.1019` for an actual publication on this topic

Question 2. (25 pts) Using the deep learning library of your choice (e.g. Flux.jl, Knet.jl, Tensorflow, PyTorch), fit a neural network to approximate the function $f(x) = (1 - x)\sin(20\log(x + 0.2))$ for the range $x \in [0, 1]$. Plot a set of 100 data points fed through the trained model and plot the learning curve (loss vs number of training epochs).

2 Challenge Problem

Question 3. (50 pts) In this exercise, you will learn a policy for the special mountain car environment, `DMUStudent.HW5.mc`. This environment is a standard mountain car with $\mathcal{A} = [-1, 1]$, except that the observation is a 100x100 matrix. The first two entries in the matrix contain the car's position and velocity. The rest of the matrix may contain useful information to receive additional rewards, but it is not needed to achieve full credit.

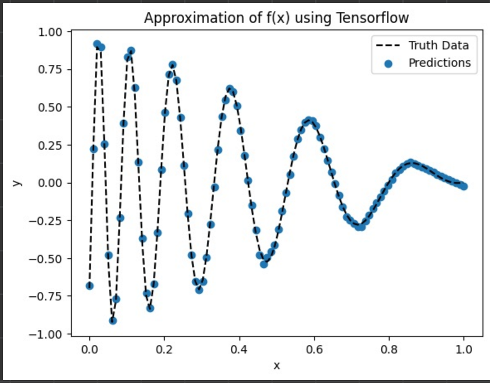
a) Implement a reinforcement learning algorithm to learn a policy for the environment and plot a learning curve. Write a paragraph describing the algorithm you implemented.[2]

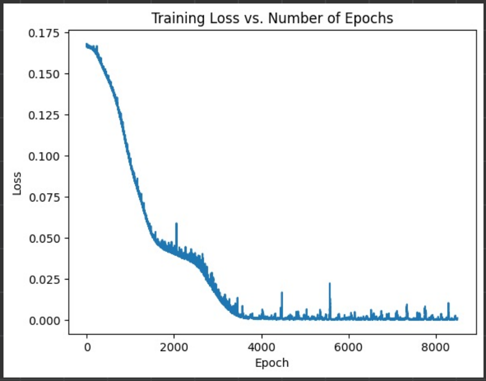
b) Evaluate a policy with `DMUStudent.HW5.evaluate`, and submit the resulting json file. You may use your code from part (a) or *any* other libraries for this part. A discount factor of $\gamma = 0.99$ is used for evaluation. A score of 40 or greater will receive full credit. Your submission should be a function that takes in a state and returns an action.

[2] I recommend discretizing the action space and implementing the DQN algorithm with only the position and velocity - see the starter code for instructions on how to create an environment wrapper to do this; this is the only algorithm that I can provide full debugging support for. DQN should be able to learn a policy that can achieve a return of in the 40s with a discount factor of $\gamma = 0.99$ in less than 10 minutes of training time.

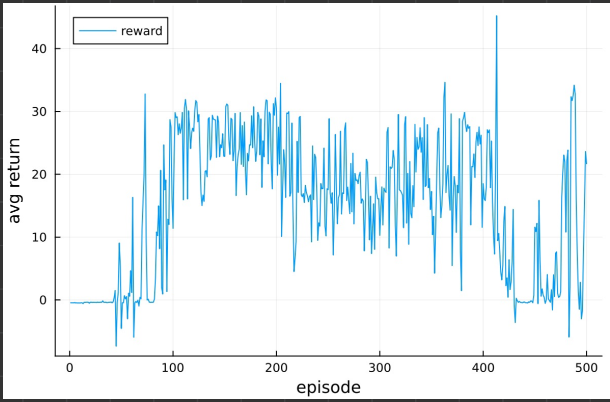
1) Score = 38.1690

2)





3) a) In order to solve the mountain car problem I used Deep Q-Network (DQN). The base of DQN was not sufficient to solving the problem, and required some modification and hyperparameter tuning. The three modifications I employed were the network structure, buffer and target freezing. For the tuning of hyperparameters, I made my epsilon decay from 0.5 to 0.05, I ran 500 episodes each with a maximum of 400 steps, I froze Q every 100 steps, and every 100 steps I trained the nueral network 100 times with 20 data points from my buffer.



b) With these parameters, I was able to receive a final score of 41.0972.