Deep Reinforcement Learning

Exercise 2

Student: Arseni Pertzovskiy, ID: 317377372.

1. Introduction

In this assignment we asked to implement two RL algorithms: SARSA and Q-learning.
Q-Learning is an Off-Policy algorithm for Temporal Difference learning. It learns the optimal policy even when actions are selected according to a more exploratory or even random policy. The SARSA algorithm is an On-Policy algorithm for TD-Learning. The major difference between it and Q-Learning, is that the maximum reward for the next state is not necessarily used for updating the Q-values. Instead, a new action, and therefore reward, is selected using the same policy that determined the original action.

2. SARSA & Q-learning Implementation

Both algorithms share the same structure of the code exept the update part.

SARSA update:

$$Q(St, At) \leftarrow Q(St, At) + \alpha(Rt{+}1 + \gamma Q(St{+}1, At{+}1) - Q(St, At))$$

Q-learning update:

$$Q(St, At) \leftarrow Q(St, At) + \alpha(Rt{+}1 + \gamma \max_a Q(St{+}1, a) - Q(St, At))$$

That is why, in order not to duplicate the code, I created common method to both algorithms called: *loop_of_algorithm(num_episodes, alpha, GLIE, gamma, update_func).*
This method recieves un *update_func* input that determins the specific update function for chosen algorithm.

3. GLIE Choice (process & results)

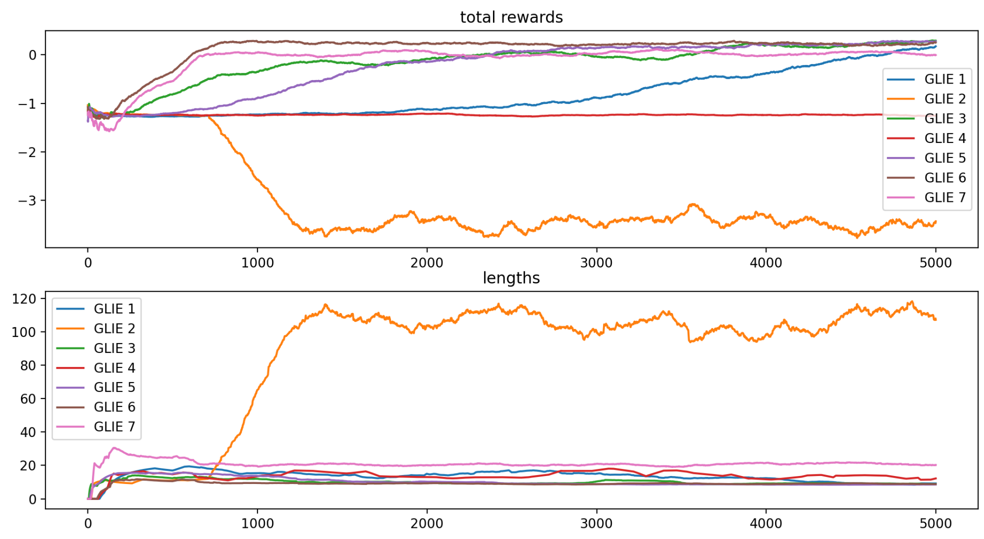
In order to get the best pair of GLIE-alpha paramenters I did the following: (1) run 5000 iterations on each GLIE option, (2) pick the best GLIE, (3) run 5000 iterations on each alpha option, (4) pick the best alpha, repeat steps (1) to (4) with chosen options from previous step several times.

All the GLIE functions represented in *main.py*. The options that I picked to check are:

- GLIE 1: $\varepsilon_i = 1 - \frac{episode}{amount\ of\ episodes}$
- GLIE 2: $\varepsilon_i = 1 - \frac{e^{episode}}{e^{amount\ of\ episodes}}$
- GLIE 3: $\varepsilon_i = 1 - \frac{\log(episode)}{\log(amount\ of\ episodes)}$
- GLIE 4: $\varepsilon_i = \max(\min\_epsilon, \min(1,\ 1 - \log(episode/25)))$
- GLIE 5: $\varepsilon_i = 0.999^{episode}$
- GLIE 6: $\varepsilon_i = 0.99^{episode}$
- GLIE 7: $\varepsilon_i = 0.9^{episode}$

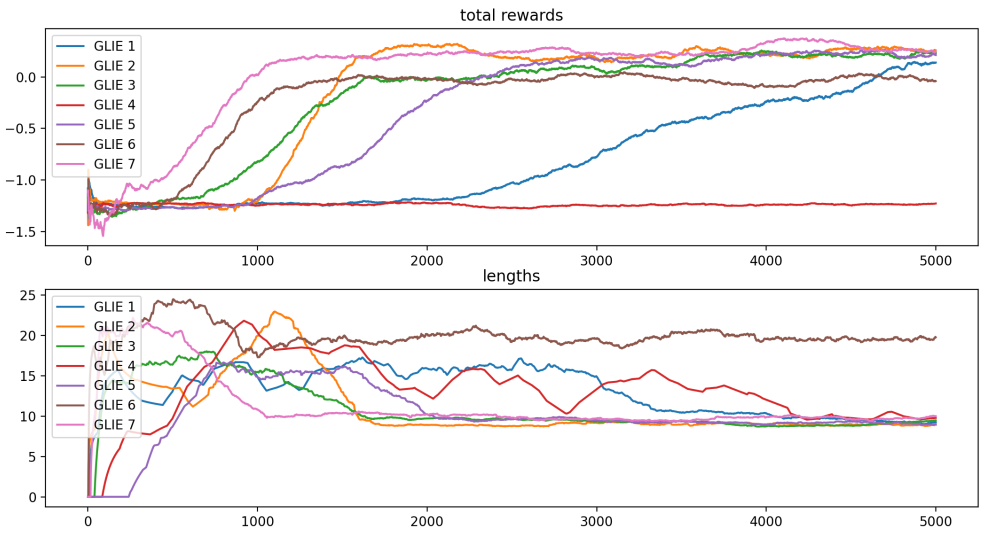
Next I'll represent the final results per each algoritnm.

For SARSA:



According to the graph the best choice will be GLIE_6 function both from total_rewards view (we want as high as possible) and length of the path view (we want as small as possible – get to the end quickly).

For Q-learning:



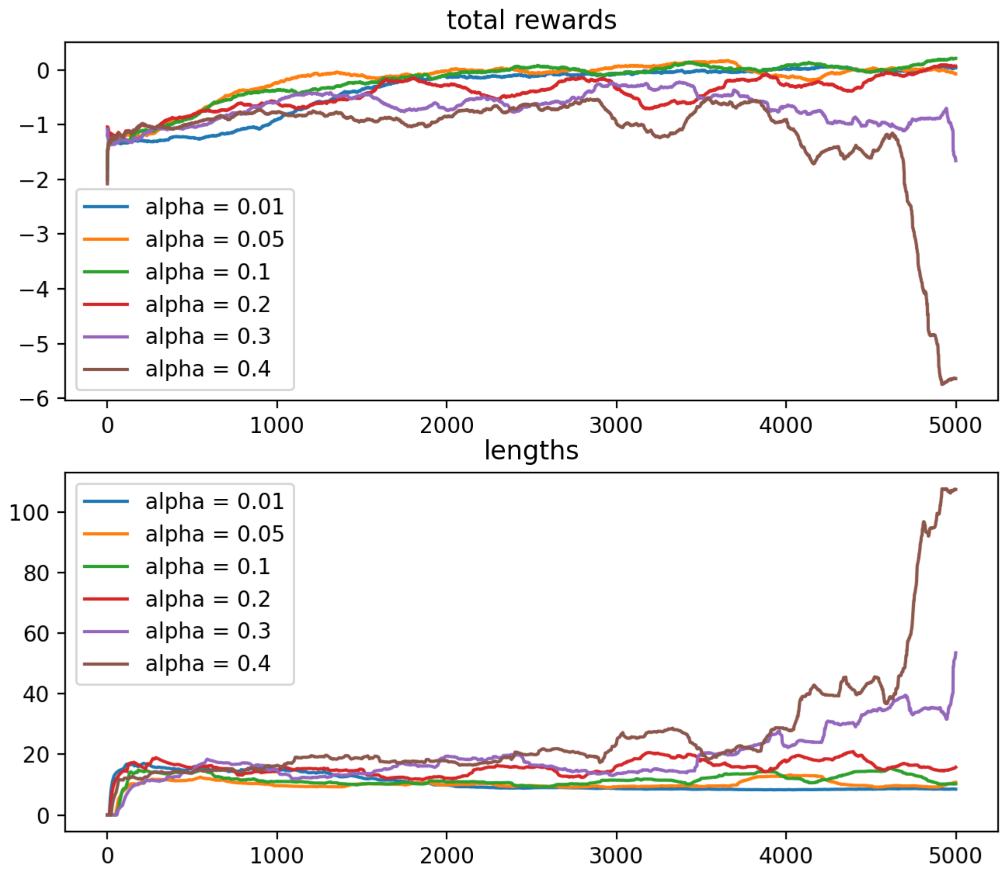
According to the graph the best choice will be GLIE_2 and GLIE_7 functions both from total_rewards view (we want as high as possible) and length of the path view (we want as small as possible – get to the end quickly). We have to pick only one, so I decided to stay with GLIE_2 beacause it seems to be more stable.

4. Alpha Choice (process & results)

As I described earlier, in order to get the best pair of GLIE-alpha paramenters I did the following proccess: (1) run 5000 iterations on each GLIE option, (2) pick the best GLIE, (3) run 5000 iterations on each alpha option, (4) pick the best alpha, repeat steps (1) to (4) with chosen options from previous step several times.

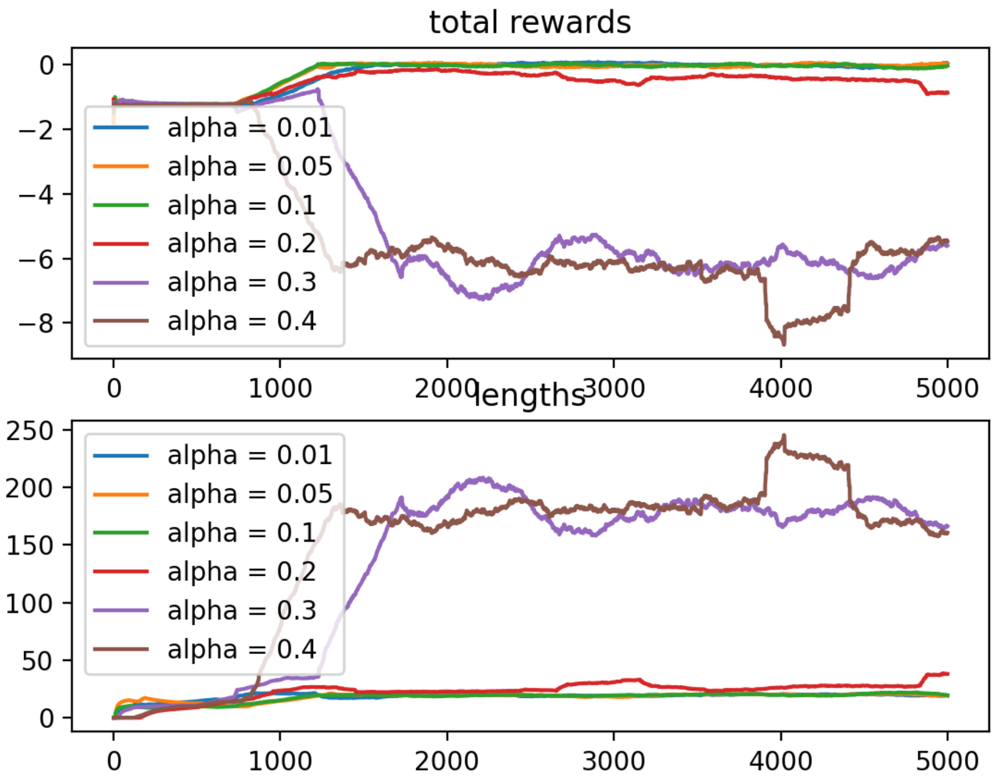
The alphas that I checked are: [0.01, 0.05, 0.1, 0.2, 0.3, 0.4]. From the smallest ones to the biggest ones. And the following results are:

For SARSA:



Total rewards - we want as high as possible.
Length of the path - we want as small as possible (get to the goal quickly).
According to the graph the best choice will be somewhere between 0.01 and 0.1. We have to pick only one, so I decided to stay with 0.05 right in the middle.

For Q-learning:



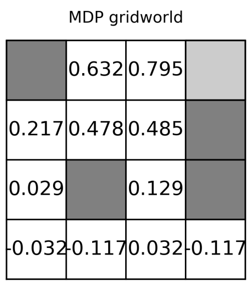
Total rewards - we want as high as possible.
Length of the path - we want as small as possible (get to the goal quickly).
According to the graph the best choice will be also somewhere between 0.01 and 0.1. We have to pick only one, and this time I decided to stay with 0.01 because mostly this alpha gave good and stable results.

5. Run-time choice (number of iterations)

I tried different approaches with 100 – 1000 – 5000 – 10000 – 100000 iterations. What I found is that after 2000 iterations the policy itself (which action to pick at which cell) converges. With more iterations the values themselves become more accurate (closer to the real MDP). The bottom lime, it is save enough to say that 2000 iterations are sufficient to this problem.

6. *plot_actionValues* function and the comparison to values from the last assignment

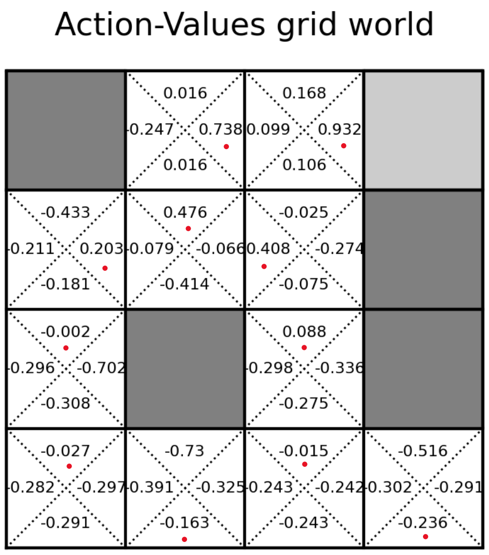
Values of cells from the previous assignment:


MDP gridworld

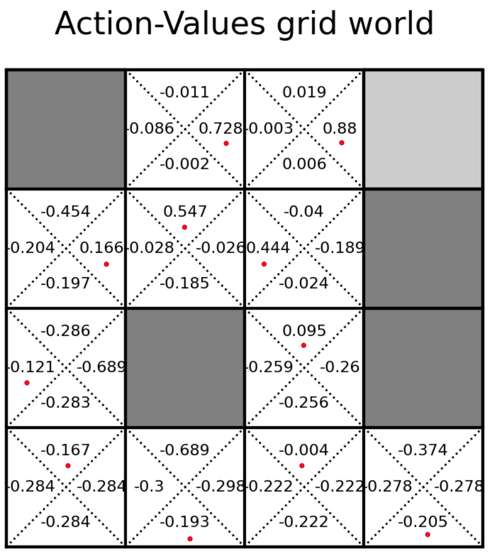
	0.632	0.795	
0.217	0.478	0.485	
0.029		0.129	
-0.032	-0.117	0.032	-0.117

plot_actionValues function outputs: we took 100000 iterations in order to get the values of each state-action pair, and the results…

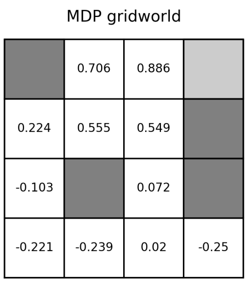
From SARSA:



From Q-learning:



In the current assignment we are working with a model-free approach where we don't know the underlying probabilistic function. Therefore, we are using sampling approach (Monte-Carlo) in order to get those underlying values and to build our policy. That is why the end results may differ from the original DP or value/policy iteration results where the probabilistic function is known.
I used *policy_evaluation* (inside *World.py* file) function to check the values per each cell, and the result is:

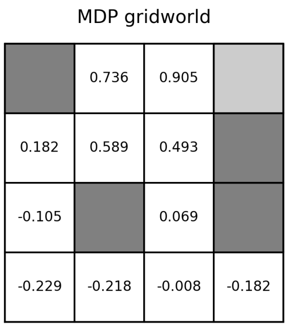

MDP gridworld

■	0.706	0.886	░
0.224	0.555	0.549	■
-0.103	■	0.072	■
-0.221	-0.239	0.02	-0.25

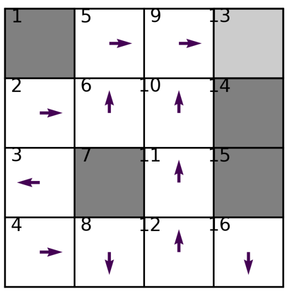
The start point was a cell number 4, as the *reset* method dictates. The amount of iterations were also limited due to requirements of the task. For all that reasons the values a little bit far from the original ones (previous assignment), but the dendency is similar, so we are on the right track.

In case we want to check the maximum value of each cell and to look at the policy, here are the plots of *plot_values* and *plot_policy* methods:

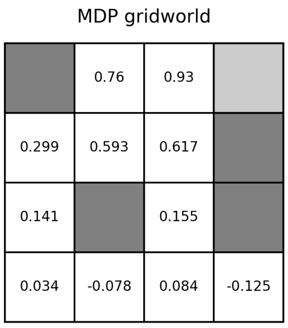
For SARSA:


MDP gridworld

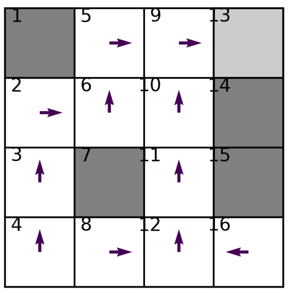
■	0.736	0.905	░
0.182	0.589	0.493	■
-0.105	■	0.069	■
-0.229	-0.218	-0.008	-0.182



For Q-learning:


MDP gridworld

■	0.76	0.93	░
0.299	0.593	0.617	■
0.141	■	0.155	■
0.034	-0.078	0.084	-0.125



As it is expected to be we can see that SARSA algorithm gives us more guarded ('save') policy than more hurried Q-learning algorithm. It can be described by the fact Q-learning is off-policy and is looking only at those actions that give him the maximum values. SARSA, on the other hand, can differ between 'bad' and 'good' action better (on-policy aspect). That is why, for example, in cell 3 SARSA prefers to escape completely from cell 7 at the cost of the solution, but Q-learning prefers to get to the goal faster and it is willing to take the risk.

7. Conclusion (final settings of our algorithms)

The final settings that I picked to go with are:

For SARSA:
Number of episodes: 2000, alpha: 0.05, GLIE function: *GLIE_6*, gamma: 0.9

For Q-learning:
Number of episodes: 2000, alpha: 0.01, GLIE function: *GLIE_2*, gamma: 0.9