Ben-Gurion University of the Negev

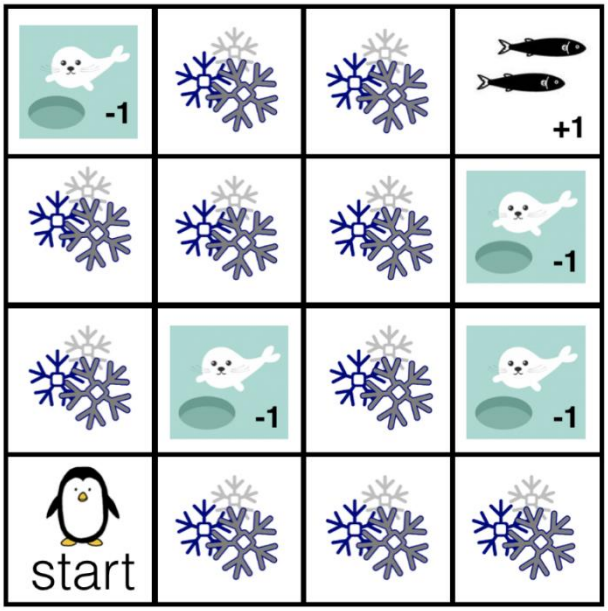
Deep Reinforcement Learning

Model-free RL – Frozen Lake

July, 2020

Lecturer: Armin Biess, Ph.D.

MSc Student: Tom Landman



1. Introduction

This report composed of two main sections: Theory and Practice. In the first section, some key concepts regarding Model-Free Reinforcement Learning, are defined, explained, and referred from Sutton and Barto Book of Reinforcement Learning [1]. In the second section, the results of the given FrozenLake problem are displayed, compared, and discussed following the implementation of the given environment and the required algorithms: Sarsa and Q-Learning.

2. Theory

This section presents important key concepts, algorithms, and optimization methods relate to the theory of *Reinforcement Learning* (RL). Most of the examples and descriptions regarding these concepts are referenced from Sutton and Barto [1].

2.1 ***Epsilon-Greedy***

In the 1st assignment, in order to find the optimal policy, a Markov Decision Process (MDP) model was used for implementing *Value Iteration* and *Policy Iteration* algorithms. Based on Sutton and Barto [1] In both of these algorithms the optimal policy derived by using the simplest action selection rule of selecting greedily an action based on the highest estimated state value function v or action-value of the Q function respectively. For example, in the case of the action-value function, that is, to select at step t one of the greedy actions, A_t^*, for which $Q_t(A_t^*) = \max_a Q_t(a)$. Therefore, this greedy action selection method can be written as $A_t = \underset{a}{argmax}\, Q_t(a)$. Since greedy action selection always exploits the agent's current knowledge to maximize its immediate reward, it doesn't spend any time to select less desired actions in order to explore and see if they might be better and retrieve a better policy. One of the common methods from which the issue of balancing between exploration and exploitation of the agent can be addressed is by choosing an action randomly in a small fraction of time and denoted as the ε-*greedy*. The ε-*greedy* method causes the agent to behave greedily most of the time, but sometimes, say with small probability ε, instead the agent selects an action randomly amongst all the actions with equal probability independently of the action-value estimates. As a result, the agent can both explore the environment and exploit its knowledge. There are several approaches for an agent to learn a certain policy in which the task of tuning the ratio between exploration and exploitation is being addressed. For example, Singh et al. [2] discuss the decaying exploration learning policy that becomes more and more like the greedy learning policy over time, and a persistent exploration learning policy

that does not. The advantage of decaying exploration policies is that the actions taken by the system may converge to the optimal ones eventually, but with the price that their ability to adapt slows down. Further, the class of decaying exploration learning policies characterized by the following two properties:

1. each action is executed infinitely often in every state that is visited infinitely often.

2. In the limit, the learning policy is greedy with respect to its Q-value function with probability 1.

Learning policies that satisfy the above conditions are denoted as *GLIE*, which stands for "greedy in the limit with infinite exploration". For ε-*greedy* exploration [1], at timestep t in state s picks a random exploration action a with probability $\varepsilon_t(s)$ and the greedy action with probability $1 - \varepsilon_t(s)$. When using GLIE the ε-*greedy* initially performs worse because it explores more, however, it performs better because its exploration decreases with time (e.g. exploitation increases).

2.2 ***Sarsa***

Temporal-Difference (TD) learning is a combination of *Dynamic Programming* (DP) and *Monte Carlo* (MC) approaches. Similarly to MC methods, TD-based methods are also model-free meaning that they don't require any given model of the environment's dynamics and can learn directly from raw experience. Further, TD-based methods are bootstrap, meaning there's no need to wait for the outcome, and like in DP, their update estimates based in part on other learned estimates. Note that these methods also used a policy evaluation step denoted as the prediction problem, that of estimating the value function v_π for a given policy π. As for the goal of finding an optimal policy denoted as a control problem, DP, TD, and Monte Carlo methods use a kind of generalized policy iteration (GPI). However, they differ from each other in their approaches to the prediction problem.

Sarsa is an on-policy TD control method which seeks to learn an action-value function rather than a state-value function. In particular, as an on-policy method, Sarsa estimates $q_\pi(s, a)$ for the current behavior policy π and all states s and actions a. As presented in Fig. 1 1 below, recall that an episode consists of an alternating sequence of states and state-action pairs:

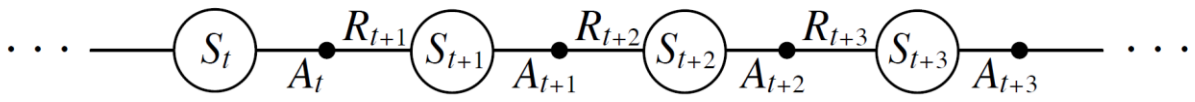


Figure 1 – States action and reward sequences (Sutton and Barto, 2018)

In the first assignment, the *FrozenLake* problem was addressed using Markov Decision Process (MDP) by considering the transitions from state to state and learned the values of states. In this scenario, transitions from state-action pairs to state-action pairs are being considered, and these values of state-action pairs are being learned. The convergence of state values under TD(0) is assured by the following update equation (1) for state-action values:

$$Q(S_t, A_t) \doteq Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (1)$$

This update is done after every transition from a nonterminal state $s_t \in \mathcal{S}$, and in case S_{t+1} is terminal, then its $Q(S_{t+1}, A_{t+1})$ is defined as zero. This update rule uses every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state-action pair to the next. This quintuple structure referred to as Sarsa which is the name of the described algorithm. Similar to all on-policy-based methods, Sarsa continually estimates q_π for the behavior policy π, and at the same time change π toward greediness with respect to q_π. The general form of the Sarsa control algorithm is given in Fig. 2.

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figure 2 - Sarsa: An on-policy TD control algorithm (Sutton and Barto, 2018)

Furthermore, from a convergence perspective, the Sarsa algorithm depends on the selection method of a given policy based on the calculated q function, such as ε-greedy or ε-soft policies in which the Exploration-Exploitation ratio can be tuned. As described in the previous subsection note that ε-greedy can be used with Greedy in the Limit of Infinite Exploration or *GLIE*, by decaying ε using to zero at $\frac{1}{k}$ where k is equal to the number of seen episodes in each iteration and k goes to infinity. It's important to emphasize that unlike MC-based methods in which task termination is not guaranteed for all policies, Sarsa or other step-by-step learning methods don't have this problem because they quickly learn during the episode that such policies are less favorite and as a result, switch to other policies.

From the aspect of Sarsa's backup diagram, as can be seen in Fig. 3 below, the update rule updates a state-action pair, hence, the top node which is the root of the backup must be a small filled action node a that was chosen by a given state s using ε-greedy. Whereas the backup is also from an action node derived from the next state s' using ε-greedy and are both represented by the bottom nodes of the backup diagram.

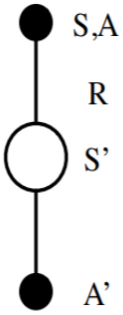


Figure 3 - Sarsa's Sample Backup Diagram

2.3 ***Q-Learning***

Unlike Sarsa which is an on-policy TD control algorithm, *Q-Learning* is an off-policy TD control algorithm, meaning the learned action-value function, Q, directly approximates q_* which is the optimal action-value function, independent of the policy being followed. As a result, early convergence proof is enabled, moreover, the analysis of this algorithm is simplified. One-step Q-learning is defined by the following update equation (2):

$$Q(S_t, A_t) \doteq Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \quad (2)$$

The policy still has an effect in the manner it determines and updates which state-action pairs need to be visited. Under the assumption that all state-action pairs continue to be updated in addition to a variant of stochastic approximation conditions on the sequence of step-size parameters (e.g. α), Q has been shown to successfully converge with probability 1 to q_*. The Q-learning algorithm is presented in Fig. 4.

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$;
  until S is terminal

Figure 4 - Q-Learning: An off-policy TD control algorithm (Sutton and Barto, 2018)

From the aspect of Q-learning's sample backup diagram, as can be seen in Fig. 5 below, the update rule updates a state-action pair, hence, the top node which is the root of the backup must be a small filled action node. Whereas the backup is also from action nodes, maximizing over all possible actions in the next state represented by the bottom nodes of the backup diagram.

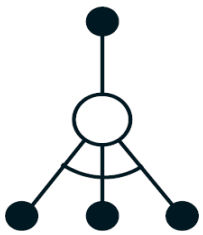


Figure 5 - Q-Learning's Sample Backup Diagram (Sutton and Barto, 2018)

3. Model-free Reinforcement Learning - *FrozenLake*

This section describes the implementation of the famous FrozenLake problem where given a penguin that can take four possible actions = $\{N, E, S, W\}$ on a frozen lake, which is described by a 4x4 grid world with holes and a goal state (fish), both defining terminal states. For transitions, to terminal states, the penguin gets a reward of +1 for the goal state and a reward of -1 for the holes, while in the rest of the transitions the penguin gets a reward of $r(s,a,s') = -0.04$. Also, note that the task was to implement both Sarsa and Q-Learning algorithms and to conduct experiments that involve hyperparameter tuning regarding each algorithm separately, that is to find both the best

policy and approximate values that are close to the optimal q values achieved in the 1st-course assignment. Further, both Sarsa and Q-Learning are based on the following hyperparameters: *Number of Episodes*, *Learning Rate* denoted as α, *Discount Factor* denoted as γ, Initial ε value for the $\varepsilon - greedy$ policy, and finally, the *Decay Rate* of ε (*GLIE*) that responsible for tuning the ratio between exploration and exploitation as discussed in the previous sections. All of these hyperparameters were tuned except the discount factor that is given, e.g. $\gamma = 0.9$.

First, I tried to use a Grid search and Harmony Search (HS) [3] techniques, however, since the optimization took too much time and since it was hard to control these parameters without understanding how the agent is affected by them during the process itself, I decided to explore the evidence regarding each hyperparameter modification as discussed in the next subsections.

a. Sarsa Implementation:

First, the Sarsa algorithm was implemented using exploring starts to increase the robustness of the learned policy by changing the initial state per each episode randomly. In order to address the task of hyperparameter tuning, first, I used an initial configuration of the following hyperparameters: $\alpha = 0.5$, $\varepsilon = 1$, GLIE's with an exponential decay rate of 0.001, and 100,000 episodes. By using a learning rate of 0.5 it means that the future agent's observations have a relatively high effect on the present policy derived from the q function values. It's important to emphasize that by exponential decay it means that in each episode the current epsilon is multiplied and decreased by a factor of 0.999, that is for the agent to be able to explore much more especially during the first episodes and later, to exploit the learned policy the closer he gets to the last episodes where the minimal epsilon defined as 0.001, meaning a 0.1% of exploration for the agent's $\varepsilon - greedy$ policy. Further, as can be seen in Sutton and Barto's book [1], after executing Q-Learning, I plotted the reward of each episode in addition to a smoothed average reward calculated by a moving average with a window size of 100 hence to clarify the agent's learning convergence trend. As presented in Eq. 3, The moving average parameters are n, the number of periods (episodes) in the moving average (e.g. window size), and R_i the total reward retrieved per episode i

$$MA_i \doteq \frac{\sum_{i=1}^{n} R_i}{n} \quad (3)$$

By viewing Fig. 6 below, it seems that the max smoothed reward of the agent is around 0.56 which is quite low and by looking both at the orange trend line and especially and the blue lines represent the reward per episode it seems that the variance between the episodes is high and noisy.

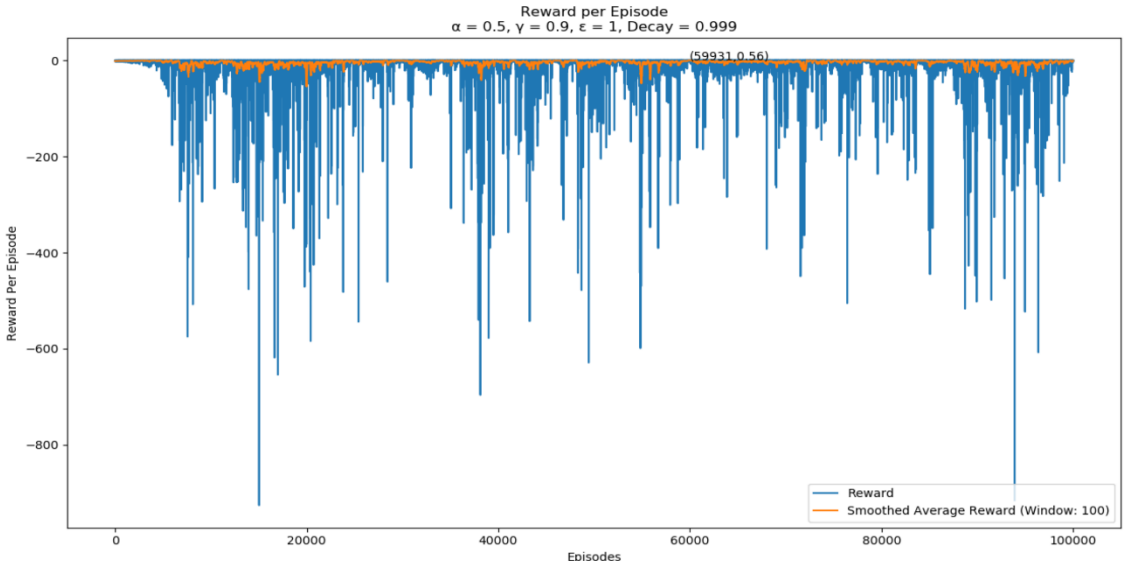


Figure 6 - Sarsa - Initial Configuration

As a result, in addition to the fact that the convergence took almost 4 hours (which is a long time relative to this project's time constraints), the achieved policy was really poor meaning that the agent learned a bad policy (an averaged reward of -4.19 among all of the episodes) and for states that are close to the holes, the agent prefers to avoid the risk by staying in the same position instead of going to another direction. Therefore, I decided to tune the learning rate parameter α by decreasing it from 0.5 to 0.1 for the agent to be able to learn a better policy based on a smaller learning step size and less weight for the update based on the future episodes.

In Fig. 7 below, during the first five thousand episodes, there's a drastic increment in the trend of the smoothed reward and then the trend line is more stable. However, by viewing the blue trend line it seems that the rewards per episode are still differed and varied and therefore, result in a noisy trend.
Also, note that there's a big improvement in the max smoothed reward resulted in 0.71 which is much closer to the maximal reward (+1). Further, in this configuration, the algorithm execution took ~16.5 minutes which is a much more reasonable time relatively to the execution time of the first configuration, and also the derived policy was much closer to the optimal policy.

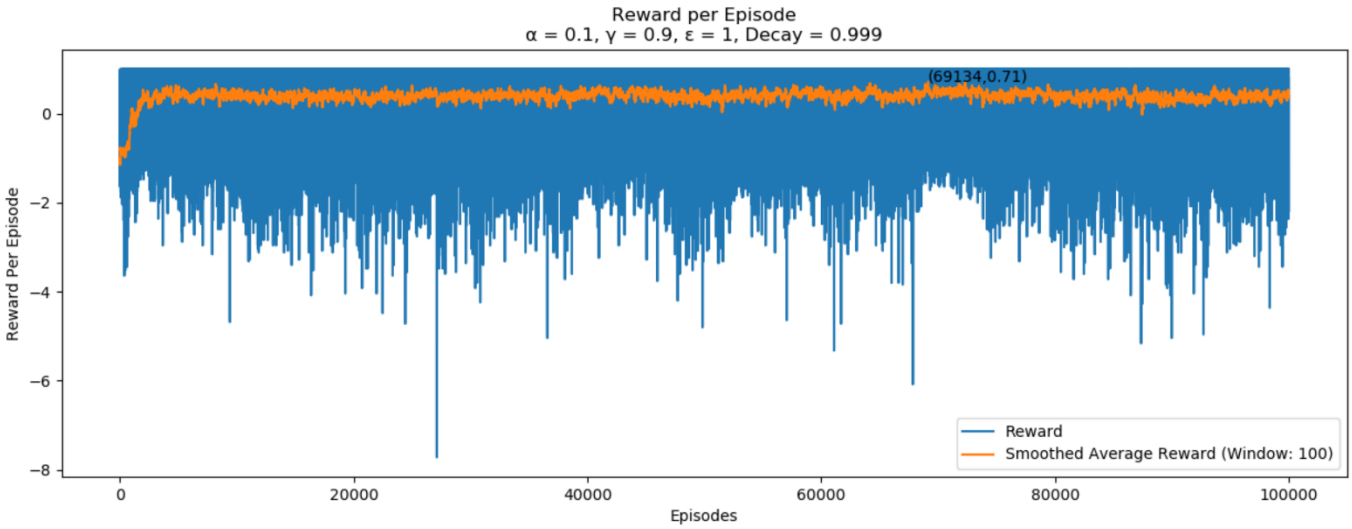


Figure 7 - Sarsa - Decreasing learning rate to 0.1

In order to better understand the root cause for the trends noisiness, I explored the number of time steps per each episode and plotted the result as can be seen in Fig. 8. Note the differences between the number of time steps during all of the episodes (except the interval around 70,000 episodes) that can be affected by the agent's exploration-exploitation ratio.

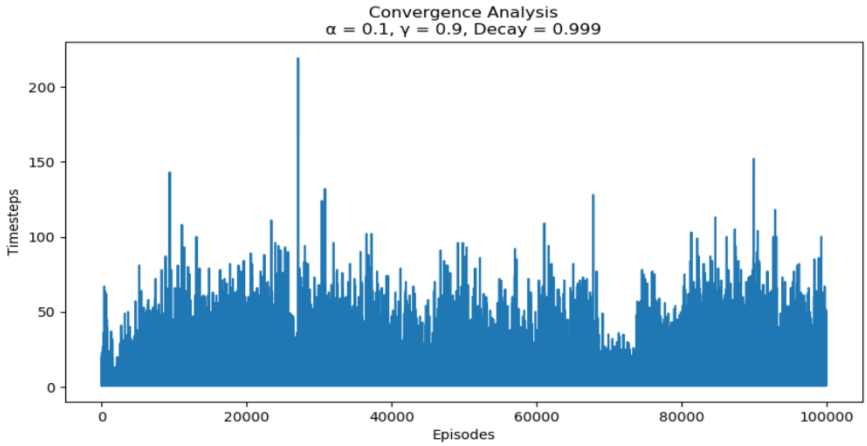


Figure 8 - Convergence Analysis of the decreased learning rate configuration

Furthermore, to determine how close are the resulted values of the agent learned policy, denoted as v_{Sarsa} to the optimal values achieved in the 1st MDP assignment, denoted as v_{MDP}, I used the L_1 norm on the subtraction of these two vectors based on the following Eq. 4:

$$L_1\ norm\ =\ \|v_{Sarsa} - v_{MDP}\|_i \doteq\ \sum_i \left|v_{Sarsa_i} - v_{MDP_i}\right| \quad (4)$$

The reason for prioritizing L_1 over $L_2\ norm$ is to better reflect the real distance between both vectors despite $L_2\ norm$ is known to be more robust to outliers since the distance is being squared and then squared root. Last in this configuration the resulting $L_1\ norm$ distance was 0.978 and an averaged reward of 0.387 among all of the episodes.

In order to decrease the variance and timesteps noisiness, I reduced the initial exploration-exploitation from 1 to 0.9, that is to let the agent to better exploit the learned policies from the beginning of its journey with an initial 10% chance for exploitation that it increased as long the agent advance towards future episodes. As can be seen in Fig. 9 below, the orange trend line of the smoothed reward continuously increased until it reaches a plateau after 7,000 episodes. Although, a faster execution time of 16 minutes, the max smoothed reward achieved was 0.67 which is less than the last configuration which is better than the last experiment. Further, the resulting $L_1\ norm$ distance was 0.87 which is closer to the q values achieved in the 1st assignment and the averaged reward achieved was 0.39.

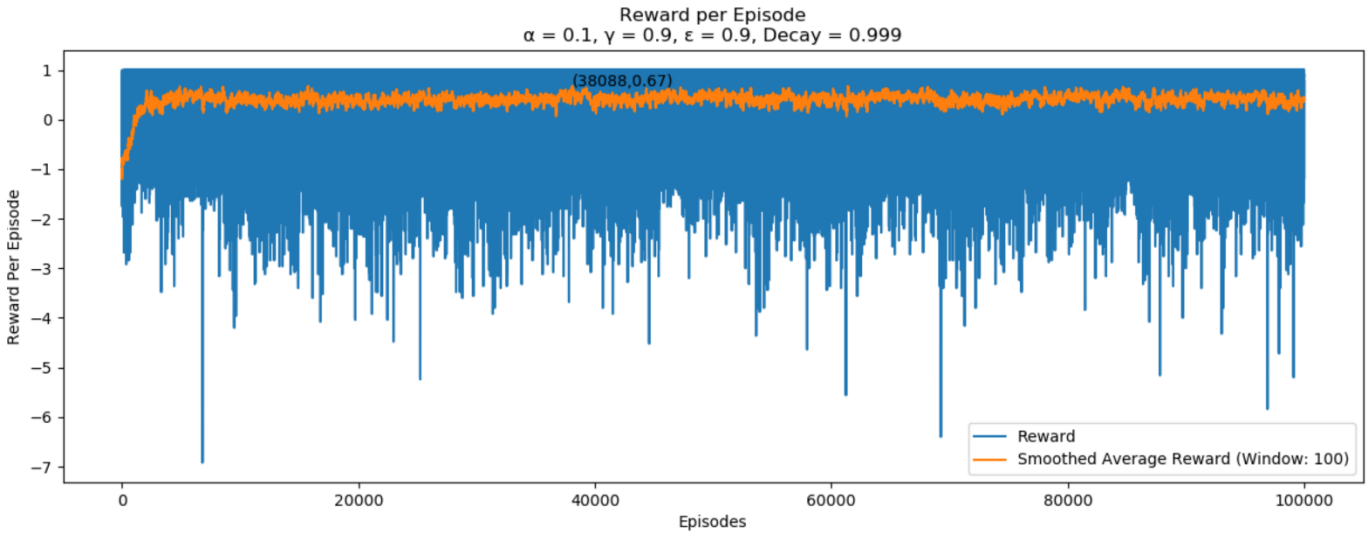


Figure 9 - Sarsa - Decreased initial GLIE's epsilon of 0.9

From the timesteps convergence analysis perspective as can be seen in Fig. 10, despite the amount being reduced compared to the last configuration, there's still no convergence and high variance between the episodes.

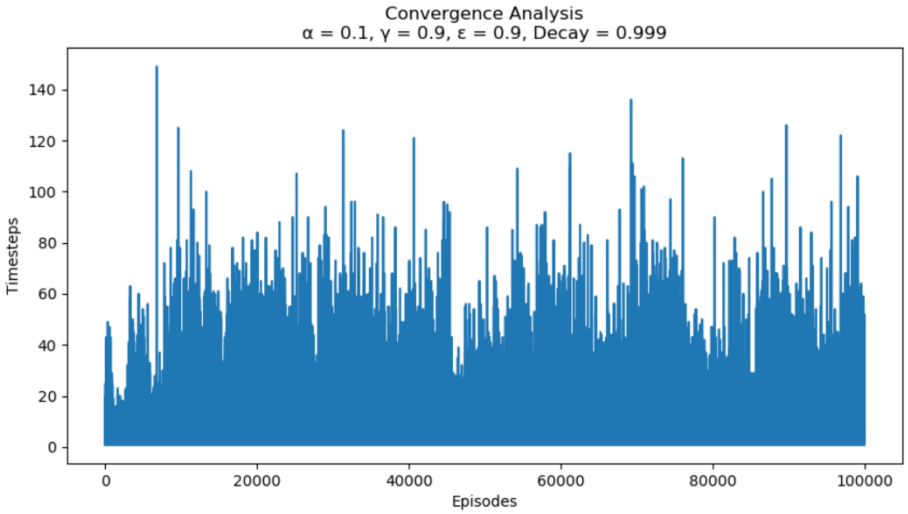


Figure 10 - Convergence Analysis of decreased epsilon configuration

Based on the above evidence, and since the retrieved policy wasn't optimal I decided again to reduce the learning rate from 0.1 to 0.01, hence to see its interaction with the new initial epsilon value and to see if it will cause the agent to learn and to converge to a better policy.

As can be seen In Fig. 11 below, the orange trend line of the smoothed reward continuously increased until it reaches a plateau after 10,000 episodes. In addition to a faster execution time of ~ 8.3 minutes, note the improved max smoothed reward of 0.76 which is better than the last experiment. Further, the resulting $L_1\ norm$ distance was 0.22 which is significantly closer to the q values achieved in the 1st assignment and the averaged reward achieved was 0.45.

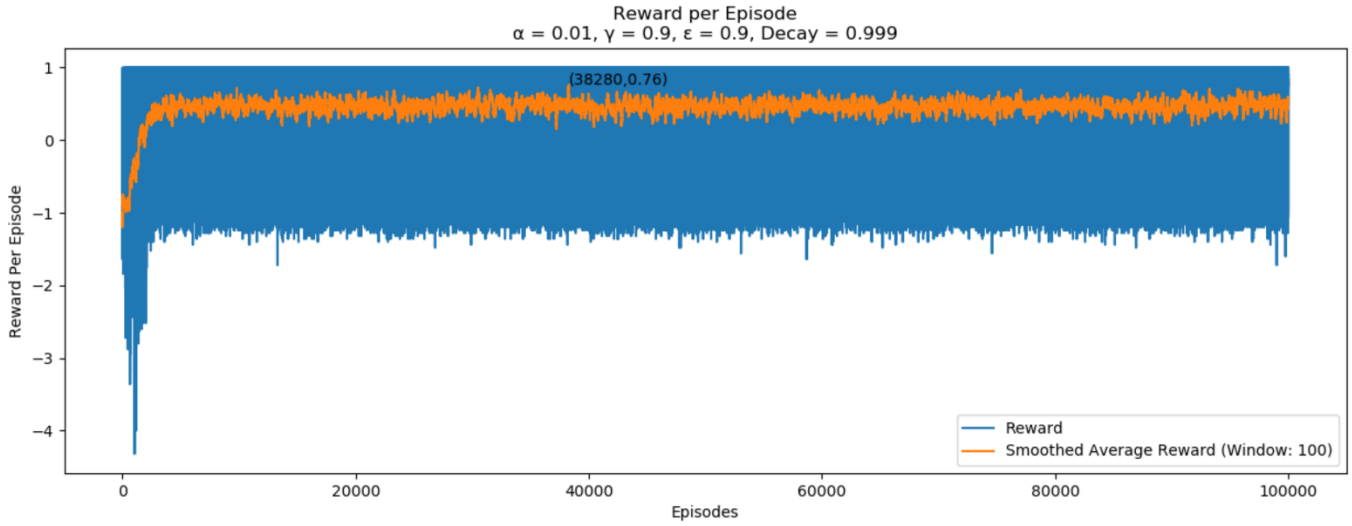


Figure 11 - Sarsa - A reduced learning rate of 0.01

In addition to the abovementioned result, the big improvement was observed in Fig. 12, in which the timesteps amount per each episode is exponentially decreased after the first 5,000 episodes, and also can be reflected in the blue trend of the total reward per episode.

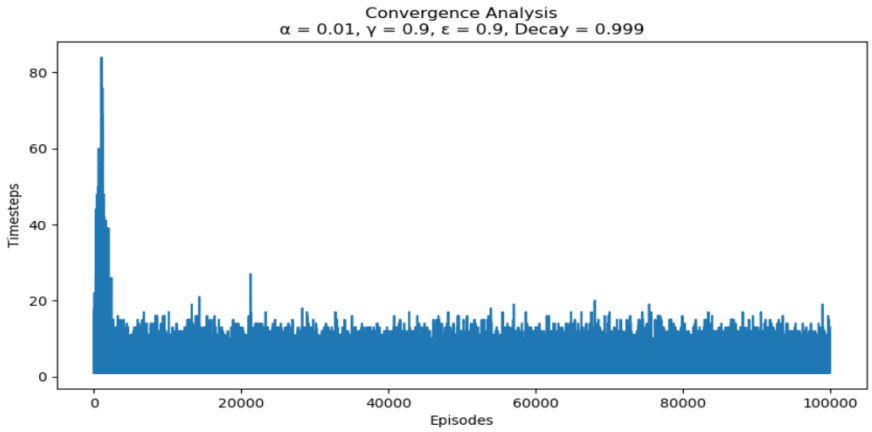


Figure 12 - Convergence Analysis of decreased alpha confiuration

Also note, that in this hyperparameters configuration, the agent learned the optimal policy. However, in order to explore if the performance can be further improved, I executed Sarsa with the exact hyperparameters configuration but with a minor change in the learning rate to see if decreasing it further will result in q-values that are much closer to the values achieved in the 1st assignment. I compared the agent performance by modifying the learning rate of $\alpha = 0.009, 0.008, 0.007$, and compared their $L_1\ norm$ values which were 0.2, 0.16, and 0.129 respectively. Finally, since the minimized $L_1\ norm$ means low distance, I chose a learning rate of 0.007 and plotted the results appear in Fig. 13 and Fig. 14. Note that the charts are similar to those of the last configuration except in the aspect of the number of time steps appears the convergence analysis in which the maximal value decreased from 80+ to 70 and therefore, supports the claim that the agent converged to a better q values represented and determine its policy.

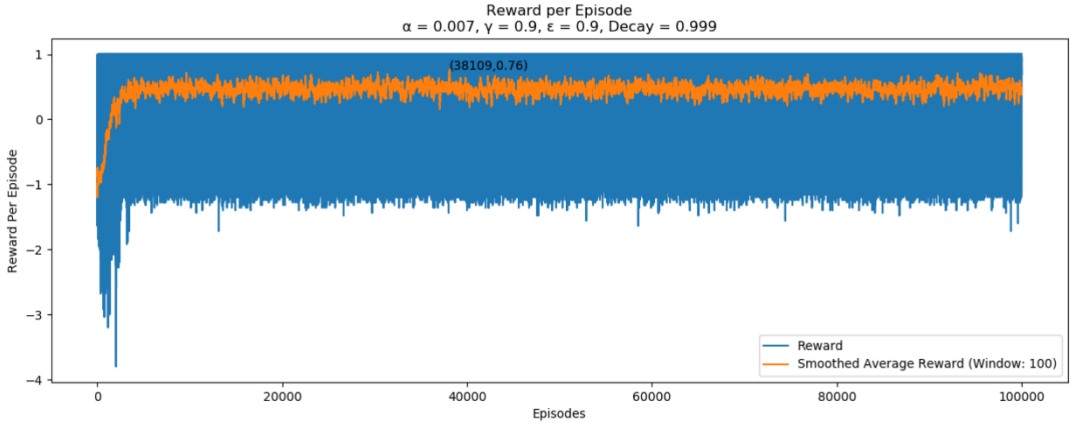


Figure 13 - Sarsa - A reduced learning rate of 0.007

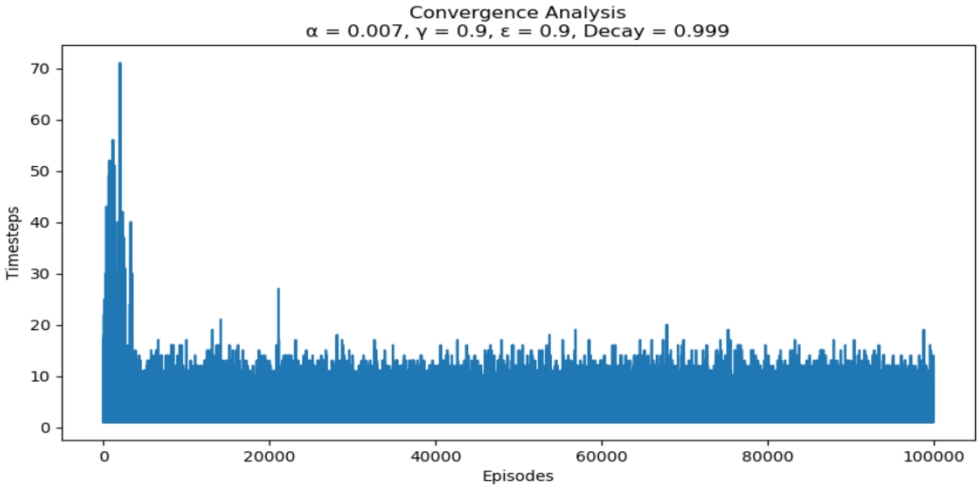


Figure 14 - Convergence Analysis of decreased alpha of 0.007

The execution took ~8 minutes and the averaged reward achieved in this configuration was 0.45. Also, note that even though the final configuration's outcome derived the optimal policy, and since the highest max smoothed average reward obtained in the 38109th episode, I explored and compared the $L_1\ norm$ of each one of the episodes in the interval in order to see if the agent learned the optimal policy in earlier episodes. As can be seen in Fig. 15 below, the lowest $L_1\ norm$ retrieved in the 56176th episode and was equal to 0.03 which outperformed substantially on the rest of Sarsa's explored configurations.

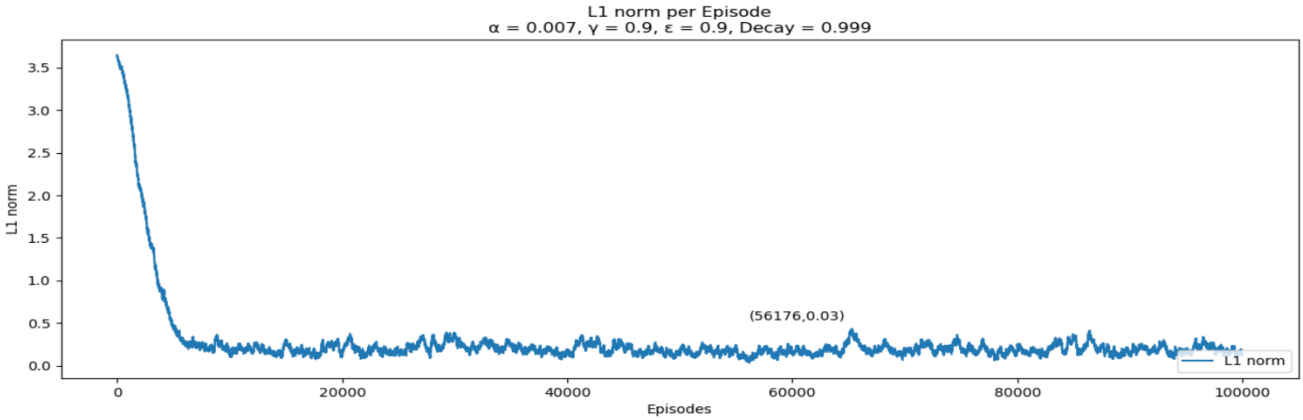


Figure 15 - $L_1\ norm$ Comparison between the episodes of the best found hyperparameters configuration

Furthermore, the obtained q values, the state values (colored with green triangles), and the optimal policy derived with the following hyperparameters configuration of $\alpha = 0.007,\ \varepsilon = 0.9,$ decay rate of 0.001, and 56176 episodes are presented in Fig. 16 that was generated using my implementation of the **plot_actionValues()** function. Hence, from the agent starting state 4, the agent learned the quickest routes to its goal, both by turning east or going north, however by viewing the values for both routes the agent prefers to go north since it a much safer route than the east one and with fewer holes during the route. Further, in case the agent is located in state 8, the policy guides him to continue east despite the riskier road that is because it is much shorter than turning west to state 4 and then going up

north. Note the small difference between the best state values retrieved in the 1st MDP assignment and presented in Fig. 17 to the best state values (green triangles) derived from the $Sarsa$ algorithm.

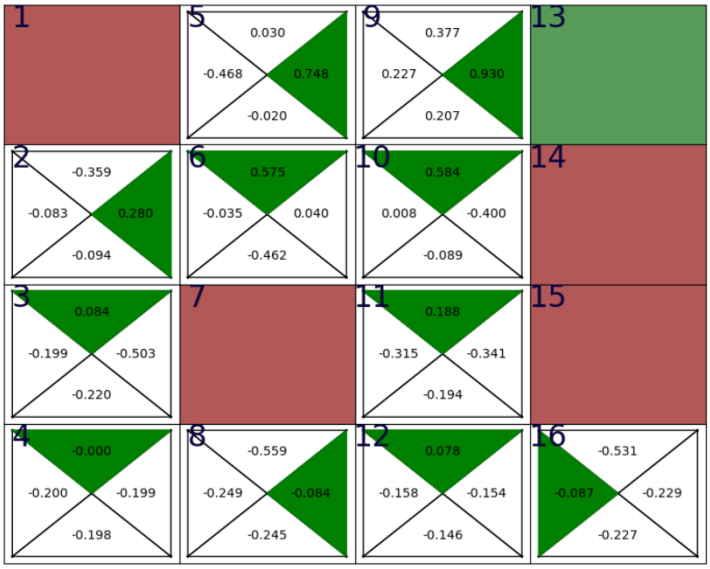


Figure 16 - Q-values, state values and the optimal Policy

Figure 17 - State Values for the optimal policy from MDP assignment

b. *Q-Learning Implementation*:

The Q-Learning algorithm was implemented using exploring starts to increase the robustness of the learned policy by changing randomly the initial state for each episode. In order to address the task of hyperparameter tuning, first, I used an initial configuration of the following hyperparameters: $\alpha = 0.5$, $\varepsilon = 1$, GLIE's with a linear decay rate of 0.01, and 100,000 episodes. It's important to emphasize that by linear decay it means that in each episode the current epsilon is linearly decreased by 0.01, that is for the agent to be able to explore much more in the first episodes and later, to exploit the learned policy the closer he gets to the last episodes where the minimal epsilon defined as 0.001, meaning a 0.1% of exploration for the agent's $\varepsilon - greedy$ policy. Further, as can be seen in Sutton and Barto's book [1], after executing Q-Learning, I plotted the reward of each episode in addition to a smoothed average reward with a window size of 100 hence to clarify the agent's learning convergence trend. By viewing Fig. 18 below, it seems that the max smoothed reward of the agent is around 0.47 which is quite low and by looking at the orange trend line it seems that the learning is quite flat, stable, and without drastic increment.

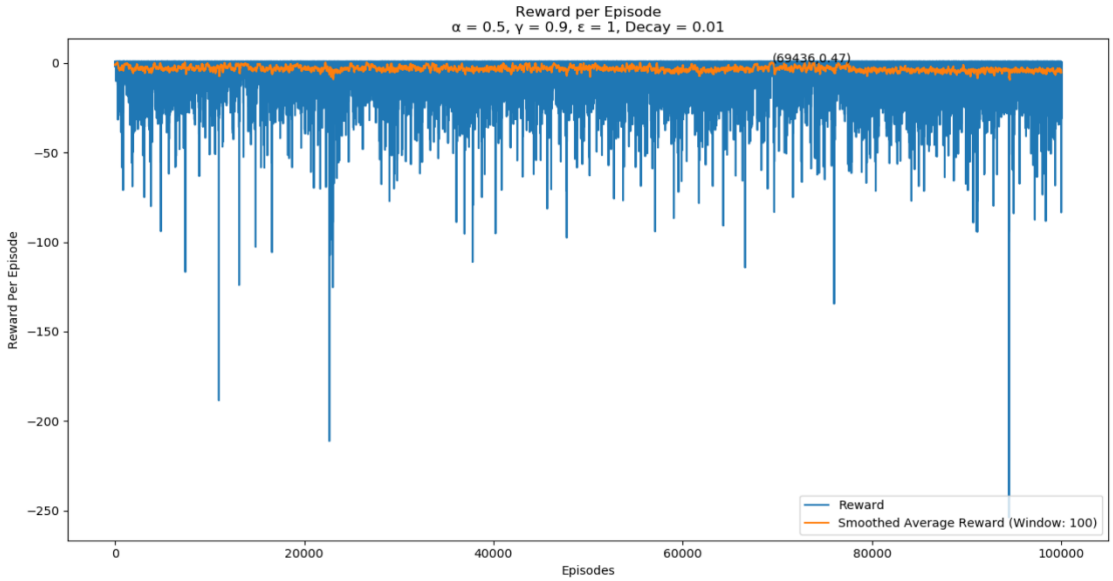


Figure 18 - Q-Learning - Initial Configuration

Moreover, besides of the fact that the convergence took almost 4 hours (which is a long time relative to this project's time constraints), the achieved policy was really poor meaning that the agent learned a bad policy (an averaged reward of -3.14 among all of the episodes) and for states that are close to the holes the agent prefers to avoid the risk by staying in the same position instead of going to another direction. Therefore, I decided to tune the learning rate parameter α by decreasing it from 0.5 to 0.1 for the agent to be able to learn a better policy based on a smaller step size. In Fig. 19 below, during the first thousand episodes, there's a drastic increment in the trend of the smoothed reward and then the trend line is more stable except

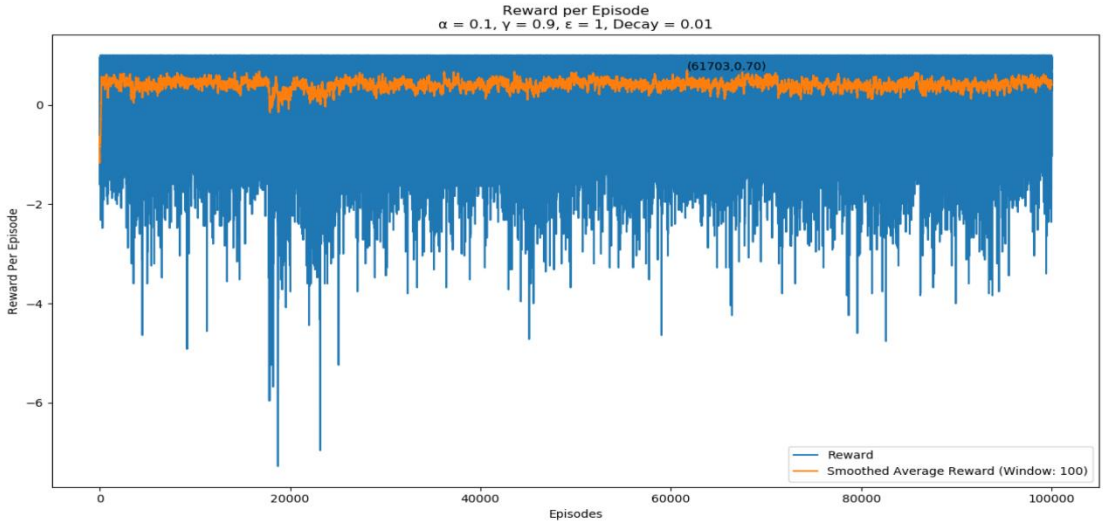


Figure 19 - Q-Learning - Decreased Learning Rate of 0.1

for the interval around the 20,000 episodes which is much noisier and less stable comparing to the rest of the episodes. Also, note that there's a big improvement in the max smoothed reward resulted in 0.7 which is much closer to the maximal reward (+1). Further, in this configuration, the algorithm execution took 21 minutes which is a much more reasonable time relatively to the execution time of the first configuration, and also the derived policy was much closer to the optimal policy. Furthermore, to determine how close are the resulted values of the agent learned policy, denoted as v_{QL} to the optimal values achieved in the 1st MDP assignment, denoted as v_{MDP}, I used the L_1 norm on the subtraction of these two vectors based on the following equation 4:

$$L_1\ norm\ =\ \left\|v_{QL} - v_{MDP}\right\|_i \doteq\ \sum_i \left|v_{QL_i} - v_{MDP_i}\right| \quad (4)$$

In this configuration the resulting $L_1\ norm$ distance was 0.8635 and an averaged reward of 0.4 among all of the episodes, results that are much better than the 1st configuration. As a result, and since the retrieved policy wasn't optimal I decided again to reduce the learning rate from 0.1 to 0.001 to see if it will cause the agent to learn a better policy. As can be seen In Fig. 20 below, the orange trend line of the smoothed reward continuously increased until it reaches a plateau after 10,000 episodes. In addition to a faster execution time of 11 minutes, note the improvement of the max smoothed reward resulted in 0.75 which is better than the last experiment. Further, the resulting $L_1\ norm$ distance was 0.062 which is significantly closer to the q values achieved in the 1st assignment and the averaged reward achieved was 0.43.

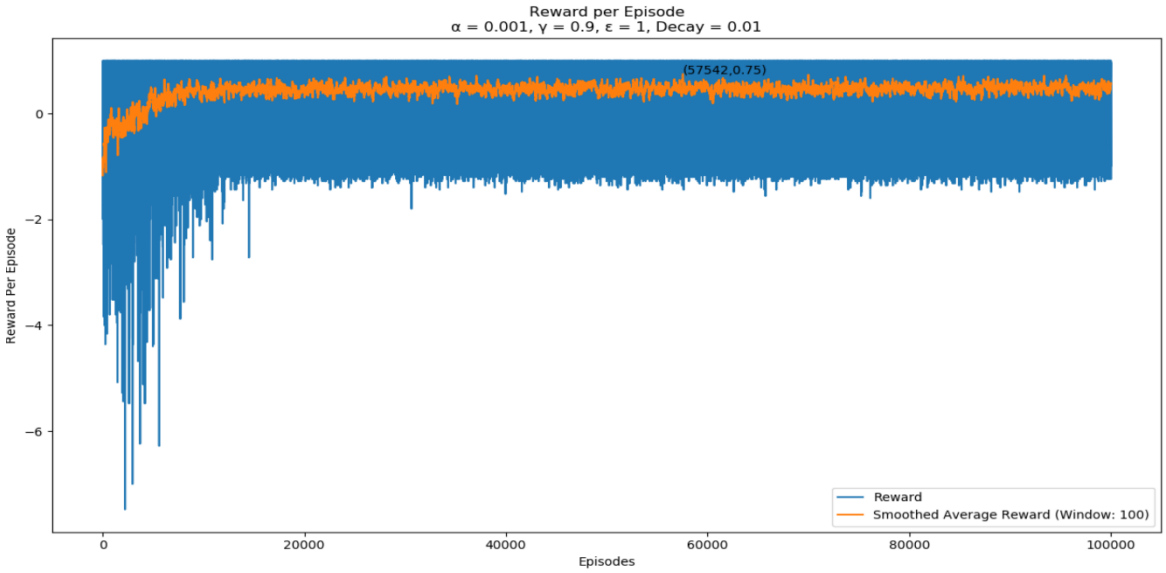


Figure 20 - Q-Learning - Decreased Learning Rate of 0.001

Despite this hyperparameter configuration resulted in the optimal policy, in order to better converge to the optimal q values retrieved in the 1st assignment, I expended the agent's exploration-based episodes period by reducing GLIE's decay factor from 0.01 to 0.001, so that the agent would explore more states and scenarios that will result in a much robust q-values that will affect the certainty of the agent's policy. In Fig. 21, the orange trend line of the smoothed reward is also continuously increased until it reaches a plateau after 10,000 episodes. In addition to a faster execution time of 10.5 minutes, in these hyperparameters settings, the trend is less noisy compared to the abovementioned trends.

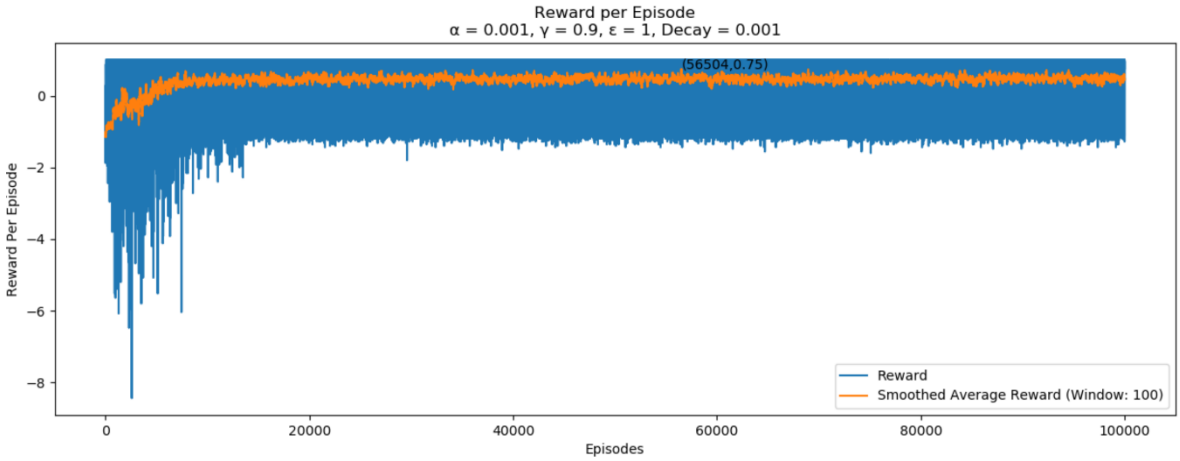


Figure 21 - Q-Learning - Decreased GLIE's Decay Rate

Moreover, a chart regarding the number of timesteps per episode is being displayed in Fig. 22. As can be seen, the number of timesteps per episode is exponentially decreased because of the $\varepsilon - greedy$ policy from which the exploration-exploitation ratio is being tuned and controlled by the decreased GLIE's decay rate of 0.001. Note the higher number of time steps in the first 20,000 episodes derived by the exploration phase in which the ε values are higher than those of the rest of the last episodes and therefore cause the probability for the agent to explore more states to increase during the first episodes.

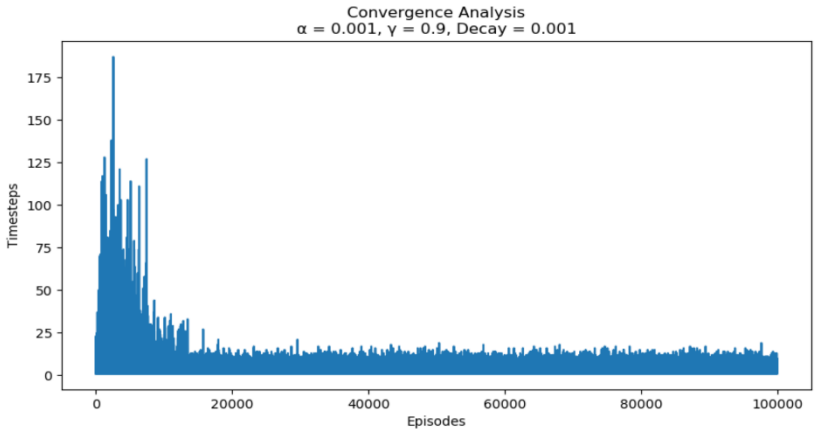


Figure 22 - Convergence Analysis of the decreased decay configuration

Furthermore, as appears in the state action plot of the q function values in Fig. 23, the agent also learned the optimal policy and the resulted $L_1\ norm$ distance was 0.055 which outperformed the rest of the abovementioned configurations regarding the Q-Learning algorithm. significantly closer to the q values achieved in the 1st assignment and the averaged reward achieved was 0.43. Also, note that even though the final configuration's outcome derived the optimal policy, and since the highest max smoothed average reward obtained in the 56504th episode, I explored and compared the $L_1\ norm$ of each one of the episodes in the interval in order to see if the agent learned the optimal policy in earlier episodes. As can be seen in Fig. 23 below, the lowest $L_1\ norm$ retrieved in the 66058th episode and was equal to 0.03 which outperformed substantially on the rest of the explored configurations of the Q-learning algorithm.

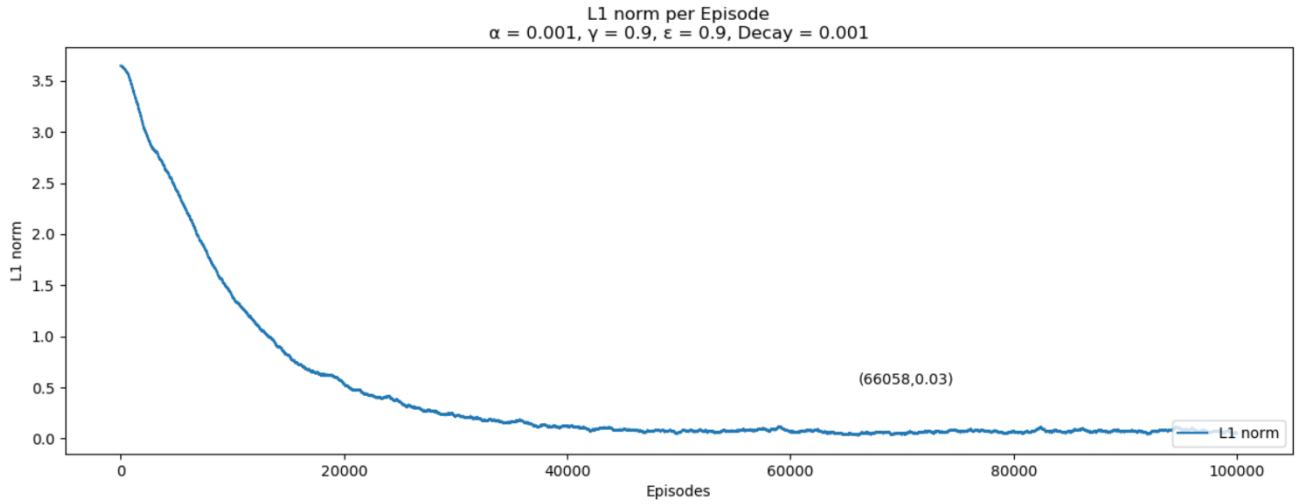


Figure 23 - $L_1\ norm$ Comparison between the episodes of the best found hyperparameters configuration

As can be seen in Fig. 24, based on the hyperparameter configuration of $\alpha = 0.001,\ \gamma = 0.9,\ \varepsilon = 1$ linear discount factor of 0.001 and 66058 episodes, the agent learned the optimal policy that derived by Q-values that are much closer to the Q-values achieved in the 1st assignment (Presented also in the code) and therefore outperformed the rest of the settings. Hence, from the agent starting state 4, the agent learned the quickest routes to its goal, both by turning east or going north, however by viewing the values for both routes the agent prefers to go north since it a much safer route than the east one and with fewer holes during the route. Further, in case the agent is located in state 8, the policy guides him to continue east despite the riskier road that is because it is much shorter than turning west to state 4 and then going up north. Note the small difference between the best state values retrieved in the 1st MDP assignment and presented in Fig. 17 to the best state values (green triangles) derived from the q-learning algorithm.

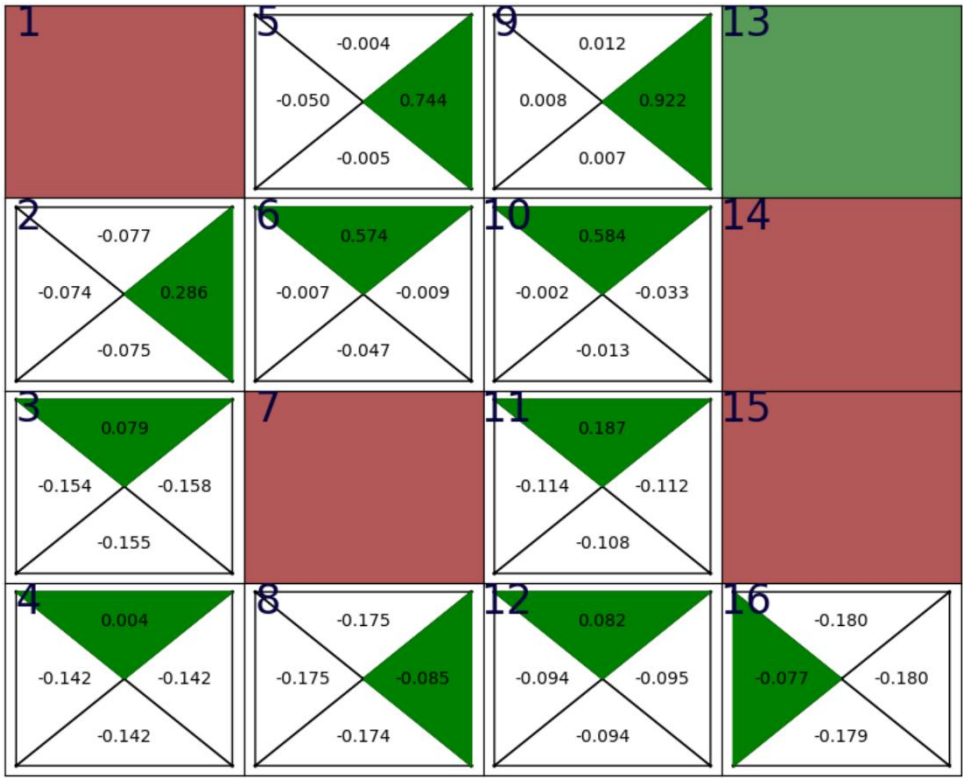


Figure 24 - Best Q-function values and the agent's learned optimal policy

4. Reference

[1] R. S. Sutton and A. G. Barto, “Reinforcement Learning: An Introduction Second edition, in progress,” 2018.

[2] S. Singh, T. Jaakkola, and M. L. Littman, “Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms,” 2000.

[3] Zong Woo Geem, Joong Hoon Kim, and G. V. Loganathan, “A New Heuristic Optimization Algorithm: Harmony Search,” *Simulation*, vol. 76, no. 2, pp. 60–68, Feb. 2001.