RL211 - HW2

Omri Himelbrand - 200863843
Nitzan Cohen - 203980750

December 2, 2020

Contents

1 Introduction

The solution is implemented in a way that makes it possible to run it as a script or as a module.
The code will explore the environment using SARSA with ϵ-greedy.
We initialized Q to be 1 for all states that are not terminating, in order to help with convergence, for the same reason we also used decaying ϵ value.
At the end of the run we create a lot of plots with different spacing and compare different sub-sets of hyper-params.
In order to run this code you must have: **numpy** installed, and have an out directory for the output files.

2 Running the solution

2.1 Running as a script

```
usage: python hw2.py [-h] [-human] [-gamma G] [-d] [-ms MAX_STEPS] [-png PNG_SUFFIX]
                     [-4x4]

AI agent using SARSA lambda for AI-Gym Frozen lake.

optional arguments:
  -h, --help       show this help message and exit
  -human           use this flag to run human agent
  -gamma G         a float for gamma in [0,1] (default: 0.95).
  -d               use this flag to get debug prints
  -ms MAX_STEPS    a int for number of steps between evaluations.
  -png PNG_SUFFIX  a suffix for png out file
  -4x4             use this flag to use 4x4 map
```

2.2 Running as module

```
import hw2

hw2.main(gamma=0.95)
```

Running with wrapper - setting globals

```
import hw2

#setting globals
hw2.set_debug(True)
hw2.use_small_map(True)#this uses the 4x4 map

hw2.main(gamma=0.95,human=False)#set human to True to run as human agent
```

3 Code details

3.1 Global variables

DEBUG: A boolean which is initialized to False.
MAX_STEPS: An integer to indicate the number of steps between running evaluations simulations, initialized to 1000.
MAP: A string indicating the map is, initialized to '8x8' .
terminating_states: The lists of terminating states.

```
#terminating states init - by the deterministic map setting for holes and goal
terminating_states = {'4x4':[15,5,7,11,12],'8x8':[63,59,54,52,49,46,42,41,35,29,19]}
```

3.2 The functions

`init_env(max_steps=250):`

Calls gym.make, sets the max episode steps for the created environment and returns it.

`set_debug(value):`

Sets the global variable DEBUG to value.

`set_max_steps(value):`

Sets the global variable MAX_STEPS to value.

`use_small_map(value):`

Sets the global variable MAP to '4x4' if value is True.

`human_agent(env):`

Prompts user to pick action for the next step of simulation, used for mostly for debugging.
Return value is a action (int).

`evaluate(env,pi,gamma,episodes_num=750):`

Given the current policy (pi), and the rest of the arguments seen in the signature, evaluates v_0^π, using a MC-like evaluation.
This function returns value v_0^π.

`eps_greedy(eps,pi,q,states,actions):`

This function changes π in place according to the ϵ-greedy scheme, using randomized argmax for better exploration.

`sarsa(env,Q,pi,gamma,Lambda,alpha,states,actions,eps,explored,max_step=5000,\`
`episode_max_steps=250,iters=0,epsilon_decay=0.99999,min_eps=0.1):`

This function does SARSA, updating the Q values and π in-place, using decaying ϵ, with extra decay according to RBED (Reward Based Epsilon Decay). The decay is stopped at a minimal value for ϵ in order to enable some exploration after a long time. The

`run_simulation(env,policy=None,human=False):`

resets env to initial state, then runs simulation either using the given policy or using a human agent.

`learn_policy(env,actions,states,gamma,Lambda,alpha):`

This function runs the whole learning process, running sarsa for MAX_STEPS, then running evaluation.

After each evaluation the V_0^π is saved with the number of total steps taken so far, and we keep track of the best π according to the evaluations so far, keeping it for return.

The return value of this function is: x - array of step counts, y - array of V_0^π collected, and the best π.

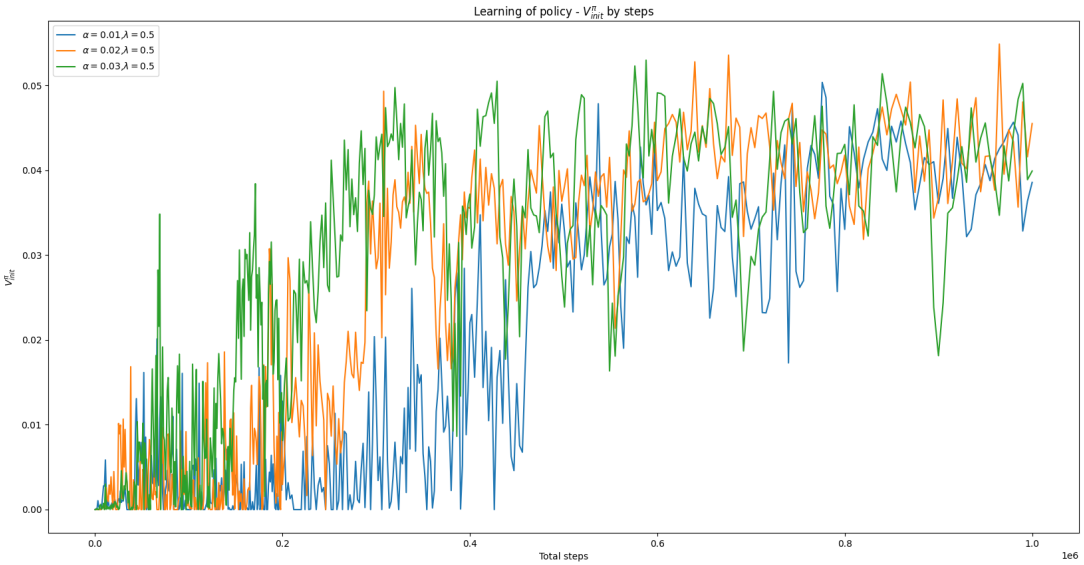
`main(gamma=0.95,human=False):`

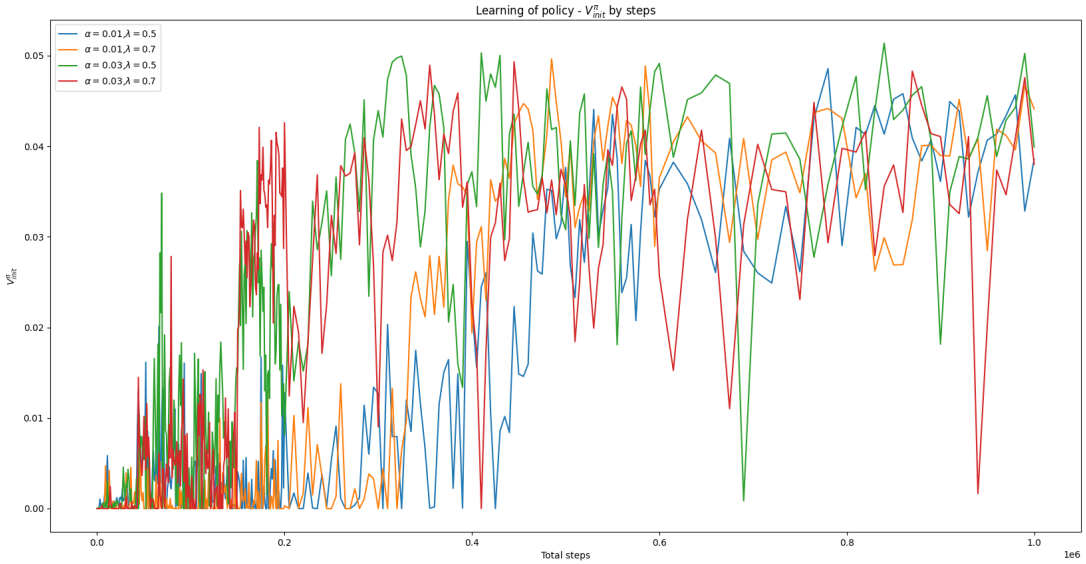
This function is called when running the code as a script, but can be used as seen above, this function does the following:

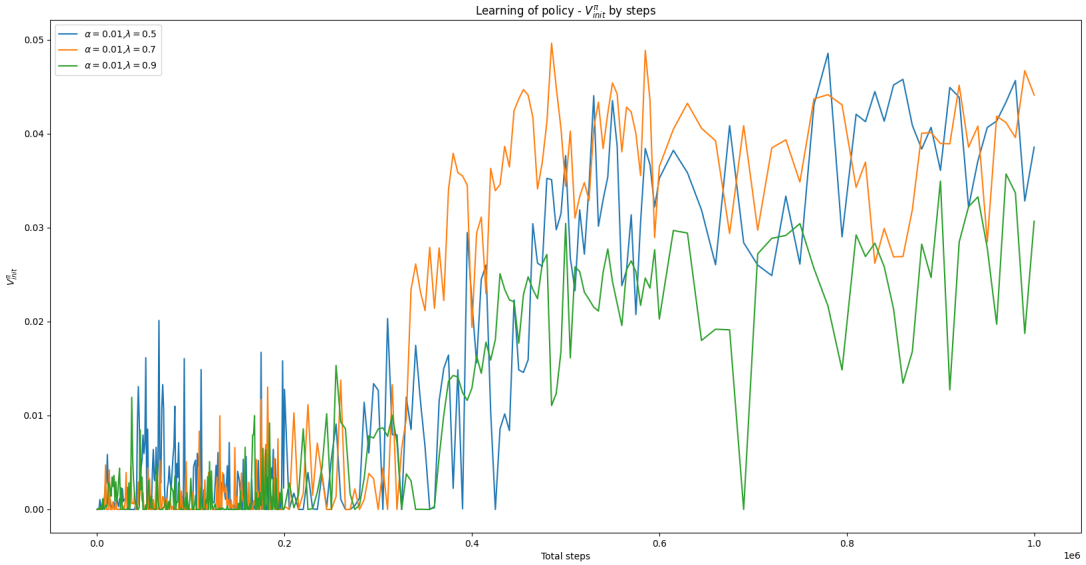
- calls "init_env"
- calls "explore_env"
- if human flag is set, runs a single simulation with a human agent.
- else calls "learn_policy" for each combination of lambdas and alphas we set.
- for each run, calls "run_simulation" using the returned pi
- Also for each run, saves the x and y values returned with key-label
- After running all runs and collecting all the x and y values, calls plot_results function

4 Plots

The plot_results function creates TONS of plots that we then examine and picked some interesting ones to present next:







We presented some of the plots, that seemed interesting and informative to us, comparing both λs and αs.