046203 – Programming Homework 3

Dar Arava - 205874951

Yair Nahum - 034462796

2 – Mountain Car Problem

1. The state space S consists all pairs of car position and car speed - $s \in S : s = \{position, speed\}$ Where-

$$position \in [\min\ \ position, \max\ \ position] = [-1.2, 0.6]$$
$$speed \in [-\max\ \ speed, \max\ \ speed] = [-0.07, 0.07]$$

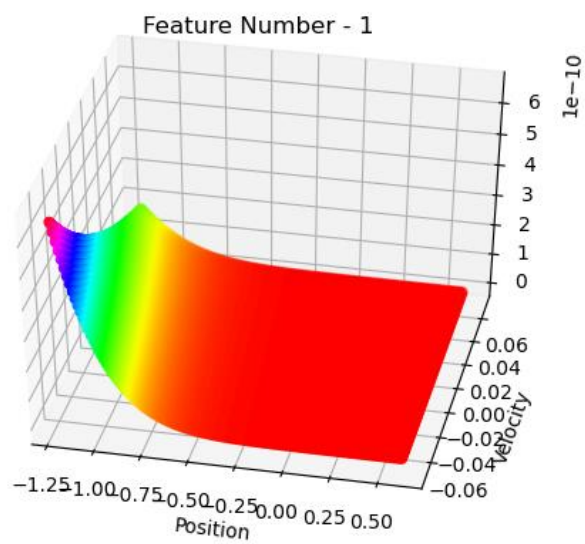
The action space A consists of three actions $A = \{0,1,2\}$ where $a \in A$ defines in which direction the car creates force (which effects the velocity of the car)-

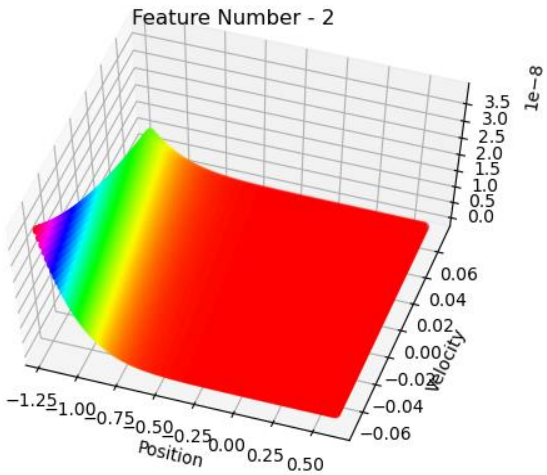
$$a = 0 - force\ applied\ to\ the\ right\ (to\ the\ target\ flag)$$
$$a = 1 - no\ force\ is\ applied\ (velocity\ is\ affected\ only\ by\ gravity)$$
$$a = 2 - force\ is\ applied\ to\ the\ left(\ opposite\ direction\ of\ the\ target\ state\ in\ our\ env)$$

The state-action reward is –

$$r(s,a) = \begin{cases} 1 & s(position) \geq goal\ position \\ 0 & else \end{cases} = \begin{cases} 1 & s(position) \geq 0.5 \\ 0 & else \end{cases}$$

Plot of 2 first features:





We can see that the 2 first features are actually being activated the most when the position is at the most left and velocity is the maximum to the left. This probably won't be activated on real experiments as we probably have almost 0 velocity on the left most position when we change the direction back to the target state (the flag).
Probably, the most activated RBF would be when we at the bottom of the hill with the maximum speed achieved (on both directions).

2. Encoding using RBF can give the following advantages:
    a. The RBFs can describe any none linear function as a superposition of its components. The θ as a vector is then used as linear combination of these RBFs to approximate any none linear function we wish. And we only need to solve a linear regression problem (instead of a none linear).
    b. We can reduce the states space to a different dimension, thus we can avoid the curse of dimensionality of states space (too big states' space).
    c. We may use the "kernel trick" on these RBF kernel functions in order to avoid complex or big calculations in the states space.

3 – LSPI

1. The data collected is with a random starting point reset for every sample s_t, a_t, r_t, s_{t+1}
   Thus, we don't need to stop when we reach a state that is already in terminal state. However, it's preferred to not include the future Q value estimation when reaching a done state. Meaning, not taking into account the $\gamma Q(s_{t+1}, a)$ when estimating (by LLN) the C matrix as this is a terminal state and there is no need for future rewards expectancy estimation for it (this might be another problem but not as described in our assignment).
   When removing this component from C estimation for terminal states, we reach 1.0 success rate.
   In our code the C estimation update is as follows:
   C += np.dot(phi, (phi.T - (**(1 - done_flags[i])** * gamma * phi_next.T)))
2. From program prints on mean and variance:
   data mean [-3.00931294e-01 7.02421111e-05]
   so the mean state is at -0.3 position and 7e-05 speed.
   data std [0.52 0.04]
   These measures fit the expectation from a uniform sampling over the states space.
3. The size of the weights vector w is as the size of the features vector $\phi(s, a)$. In our case the size of the features vector (as we estimate Q) is $|\phi(s, a)| = |\mathrm{A}||\phi(s)| = 3|\phi(s)|$
   In our case, when we used 12x10 features per state, we get $3|\phi(s)| = 3 \text{ x } 120 = 360$
   If there is a bias coefficient added to the features vector per state, we have:
   $$|\phi(s, a)| = |\mathrm{A}||\phi(s)| = 3|\phi(s)| = 3 \text{ x } 121 = 363$$

4. Will be implemented before and in the LSPI iterations loop to evaluate the success rate and build the average performance per iteration plot.

5. See implementation in code.
   In class we saw we need to do the following per iteration:
   Critic:
   a. Convert the state-action space to features space.
   b. By using the current w and features vector for the next state in each tuple of (state, action, reward, next state) estimate $Q(s', a)$ for every action and apply the greedy policy over it to find the next best action (current optimal estimated Q according to current policy).

c. Accumulate according to the below formulas the d vector and C matrix (From the lecture notes):

Batch - Least Squares Policy Iteration (LSPI)

One can also derive an approximate PI algorithm that works on a batch of data. Consider the linear case $\hat{Q}^{\mu}(s,a) = \theta^T \phi(s,a)$. The idea is to use LSTD(0) to iteratively fit $\hat{Q}^{\mu_k}$, where μ_k is the greedy policy w.r.t. $\hat{Q}^{\mu_{k-1}}$.

$$\hat{d}_n^k = \frac{1}{n}\sum_{t=1}^{n} \phi(s_t,a_t) r(s_t,a_t)$$

$$\hat{C}_n^k = \frac{1}{n}\sum_{t=1}^{n} \phi(s_t,a_t)(\phi^T(s_t,a_t) - \gamma\phi^T(s_{t+1},a_{t+1}^*)),$$

$$\theta_k = (\hat{C}_n^k)^{-1}\hat{d}_n^k.$$

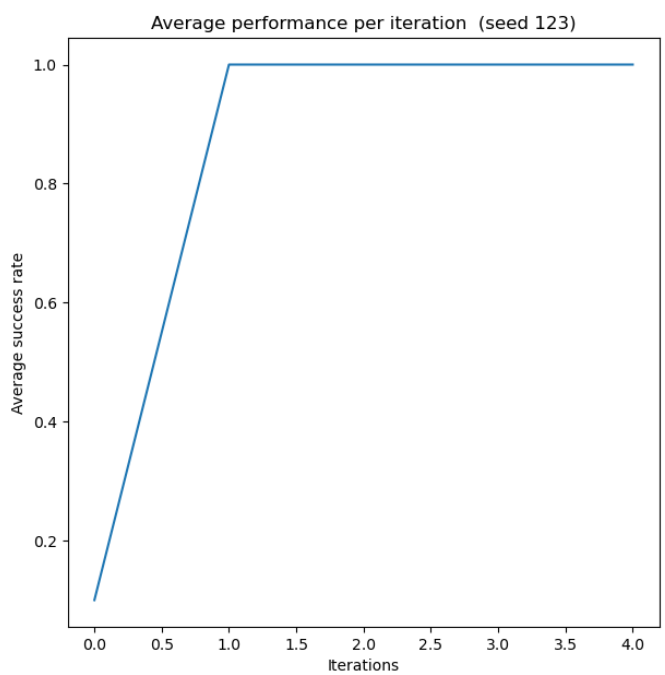
where $a_{t+1}^* = \arg\max_a \hat{Q}^{\mu_{k-1}}(s_{t+1},a) = \arg\max_a \theta_{k-1}^T \phi(s_{t+1},a)$.

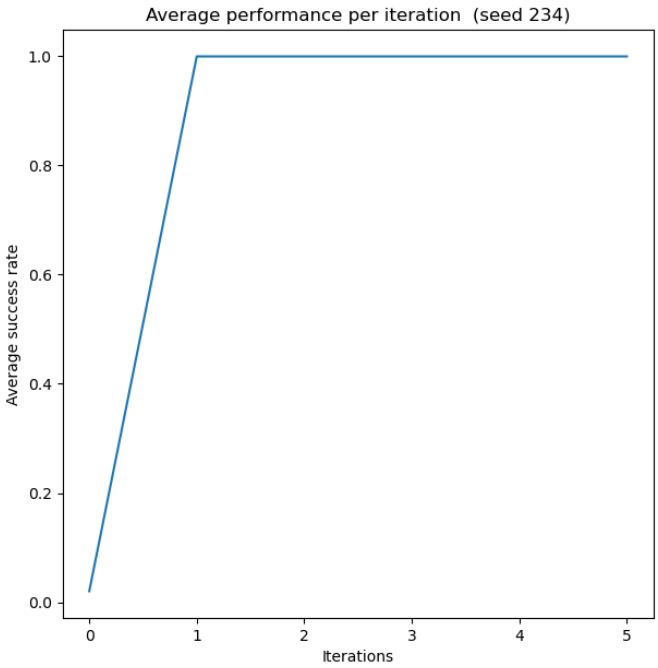
d. Apply inverse on matrix A and multiple with b vector to get the new estimation of w coefficients (new $Q(s,a)$ estimate)

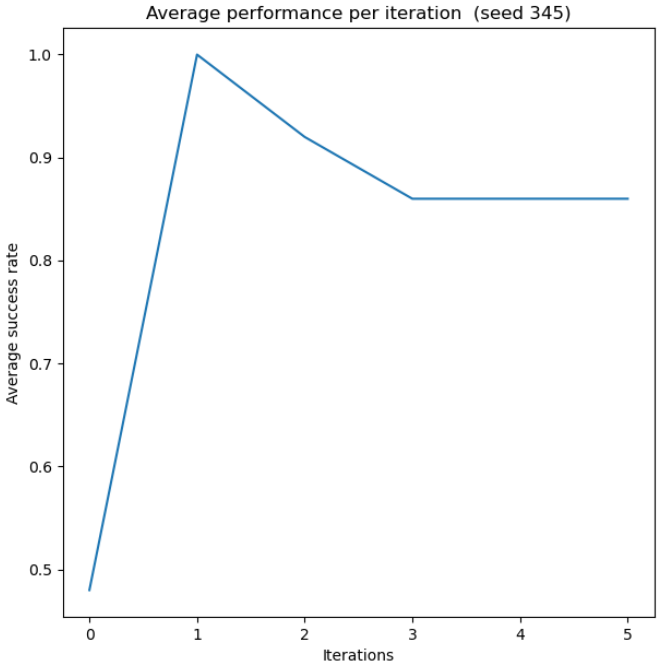
Actor:

e. Apply greedy bellman operator on the new $Q(s,a)$ estimation and get a new greedy (or maybe close to greedy) policy.

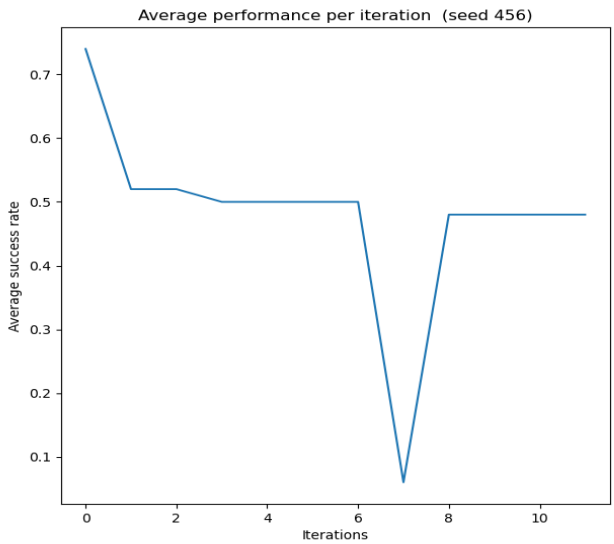
Plots of the average success rate over 50 random start states (w/ velocity 0) as described in section 4 of the question. The average success rate is evaluated vs the lspi iterations (until convergence). We tested several random seeds:



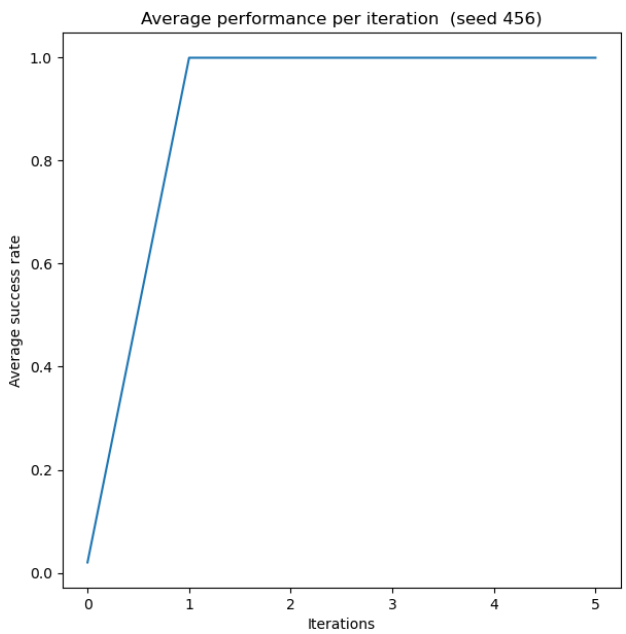

Average performance per iteration (seed 234)
Average success rate
Iterations
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5


Average performance per iteration (seed 345)
Average success rate
Iterations
0.5
0.6
0.7
0.8
0.9
1.0
0
1
2
3
4
5

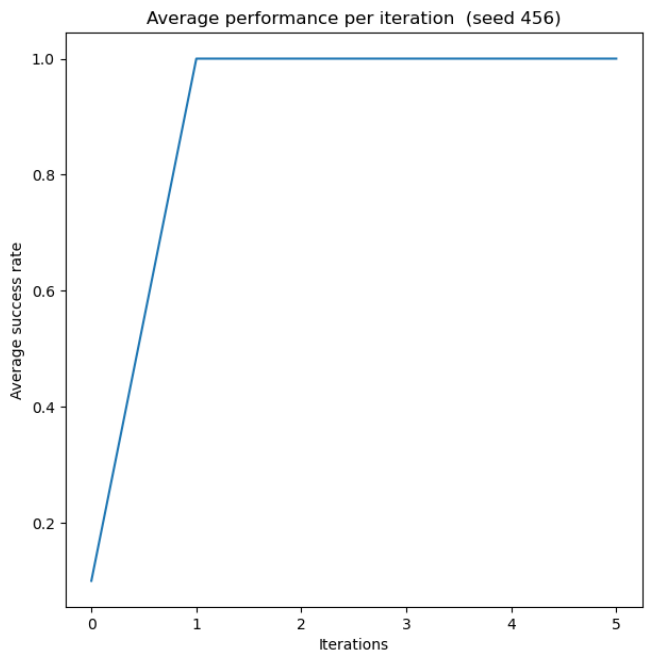
6. We collected the final greedy policy over final Q and measured the average success rate vs the amount of data samples collected for training. The plots per seed 456:
   For 10000 samples it seems to converge (less than 20 iterations) but the success rate is very low. We have a bias from the optimal policy and lack of data to train over. Thus, the success rate is very low.



For 100000 samples we get:

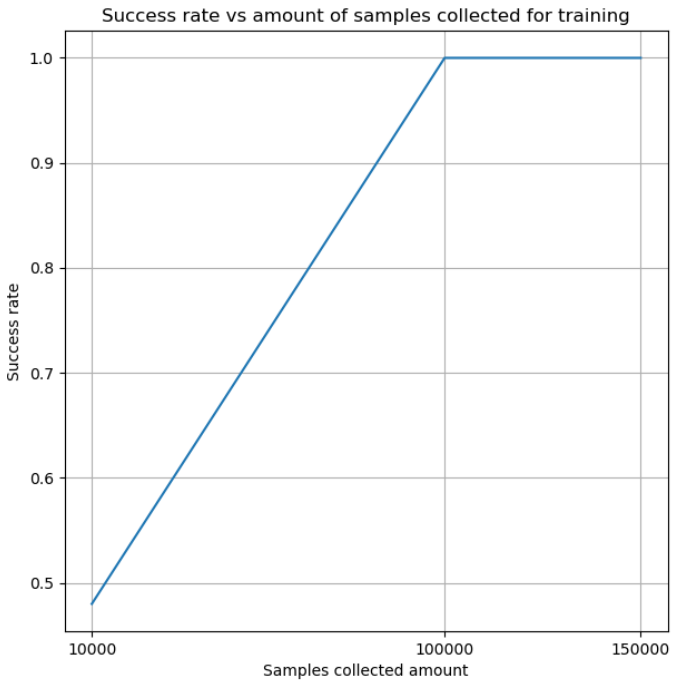


For 150000 samples we get:



Since we have more data to train on, we have less overfit per iteration (as the variance error is in contrast to the amount of samples as seen in ML).

We've collected the final greedy policy over final Q and measured the average success rate vs the amount of data samples collected for training. We did that by adding an external loop over the lspi code that iterates over samples amount and collects the average success rate on the last policy found. Plot:



4 – Q-Learning

1. The reward function according to the reward specification is a cumulative reward -
   For goal not reached at step T-

$$R_T(s \neq goal) = \sum_{t=0}^{T-1} r(s_t, a_t) + r(s_T) = \sum_{t=0}^{T} r(s_t) = \sum_{t=0}^{T} (-1) = -T$$

For goal reached at step T-

$$R_T(s = goal) = \sum_{t=0}^{T-1} r(s_t, a_t) + r(s_T) = \sum_{t=0}^{T-1} (-1) + 100 = \sum_{t=0}^{T} (-1) = 101 - T$$

Setting the reward at the goal state as a relatively high positive value (instead of a 0) affects the agent. The high reward for goal state rewards a case of reaching the goal state after up to 100 actions higher than any case of not reaching the goal state in this time horizon. Also, it enables us to grant negative rewards for each state reached which is not the goal state, instead of 0 reward in previous settings. This way the agents is encouraged to reach goal in fewer steps (higher reward).

The required cumulative reward is set to -75. The car is required to reach the peak in 175 steps at max. Since the cumulative reward in monotonically decreasing by 1 in each step for which the goal isn't reached, if by the 175th step the car is not at the peak of the mountain, the cumulative reward is $R_{25}(s \neq goal) = \sum_{t=0}^{175}(-1) = -176$. Reaching the goal state at a later step will add ≤ 100 to the reward, meaning we will not reach the target of -75.

2. We implemented the following evaluation criterion –
   Randomly selecting 10 starting states, each starting in position −0.5 (bottom of valley), and with a small uniformly distributed velocity. We ran the greedy policy with respect to the current Q function (without exploration) for each iteration.
   The average success rate was defined as–

$$\frac{\#\ iterations\ car\ reached\ top, with\ reward > -75}{\#\ iterations (= 10)}$$

   Before applying the learning process, the average success rate of reaching the top was 0.

3. We Implemented the Q-learning algorithm and calculated the performance as defined in section (2).
   The update of θ was implemented by the relation seen in lectures –

$$\theta_{n+1} = \theta_n + \alpha_n \left(r(s_n, a_n) + \gamma \max_a Q(s_{n+1}) - Q(s_n, a_n) \right) \nabla_\theta Q(s, a, \theta)$$

   In our problem Q is linearly dependent on θ –

$$Q(s, a, \theta) = \theta \cdot \phi(s, a)$$

   $\phi(s, a)$ being the features of the problem. Therefore –

$$\nabla_\theta Q(s,a,\theta) = \phi(s,a)$$

$$\theta_{n+1} = \theta_n + \alpha_n \left(r(s_n, a_n) + \gamma \max_a Q(s_{n+1}) - Q(s_n, a_n) \right) \cdot \phi(s,a)$$

For the terminal state s_n we get –

$$\theta_{n+1} = \theta_n + \alpha_n \big(r(s_n, a_n) - Q(s_n, a_n) \big) \cdot \phi(s,a)$$

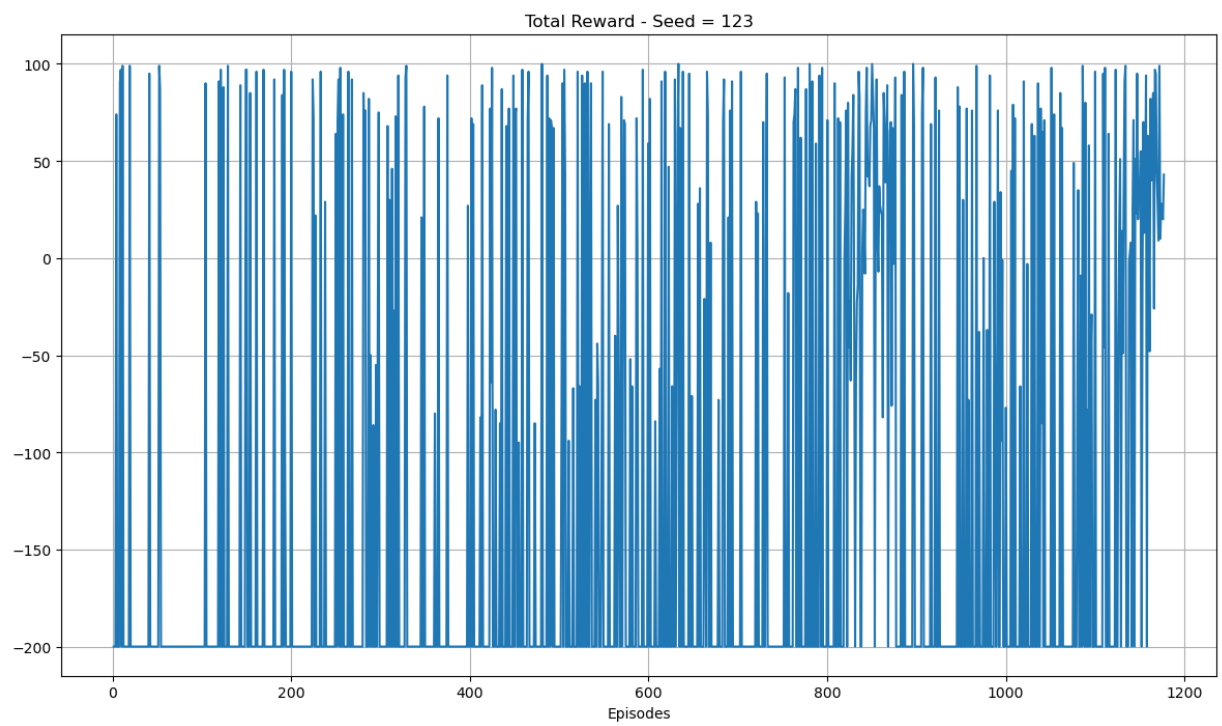
The results for running the algorithm <u>until finding solution</u> (finding solution being – reached performance (=success rate) = 1) are displayed below.
Since initial values of $Q(s,a)$ are randomly selected, in addition to a random factor of selecting starting states, the run time until reaching solution varied between runs.

In results displayed below, solution was found after-

- First Seed = 123: 1179 episodes (first run) and after 1829 episodes (second run). We chose to display for two runs to show variance of running time.
- Second Seed = 234: 199 episodes (third run).
- Third Seed = 345: 419 episodes (forth run).

For all runs we chose - $\gamma = 0.99, \alpha = 0.01, \epsilon = 0.1$

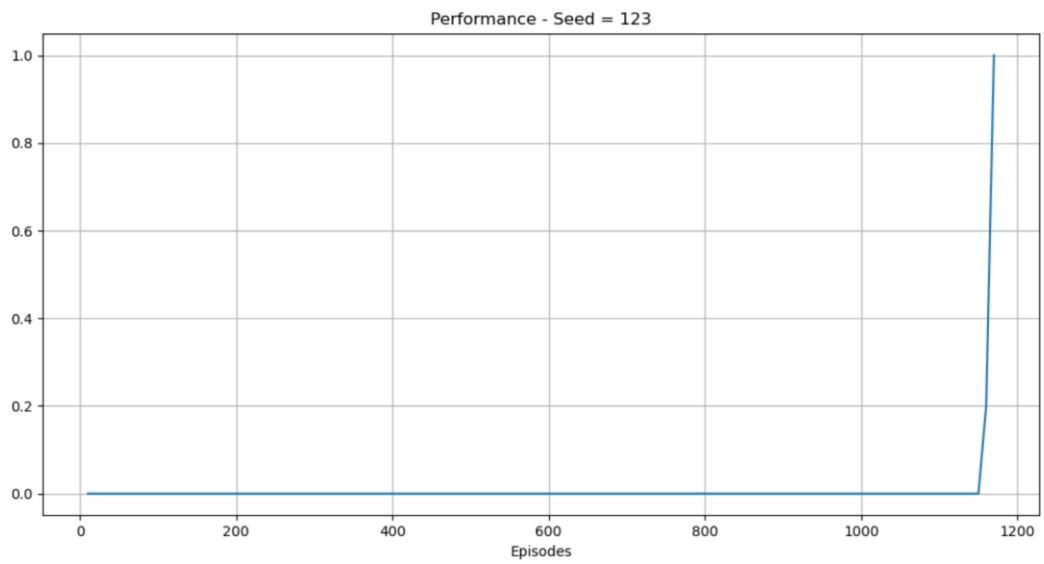
- First run – Seed = 123, 1179 episodes of training.



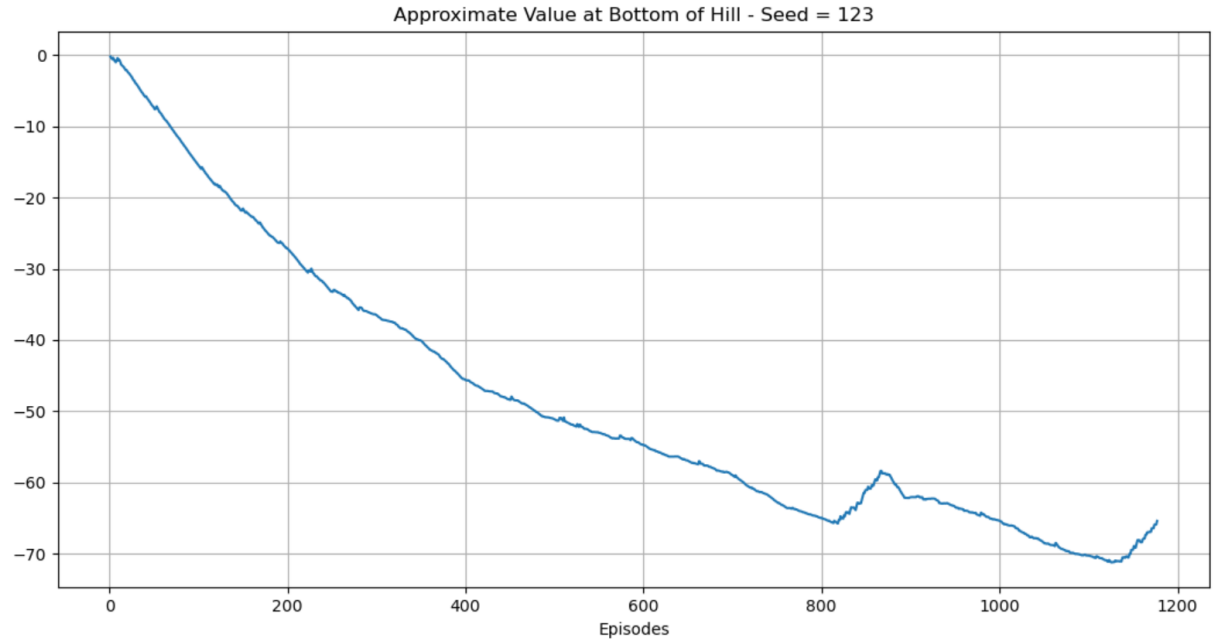
As seen above, a large portion of the training episodes resulted with reward -200. This reward is result of not reaching the top in the episode (max number of steps in episode was set to 200).

Also seen above are episodes with high reward early-on in the training process. Since each episode randomly selects a starting state, there is some probability of choosing a state close to the terminal state. For these episodes reward is high (up to 100) however the evaluation criteria are set for the bottom hill state, meaning high reward in plot above does not indicate closeness to solution. If, however, for a majority of consecutive episodes we see reward ≥ -75 this may indicate with some probability the rewards for bottom hill state being > -75.

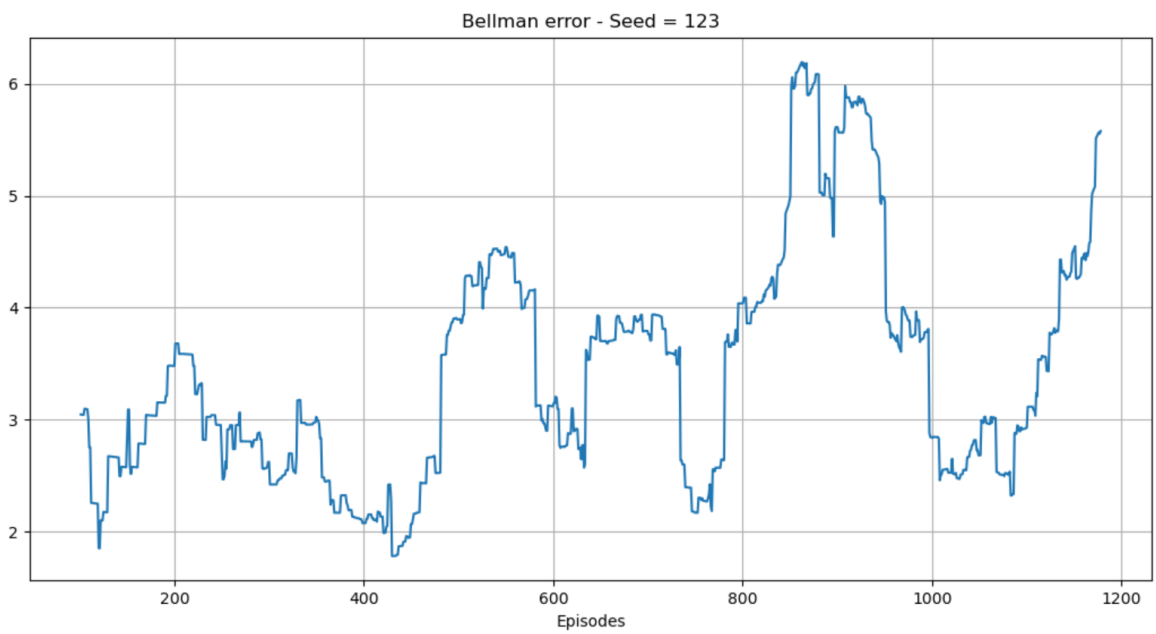
As seen in the last ~50 training episodes, the reward was > -200, meaning the car reached the top for these episodes. Also, the reward for last ~50 training episodes was ≥ -75, indicating with some probability that we are close to reaching solution.



Performance was calculated every 10 episodes. Performance definition is defined in section (2).



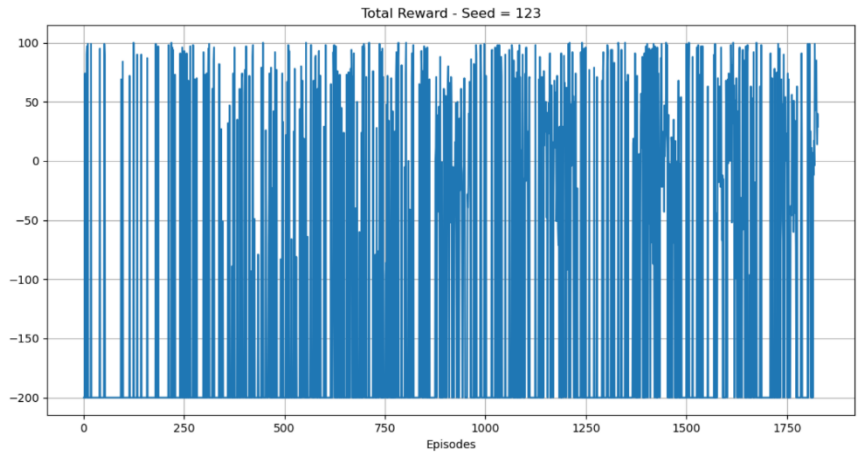
The value of the bottom hill state converges to about -65. We are required to reach reward of -75 and above (for location of bottom hill will small velocity), therefore we expect the training process to converge to this value (or larger).

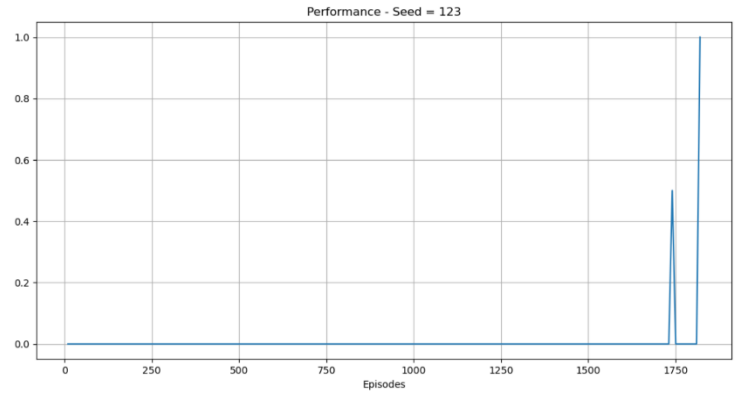


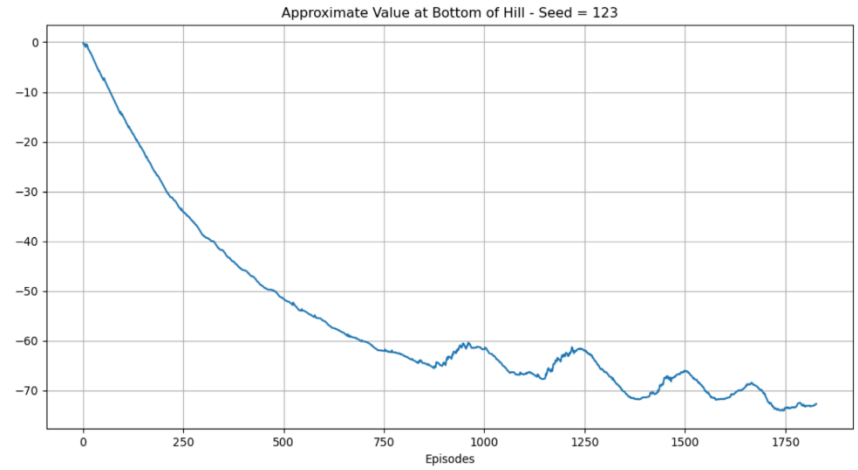
Bellman error displayed is average over 100 last episodes.
In each episode Bellman error is average over all steps of-

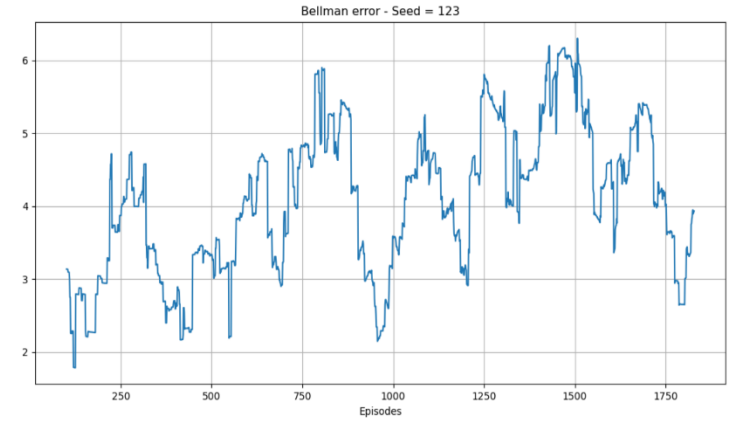
$$Bellman\ Error(for\ step) = \left|\delta_n\left(s_n, a_n, s_{n+1}, r(s_n, a_n)\right)\right|$$
$$= \left|Q(s_n, a_n) - \left(r(s_n, a_n) + \gamma \max_a Q(s_{n+1})\right)\right|$$

- Second run – Seed = 123, 1829 episodes of training

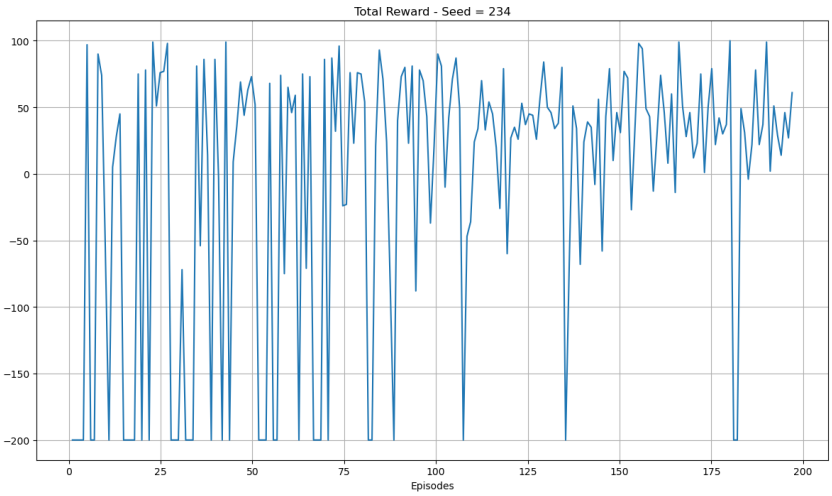


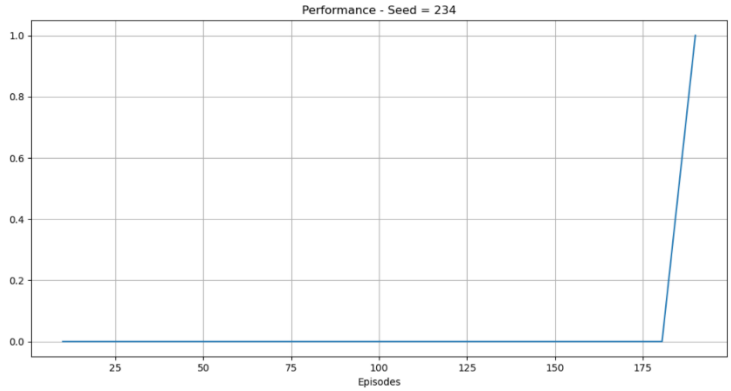


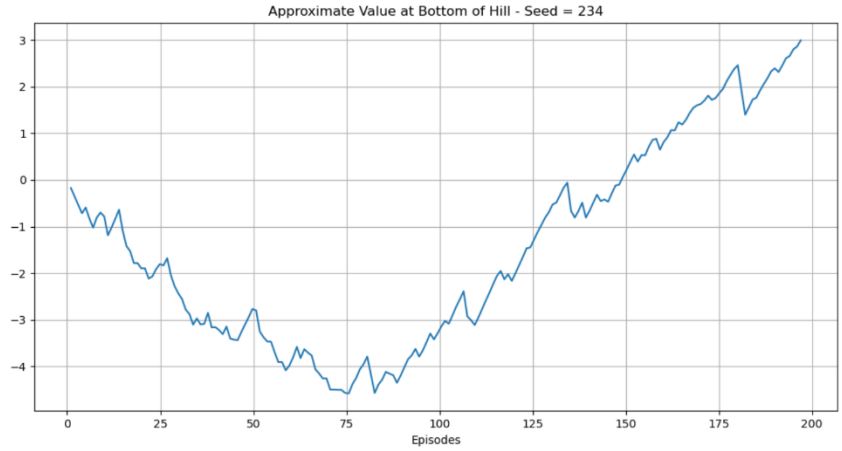


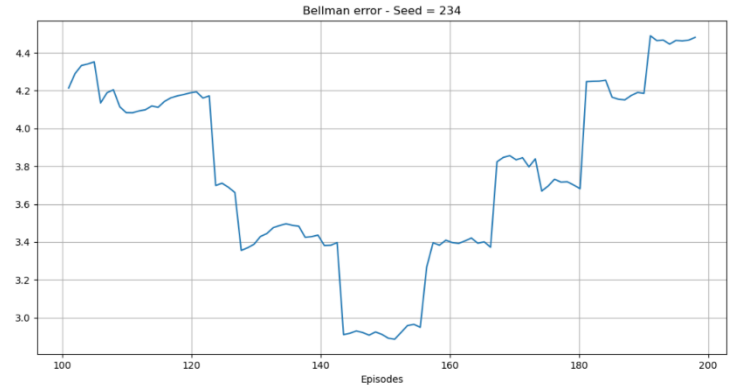


- Third run – Seed = 234, 199 episodes of training

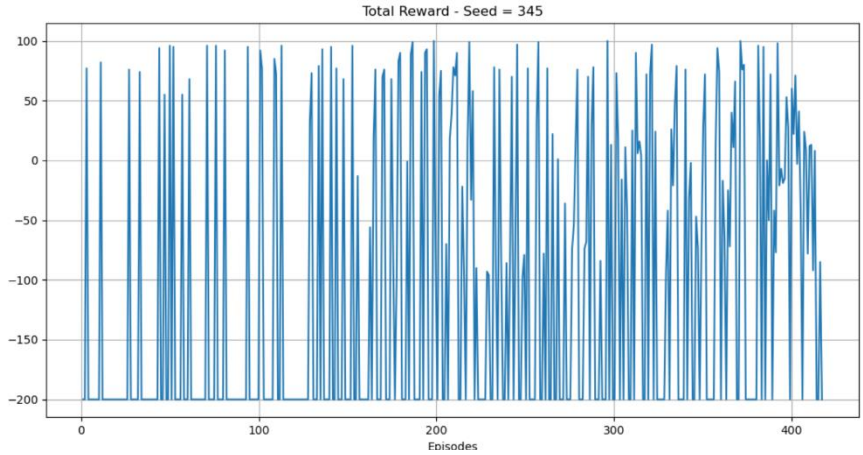


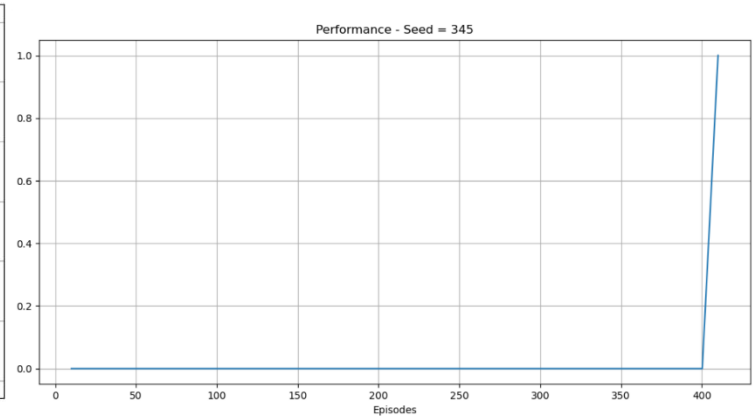


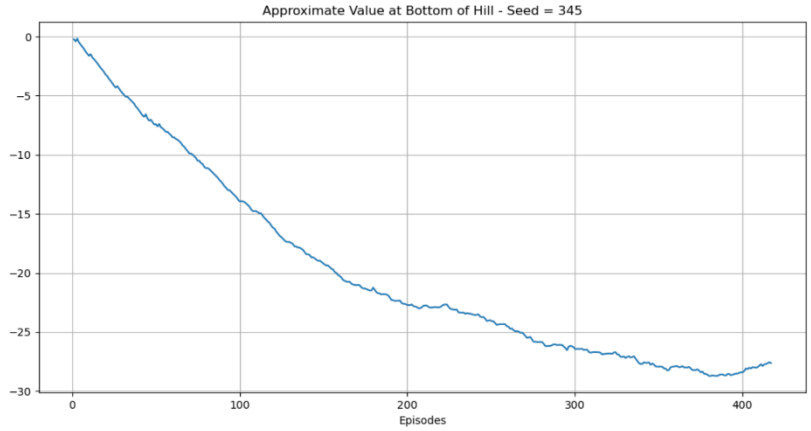


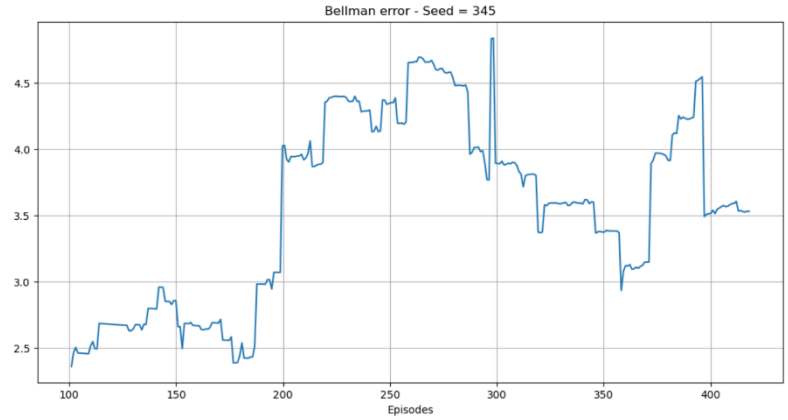


- Forth run – Seed = 345, 419 episodes of training



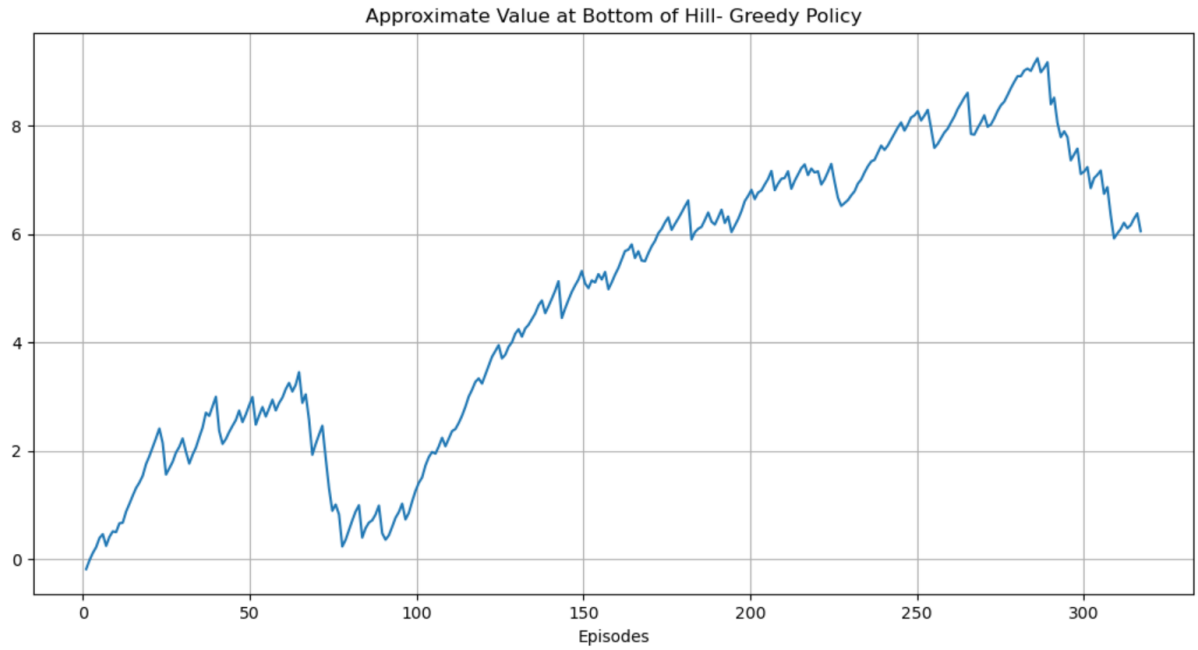




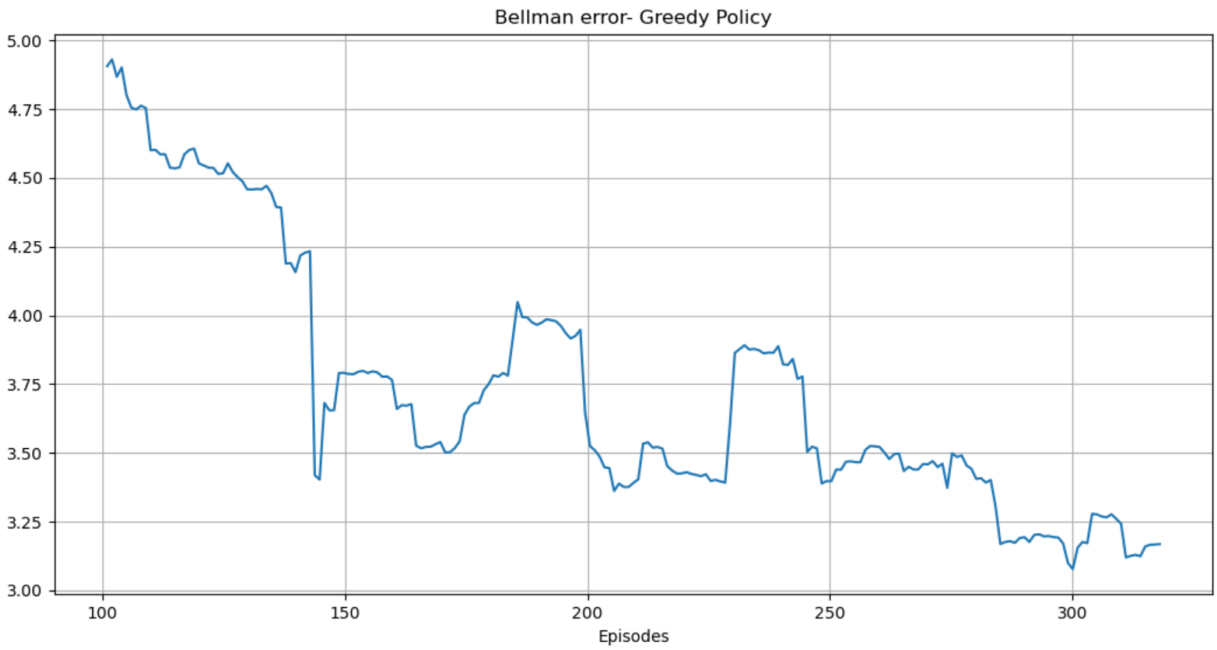


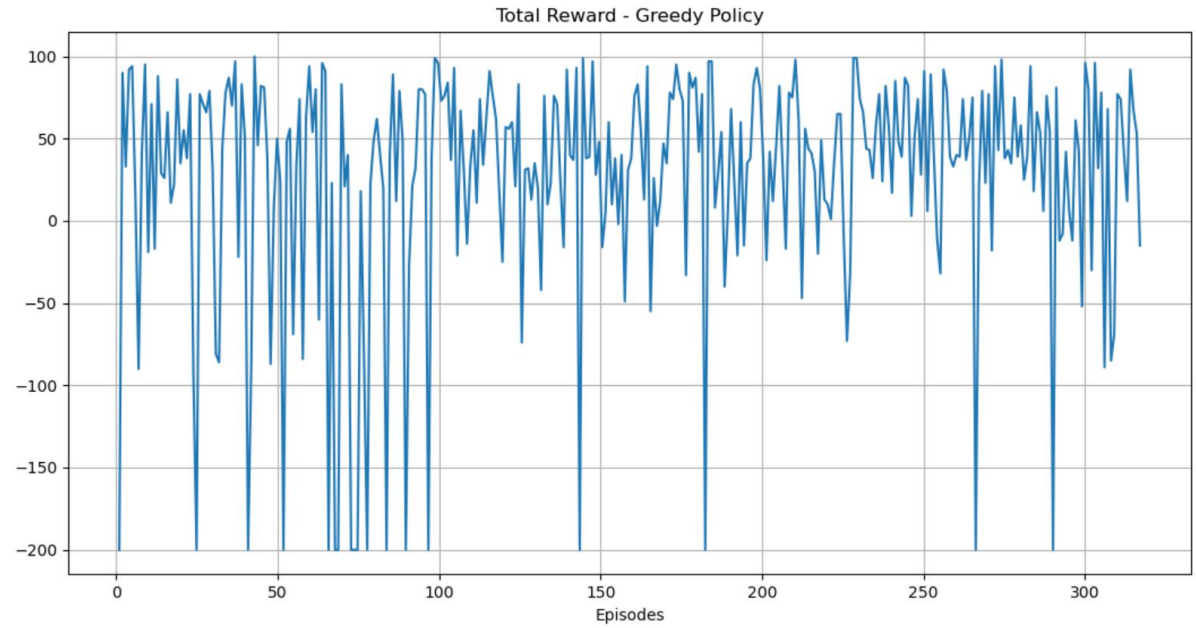
In all seeds we observed no converges to 0 for the Bellman error, however it is bonded (for all cases - $|\delta| < 6$). The converges of the bellman error to 0 is not promised as the Q-Learning in our case is for approximate values of Q (weighted features) and all convergence theorems have been proved for tabular Q function.

4. We ran the greedy policy (no exploration - $\epsilon = 0$) until solution was found (criteria for finding solution is described in previous sections (performance = 1)).
   We used same - $\gamma = 0.99, \alpha = 0.01$ and seed was 123.
   It took the greedy policy 319 iterations to reach solution.

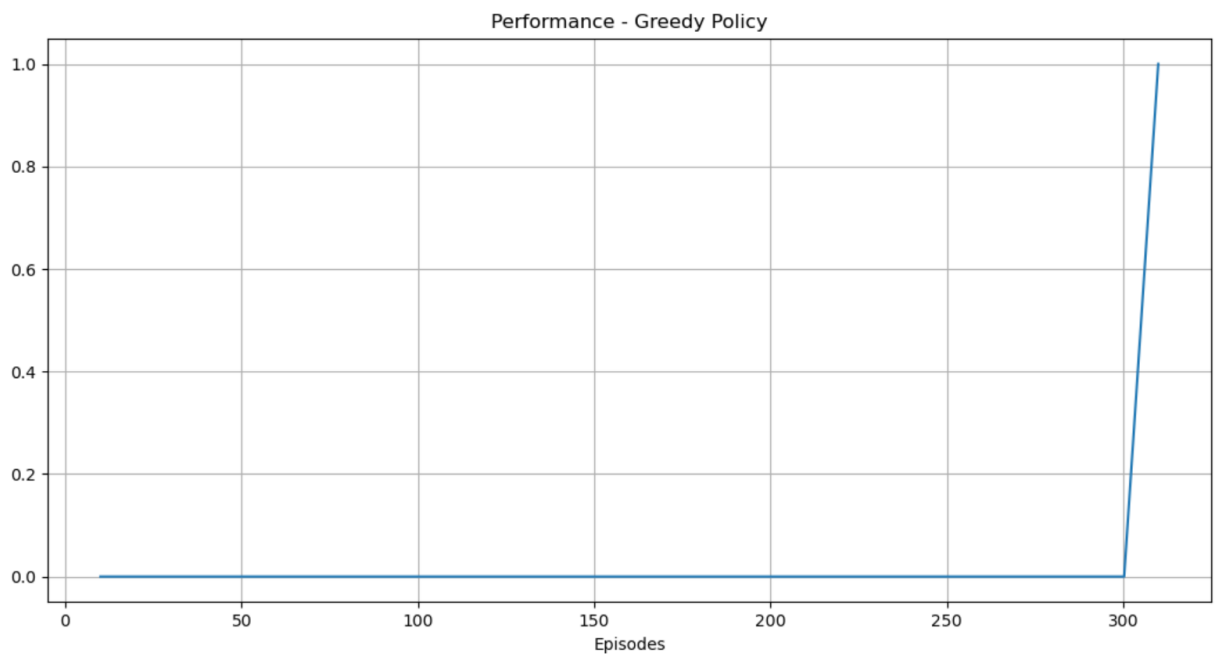


As seen above, the greedy policy reached a solution consisting less than 100 steps to reach the top (expected cumulative reward for bottom hill state is positive). This value is reasonable since we are searching for a solution consisting less than 175 steps.

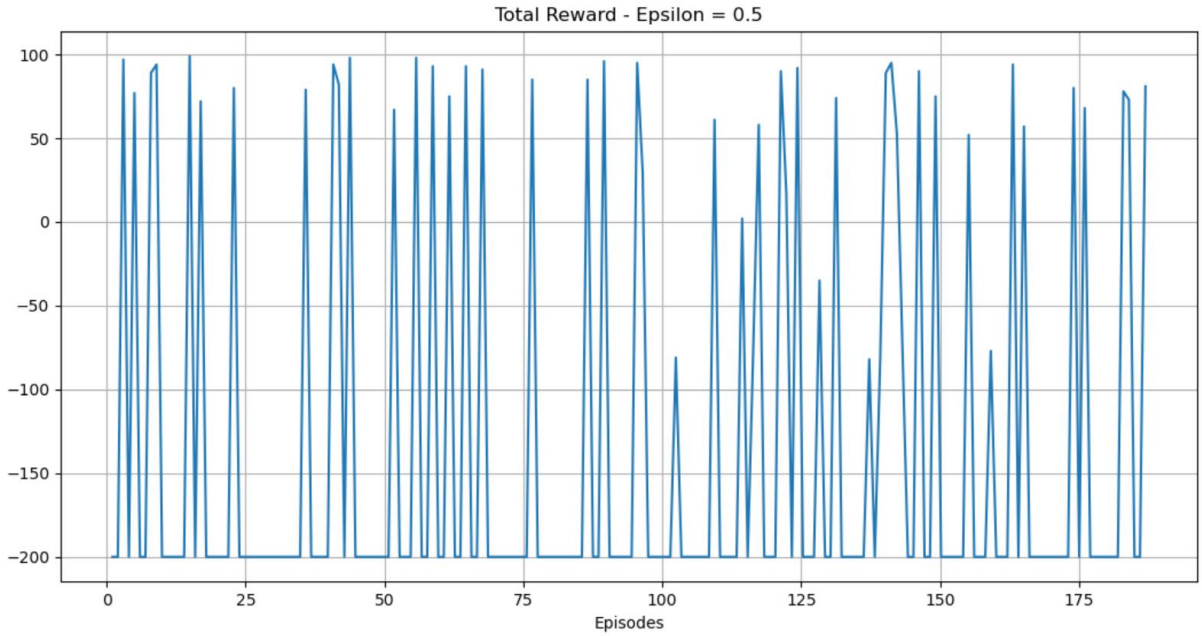




As seen, most episodes reached the top from early on in the training process (reward > -200).

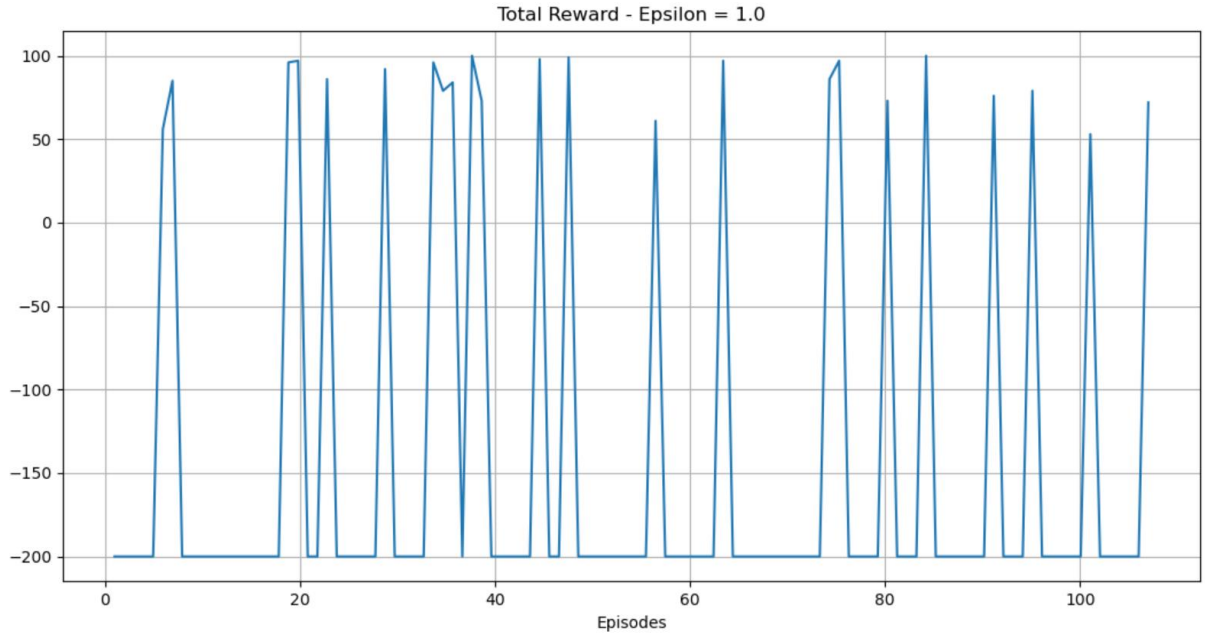


5. We ran the Q-Learning algorithm for 5 different ϵ values - 1.0, 0.75, 0.5, 0.3, 0.01. Plots of total rewards over episodes for each ϵ displayed below. The best value is $\epsilon = 0.5$ due to its short training run time till finding solution-



Solved in 189 episodes.

$\epsilon = 1$ also found solution in short training time, as seen below-

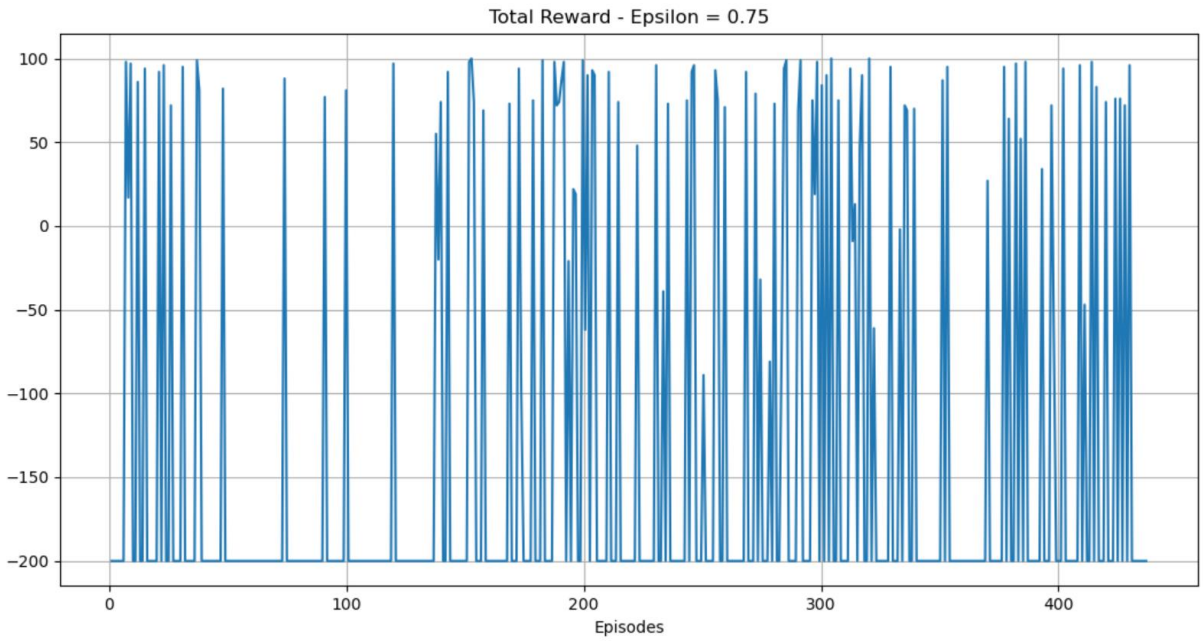


Solved in 109 episodes.

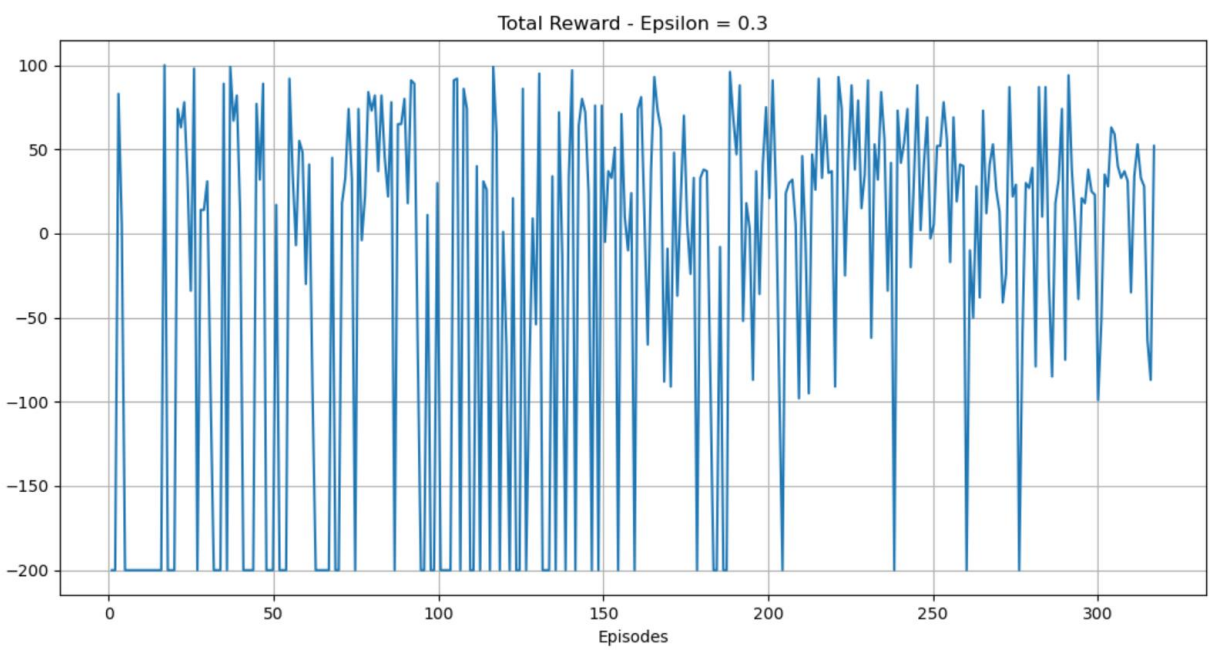
In oppose to the greedy policy, the reward for most episodes was -200 (meaning most episodes did not reach the top). However, convergence to the solution took less episodes thanks to the exploration process. This could be explained due to relatively small state space. Therefore, the exploration method reaches goal state in short time. However, since the training process stops when reaching

solution, the policy reached may be far from optimal – solution in up to 175 steps, while solution of less than 100 steps exists.

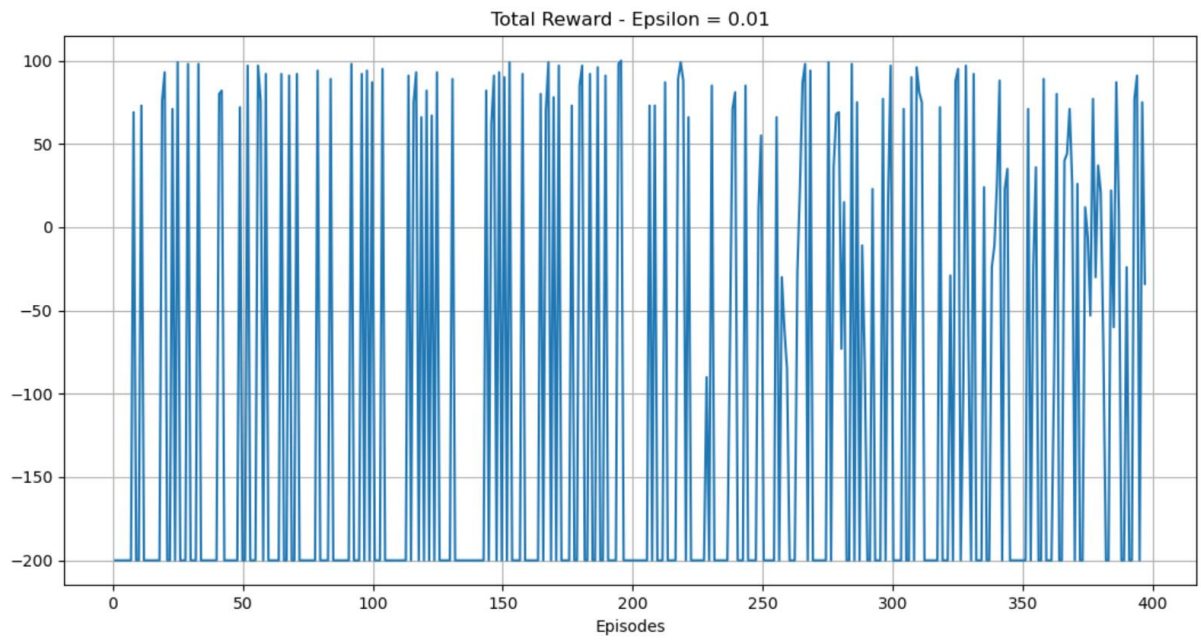
Other ϵ values-



Solved in 439 episodes.



Solved in 319 episodes.



Solved in 399 episodes.

5 – Bonus

The advantage of training while beginning only from the bottom hill state is that the bottom hill state is the state we calculate performance by. We are interested in solution for this state (or very close states of it). By limiting our agent, we achieve updating of the bottom hill state in each training iteration rather than "wasting" episodes on states which aren't critical to finding solution.

The downfall of limiting our agent to start only from the bottom of the hill is that many states will remain unreached. One of the unreached states may be the goal state, holding the positive reward. If no positive reward is obtained, the training process is useless- all iterations will only decrease the values of the states (reward -1).

Our suggestion to overcoming this issue is to initially apply an exploration time period ($\epsilon = 1$), and then applying an ϵ-greedy policy as before with gradually decreasing ϵ values.

We implemented the following update rule for ϵ-

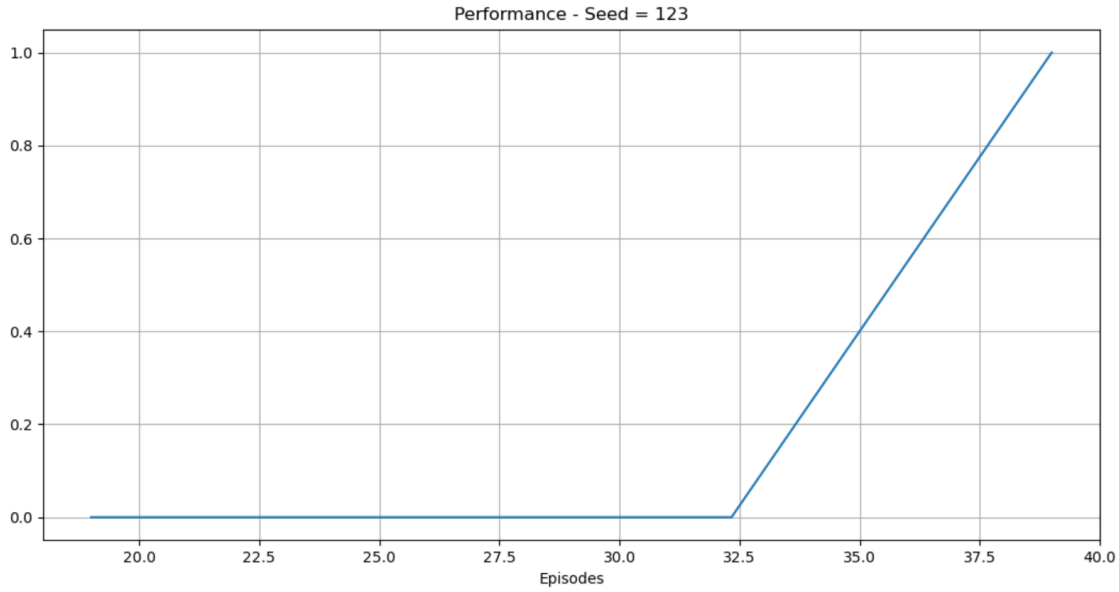
Episodes 0-9: initial $\epsilon = 1$ (solely exploring)

Each tenth episode update – $\epsilon := \epsilon * 0.9$

When reached $\epsilon \leq 0.1 \rightarrow \epsilon = 0.1$

In the settings - $seed = 123$, $\gamma = 0.99, \alpha = 0.01$ we reached solution after 39 episodes. This is the shortest time period of finding solution in compare to the methods of part 3.

Performance plot demonstrating our success -



As seen above solution was found after 39 episodes. We continued the training process after the solution was found, for total of 300 episodes and got the following results: