CS5814 Assignment 4

Reinforcement Learning

April 29, 2022

Sam Donald

1. Problem 1 [40 points]:

Problem 1 involves using both value and policy iteration to solve the OpenAI Frozen Lake environment. The environment emulates an ice lake with holes and is displayed within figure 1. This environment can be configured to be both a deterministic and stochastic, with the stochastic mode allowing movement in the desired direction with probability 1/3, and 1/3 in each perpendicular direction. The available actions are left, right, up and down.

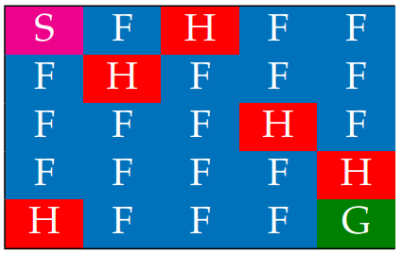

Figure 1: Frozen Lake 5x5 configuration used, with S = start, F = free, H = hole and G = goal

1.1 Questions:

a) Install OpenAI Gym and set up the problem using the given code.

This is done within the associated .ipbyn notebook.

b) Assume is_slippery=False. Answer the following questions with proper justification. Consider cases 0 < γ < 1 and γ = 1 for (i), (ii), and (iii).

i. Does this MDP have a unique optimal value function?

Yes, the value of any given state will converge to a value of γ^{k-1}, where k is the minimum number of steps required to reach the goal for the +1 reward from the current state. This is because the optimal value function must satisfy the Bellman equation for state values outlined below, where for the deterministic environment $P^a_{s,s'} = 1$ and $R^a_{s,s'} = 1$ when s'=goal, otherwise $R^a_{s,s'} = 0$. The value of γ therefore does not impact the uniqueness of the value function.

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} P^a_{s,s'} \left[R^a_{s,s'} + \gamma V^*(s') \right]$$

ii. Does this MDP have a unique optimal policy?

0 < γ < 1: While there exists a unique value function for both 0 < γ < 1 and γ = 1, for the given Frozen Lake environment there are states where adjacent accessible states s'_1 and s'_2 of equal value. As the optimal policy is gained by acting greedily with respect to the optimal value function, this will result in multiple optimal actions and therefore multiple optimal policies. An example of two policies derived from identical value functions are displayed within figure 2. Green circles are used to identify a pair of states with equal values, as they both require a minimum of 5 steps to reach the goal. Note that only the actions for the path taken within this deterministic environment are displayed.

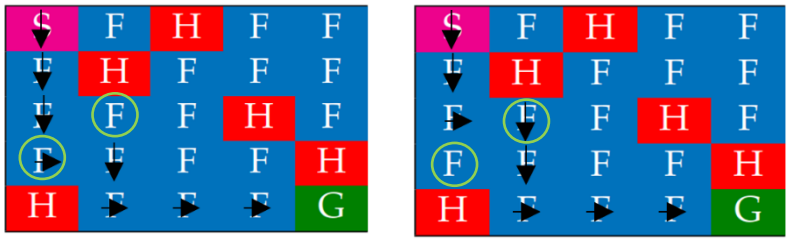

Figure 2: Example of non-unique optimal policies.

γ = 1: As the value function is equal to 1 for every non goal or hole (G and H), acting greedily with respect to the value function will result in multiple policies due to adjacent accessible states having the same value. Note that some of these deterministic optimal policies will not reach the goal state as an agent can get stuck in an infinite loop as displayed within figure 3. Additionally, as there is no penalty for taking a longer path to the goal within this problem, using γ = 1 may not result in a policy that takes the minimum number of steps displayed within figure 3. For these reasons, values of γ below 1 are typically used, or a reward of -1 is applied for every step the agent takes within the environment to incentivize ending the episode.

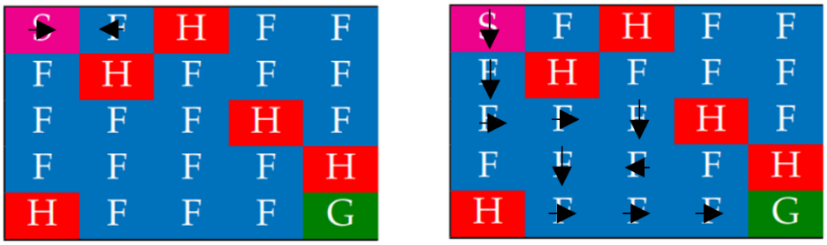

Figure 2: Example of infinite loop (left) and long policies (right) based on γ = 1 value function.

iii. Does this MDP have a deterministic optimal policy?

Yes, given that the environment is both deterministic and its dynamics are fully known and accessible the optimal policy, which is gained from acting greedily with respect to the optimal value function, will be deterministic. Note that there are multiple deterministic optimal policies as outlined within question ii.

iv. Considering that γ < 1, does the value of γ affect the length of the walkable path given by an optimal policy?

Given 0 < γ < 1, the length of the walkable path for any given optimal policy will be the same length. The value of any given state will converge to a value of γ^{k-1}, where k is the minimum number of steps needed to move to the goal state, as described by the Bellman optimality equation within part i. Therefore, by acting greedily with respect to the value function the value maximizing action will always be chosen. This reduces the minimum number of steps to the goal and ensures the length of the walkable path is constant.

c) Implement Value Iteration (VI) algorithm and run it until convergence. Report the optimal values of each state obtained at convergence.

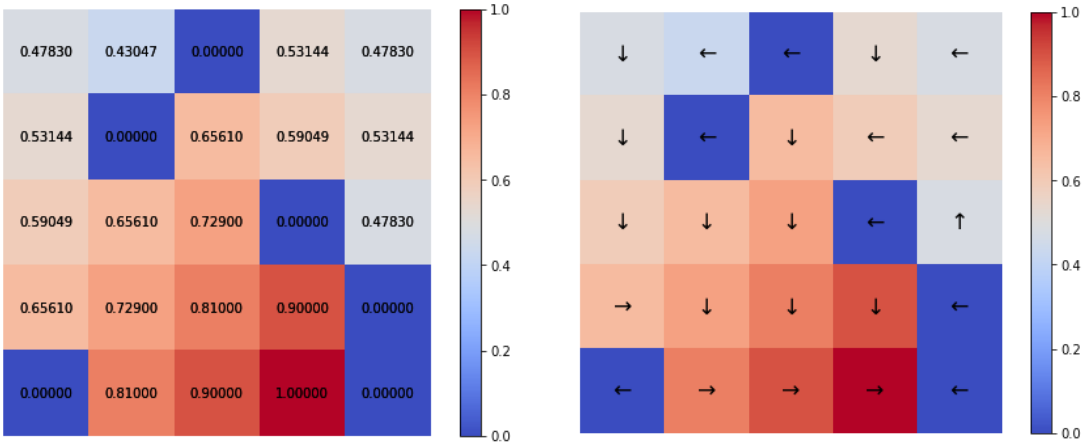

Figure 3: Optimal values (left) and policy (right) from Value Iteration algorithm at convergence.

d) Implement Policy Iteration (PI) algorithm and determine the optimal policies. Report the values (at each state) obtained at the optimal policy (i.e., upon convergence). Compare with the values obtained using VI method. Are they the same?

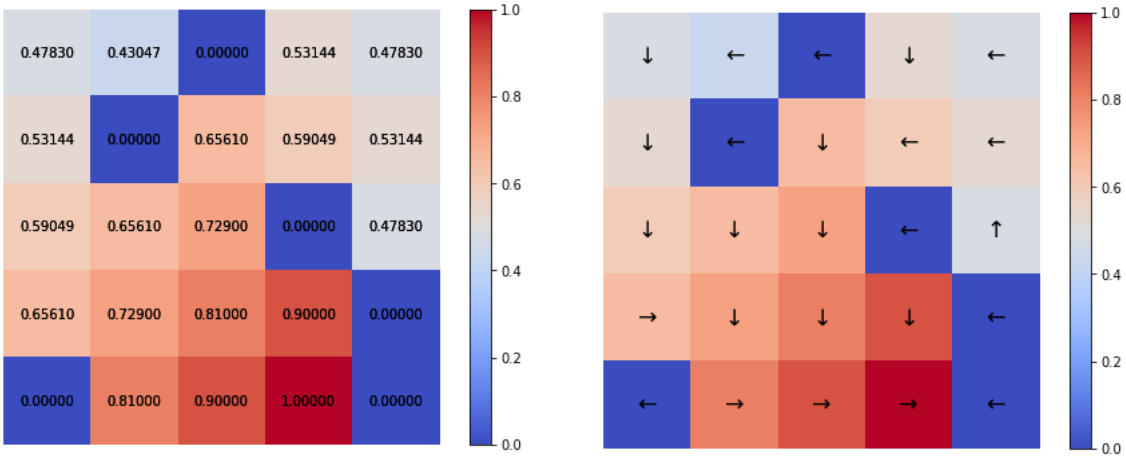

Figure 3: Optimal values (left) and policy (right) from Policy Iteration algorithm at convergence.

The values of the optimal value functions are the same for policy and value iteration as expected. The associated polices are the same as the same deterministic tie breaking code was used for actions of equal value.

e) Reduce the discount factor γ to 0.6 and run the VI algorithm. Report the number of iterations it takes for convergence. Also, report the new values for each state upon convergence.

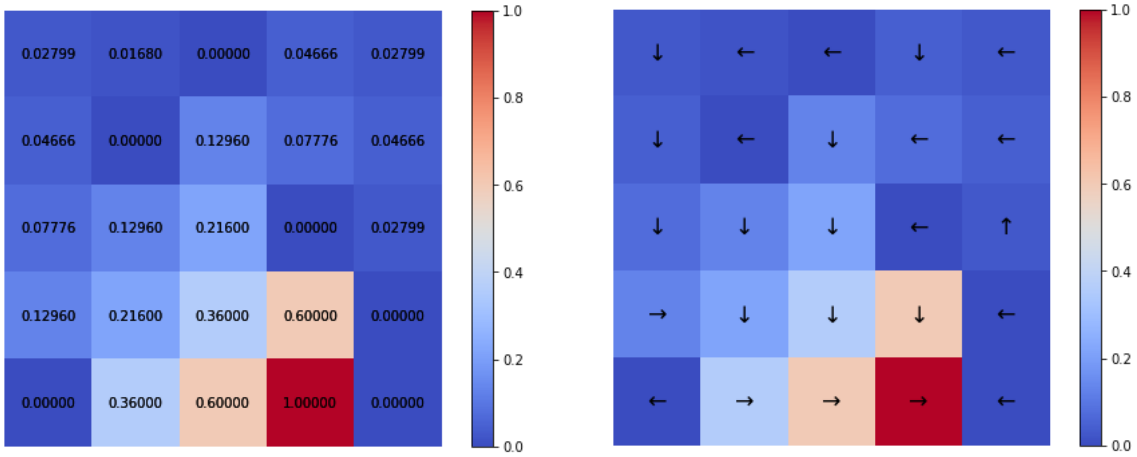

Figure 4: Optimal values (left) and policy (right) from Value Iteration algorithm at convergence with γ = 0.6.

The algorithm converged within 9 steps for γ = 0.6, this is the same number of steps as γ = 0.9.

f) **Report the new values with VI and PI and show if the optimal policy changes (when using the stochastic version with is_slippery = True).**

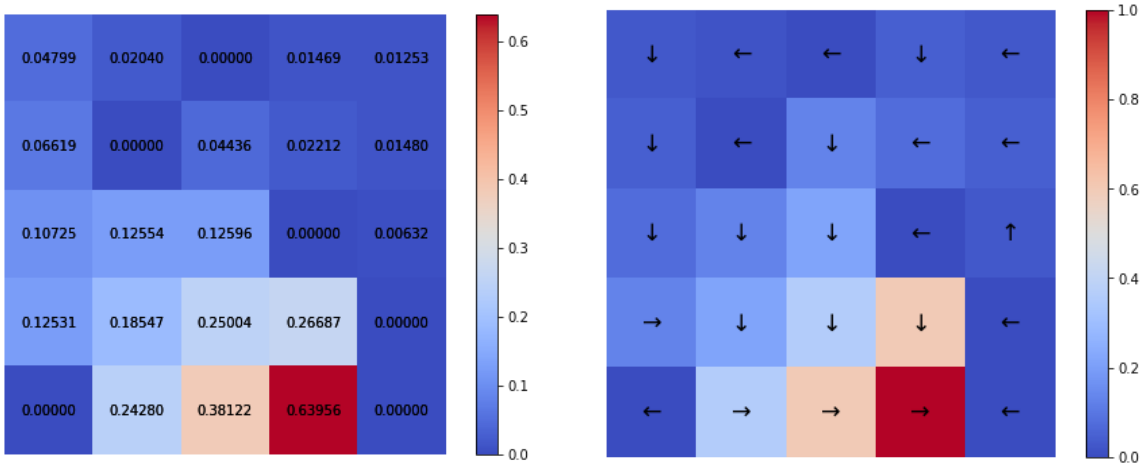

Figure 5: Values (left) and policy (right) from Value Iteration algorithm at convergence with stochastic environment.

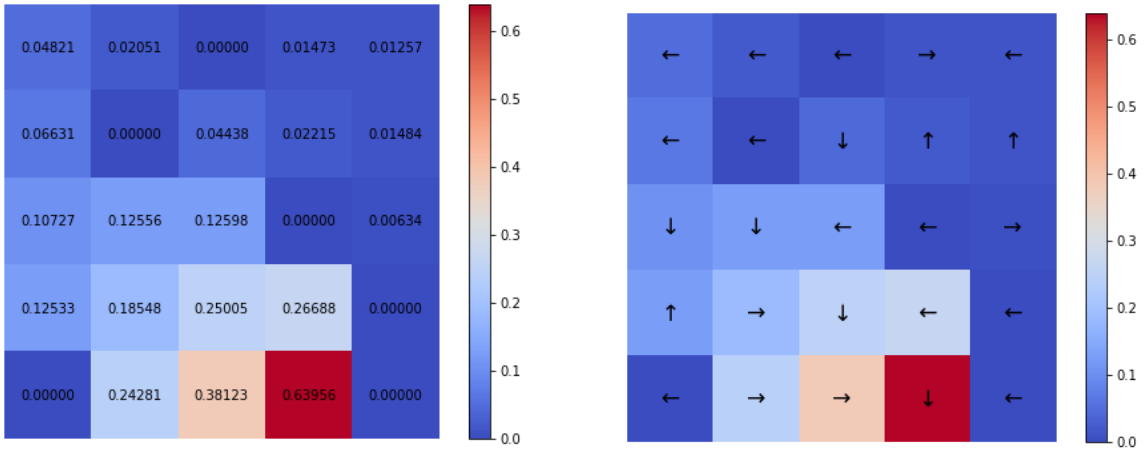

Figure 6: Values (left) and policy (right) from Policy Iteration algorithm at convergence with stochastic environment.

As expected, the policies significantly change between the stochastic and deterministic environments, as the agent is now unable to reliable move to its desired destination. The values and associated policies displayed in figures 5 and 6 are also slightly different due to this stochasticity.

Once converged, the value iteration policy was able to reach the goal with a success rate of 53.90%, while the policy iteration policy was able to achieve a slightly higher rate of 57.30%.

2. Problem 2 [30 points]:

Problem 2 involves the same Frozen Lake environment as per problem 1 yet is solved through Q-learning. The deterministic version of the environment is used here.

2.1 Questions:

a) Solve the problem with Q-learning, using γ = 0.6 and α = 0.5, until it has converged based on the Q-values don't change by more than 10−4 for 2000 updates. Report the optimal values obtained from Q-learning at each state. What policy did you use to sample (s,a,s',r) tuples?

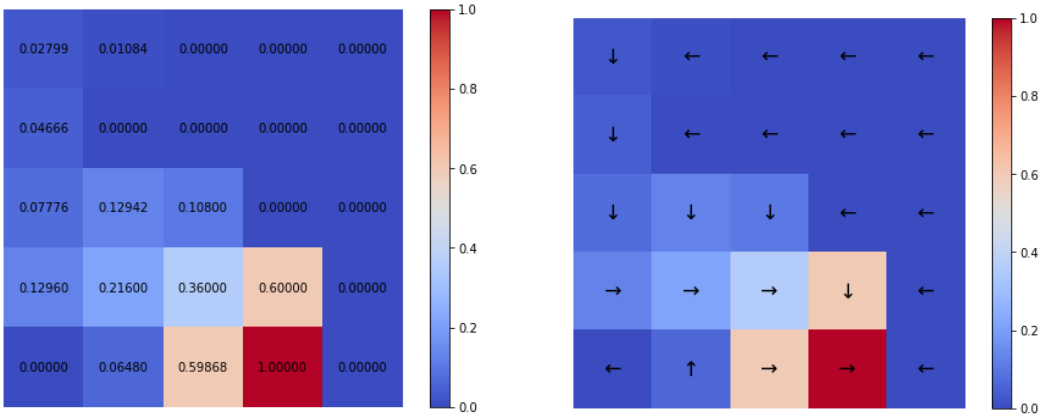

Figure 7: Values (left) and policy (right) from Q-learning algorithm at convergence.

The policy used to sample (s, a, s', r) tuples was an epsilon greedy one, based on a decaying epsilon as displayed in figure 8. With this policy the model was able to converge within 733 episodes, or 5769 updates. Increasing the value of epsilon would allow the agent to explore more and perform significantly better on the stochastic version of the environment.

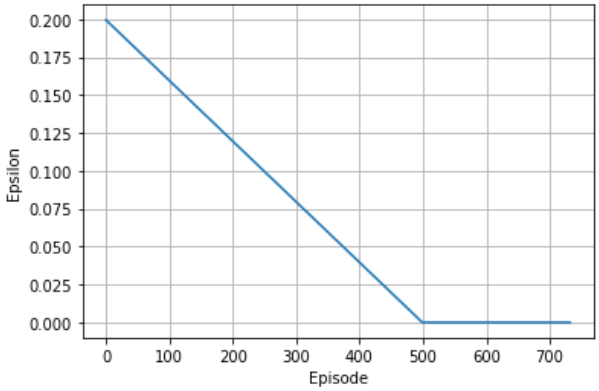

Figure 8: Epsilon decay used for epsilon greedy policy.

b) Should the policy used to sample (s,a,s',r) tuples for Q-learning necessarily be based on the Q-values?

An epsilon greedy policy in its extreme, with epsilon = 1, would be sampling tuples randomly and therefore not based on Q-values. This is a viable method to explore an environment early on during training yet must be accompanied by an eventual decrease in epsilon such that learning of Q values can occur.

3. Problem 3 [30 points]:

This problem was first described by Sutton in 1996. From Sutton96, "the acrobot is a two-link underactuated robot displayed within figure 9 roughly analogous to a gymnast swinging on a highbar." The two links are initially hanging downwards and the goal is to swing the free end above a certain height by applying torques on the actuated joint. The state is a 6-length vector containing cosine and sine of the two angles θ1, θ2 and the two angular velocities $\dot{\theta}_1$, $\dot{\theta}_2$. The possible actions include positive torque, negative torque, and no torque. The reward is -1 for each step, so we want to reach the goal in as few steps as possible.

This Acrobot problem is solved with deep Q-learning (DQN).

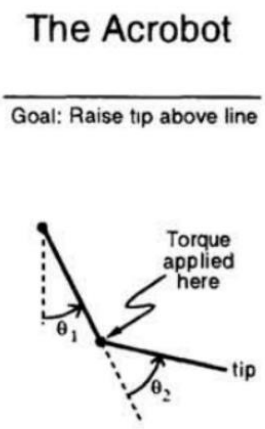

Figure 9: The Acrobot

3.1 Questions:

a) Install OpenAI Gym and build the Acrobot environment.

This is done within the associated .ipbyn notebook.

b) Write an expression for the return for an episode in terms of the discount factor γ and episode length L.

For a given trajectory, the total reward does not depend on the discount factor.

$$R_{episode} = \sum_{t=0}^{t=L} R_t$$

However, the expected discounted return of future rewards for a given timestep t, can be modeled by the following equation.

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots = \sum_{k=0}^{L-t} \gamma^k R_{t+k}$$

Where R_t is the expected reward at timestep t. When considering the entire trajectory, t is equal to 0 and the above expression simplifies to.

$$G_0 = \sum_{k=0}^{L} \gamma^k R_k$$

c) Solve the Acrobot problem with deep Q-learning (DQN). Plot episode length vs training steps and the moving average with a window size of 10. Once trained run 100 more episodes and report the average no. of steps needed to reach the goal.

Figure 10 below displays the converging episode length during the training process. Once trained the model was able to solve the environment within an average of 109.64 steps, averaged over 100 episodes. The epsilon decay function corresponds to the function $\varepsilon = \max(0.99^{episode}, 0.01)$ and is displayed within figure 11, while hyperparameters used are outlined within table 1. The architecture used is a simple FNN with one hidden layer, using a ReLU activation function.

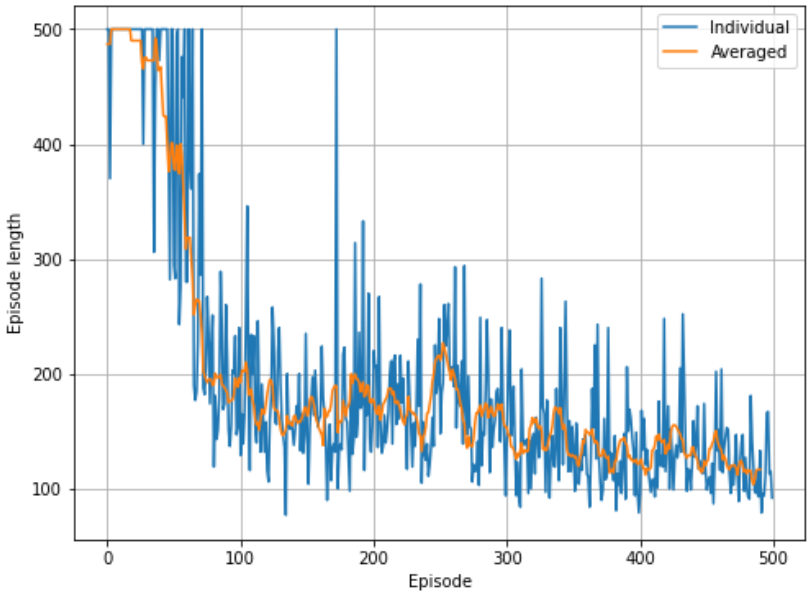

Figure 10: Episode length throughout training process

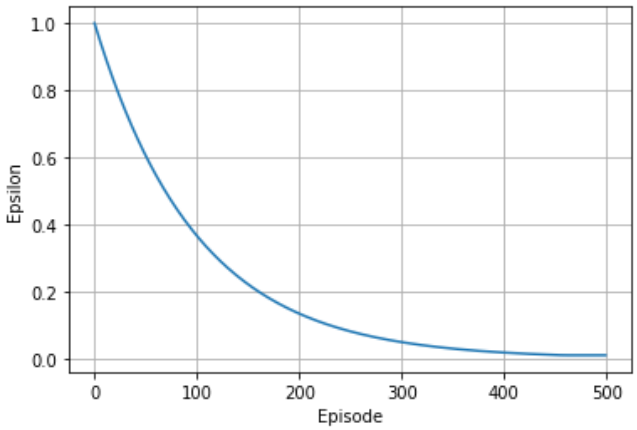

Figure 11: Epsilon decay used for epsilon greedy policy

Hyperparameter	Value
Hidden layer width	30
Learning rate	0.001
Gamma	0.999
Max episodes	500
Batch size	128
Target network update	Every 2 episodes

Table 1: Hyperparameters for DQN

An experiment conducted during the tuning process is displayed below. Notably the epsilon decay function is linear which had significant impacts on the convergence shape in figure 12, which is also approximately linear. While the final model was able to solve the environment within 166.27 steps, the model was inconsistent between training runs, this was significantly improved through the usage of the decay function outlined in figure 11.

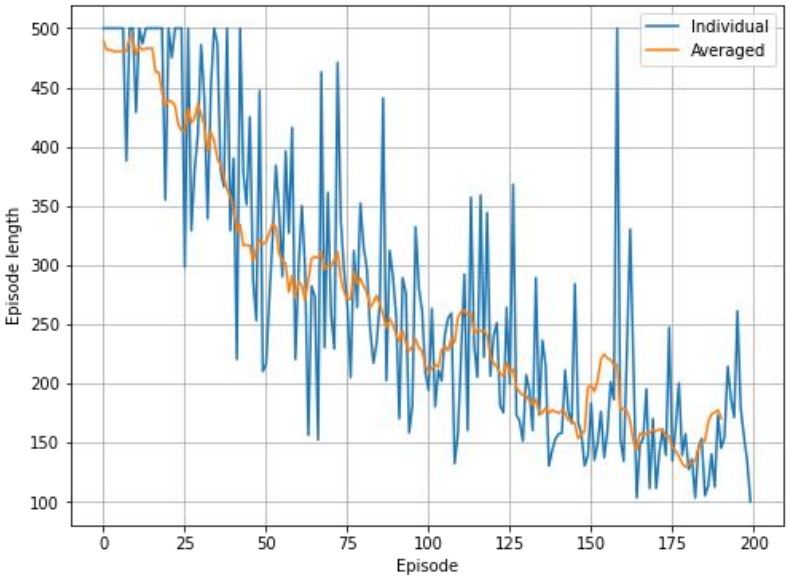

Figure 12: Episode length throughout training process (linear epsilon decay)

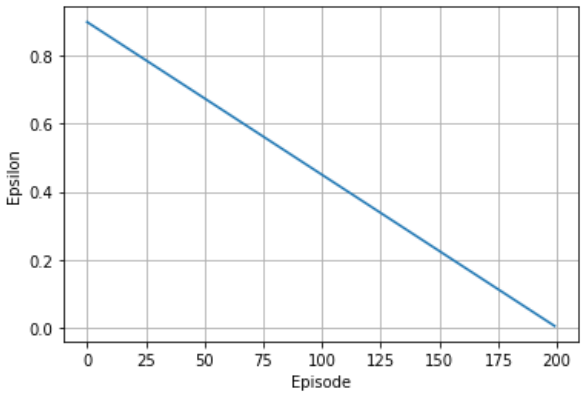

Figure 13: Linear epsilon decay used for epsilon greedy policy

Hyperparameter	Value
Hidden layer width	20
Learning rate	0.002
Gamma	0.999
Max episodes	200
Batch size	128
Target network update	Every 2 episodes

Table 2: Hyperparameters for DQN (linear epsilon decay)

4. Problem 4 [20points]:

This problem uses the same Acrobot environment as problem 3, however is solved using the REINFORCE policy gradient algorithm.

4.1 Questions:

a) Use the REINFORCE algorithm to solve the same acrobot problem.

This is done within the associated .ipbyn notebook. The hyperparameters used for the model are outlined within table 3 below. The model architecture is the same as that used in problem 3, with the addition of a softmax layer to infer action probabilities.

Hyperparameter	Value
Hidden layer width	100
Learning rate	0.0001
Gamma	0.999
Max episodes	500
Batch size	1

Table 3: Hyperparameters for REINFORCE

b) Show the plot containing episode length and its moving average vs training steps as in the previous problem. Run 100 more episodes and report the average no. of steps needed to reach the goal.

Figure 14 below displays the converging episode length during the training process. Once trained the model was able to solve the environment within an average of 117.97 steps, averaged over 100 episodes. Training would likely improve with the implementation of batched learning, however using a batch size of 1 was sufficient for this problem.

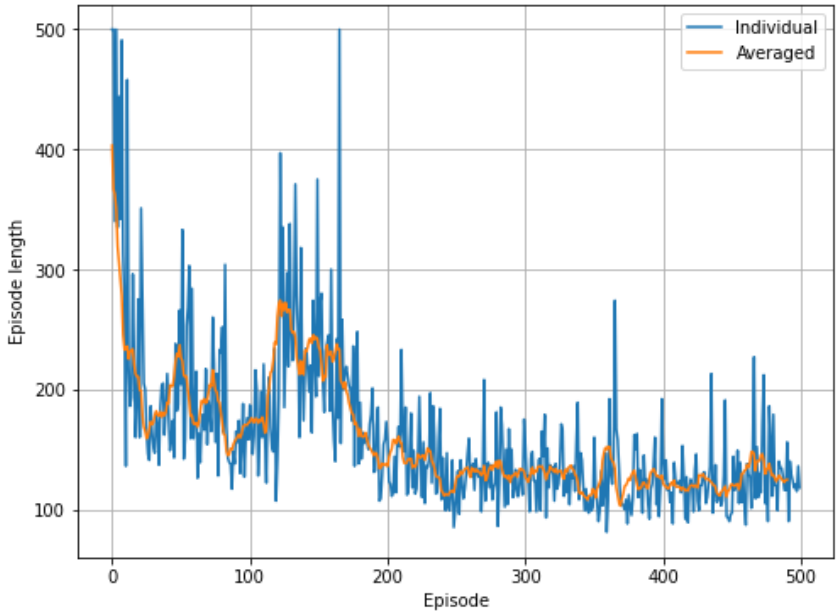

Figure 13: Episode length throughout training process for REINFORCE algorithm.