CS 747 - Foundations of Intelligent and Learning Agents
Assignment 1

Gagan Jain
180100043

November 12, 2020

Task 1: Creating the Environment

A (4×8) grid is modeling as a MDP with given start and end states as well as the wind details. There are 4 actions for the normal grid and 8 actions for king grid. The update is made taking into account the winds. For a move from a location which had non-zero wind, the resultant state is that much above the goal for that move. For the states near the edges, any attempt to go beyond does not change the state. This has been taken care of in the normal as well as the stochastic wind scenario.

The reward for every transition is taken to be -1 except for the goal state, which has 0 reward. The idea is to accumulate as less negative reward as possible which will drive the agent to reach the goal position as soon as possible. All the transitions are deterministic for the initial two cases and stochastic for windy regions for the stochastic world. I have an important observation to make here. The way the rewards are defined matter to a great extent and change the plots as well. A "-1" reward for the end state would have given a slower performing agent. Increasing the goal reward leads to increased speed, but the policy fluctuations also increase. For a reward of say "+100", the graphs are (almost) flat for some regions and very rapid for other. This means that the agent takes a greater time in exploring the entire grid. Another point to be made here is that when I say that the end state has 0 reward, it means that the rewards are defined on the basis of the state they are reaching in (one of the variants discussed in the lectures).

I have also created a visual representation of the scenario which can be used to visualise the transitions happening as the agent takes actions. Note that this being a GUI, this visualisation cannot be seen on docker unless GUI settings are done on docker. Outside docker, it works perfectly.

Here is a snippet of the visualisation of the agent in its environment. The increasing shades of blue indicate the greater wind intensity, the yellow is the agent and the green is the goal state.

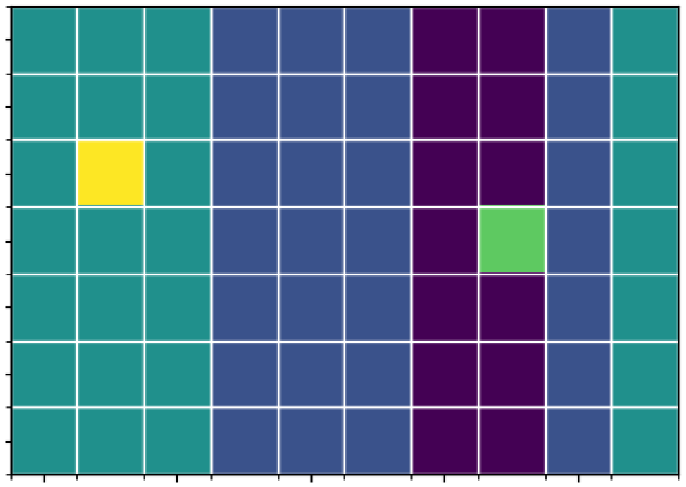

Figure 1: Visualisation: Agent in action

Task 2: Baseline Plot

The SARSA Algorithm is used to find the optimal path to be followed in the windy gridworld environment. The parameters used are:

- Learning rate $(\alpha) = 0.5$
- Exploration $(\epsilon) = 0.1$
- Discount $(\gamma) = 1$
- Number of episodes = 200

The following plot shows the performance of the algorithm for the normal environment. The performance is averaged over 10 random seeds. The averaging is done for plotting in all the subsequent sections as well. The plot obtained is very similar to the plot given in the book by Sutton and Barto.

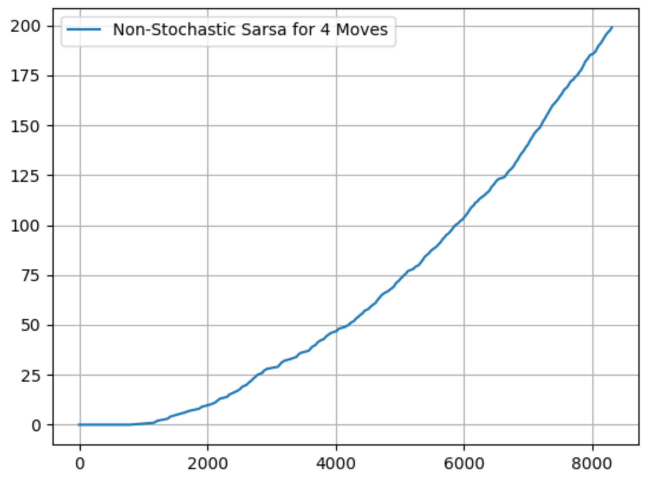

Figure 2: Non-stochastic gridworld with 4 actions - SARSA

While experimenting with the parameters and observing the results, I observed that the policy is very close to optimal after certain number of steps (around 100 for this case). After that, keeping $\epsilon = 0.1$ is too much exploration. Instead, we can decay the value as inversely proportional to the number of episodes completed, keeping it constant at 0.1 for some number of time steps (say 100). Mathematically, $\epsilon = min(0.1, 10/E)$ should perform better than the current values and it indeed does. Most of the times in the later time steps, it goes by the optimal route which takes only 15 timesteps per episode.

Task 3: King's Moves Plot

The only change here is that now diagonal moves are also allowed. Here is the plot obtained for 200 episodes considering 8 allowed actions. All the parameters have been kept same as mentioned in Task 2.

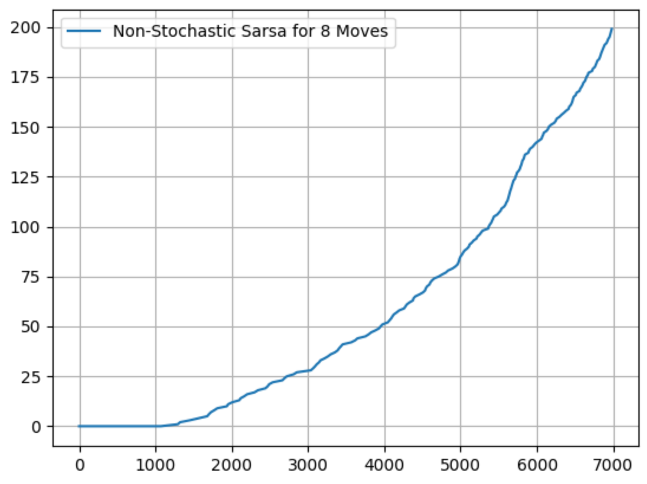

Figure 3: Non-stochastic gridworld with 8 actions - SARSA

The thing to be noted here is that while the previous setting required over 8000 timesteps to reach 200 episodes, this setting does the same task in approximately 7000 timesteps. This is because of the increasing capability of the agent. In fact, the optimal path requires only 7 steps to reach to the goal. Again, the decaying exploration rate is something that should be tried as it does much better as compared to this for the same task.

Task 4: Stochastic King's Moves Plot

Now, the wind has a stochasticity of magnitude 1 wherever it is present. This results in more uncertain outcomes and thus finding the optimal way out is more difficult. We again work with the 8 moves setting and the same set of parameters. Here is the plot which we get.

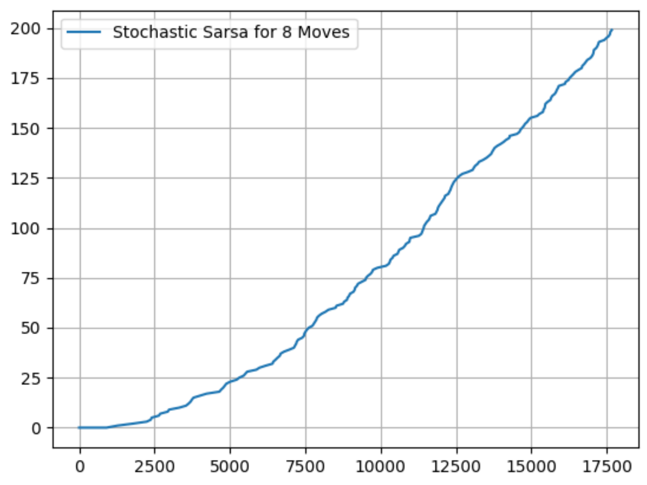

Figure 4: Stochastic gridworld with 8 actions - SARSA

It can be easily observed that the agent takes around 17500 timesteps to reach 200 episodes. This is not unexpected as the agent needs to learn to adapt to the uncertainity that is present in the environment. Another point to be noted is the distinctly non-smooth behaviour of the graph which is again due to the randomness. This is the place where the need for exploration is more. In the previous scenarios, the environment behaviour had to be learnt but once that is done, the agent always knows fixed path(s) which are optimal. Here, the uncertain behaviour of the environment requires the agent to behave a little more conservatively rather than sticking to the same plan always.

Task 5: Comparing SARSA, Q-learning and Expected SARSA

Now, we again go back to the initial environment settings and compare the performance of three algorithms - SARSA, Q-learning and Expected SARSA. We have already seen the performance of SARSA on this environment. The following plot compares helps us see the performance of these algorithms for the same set of parameters.

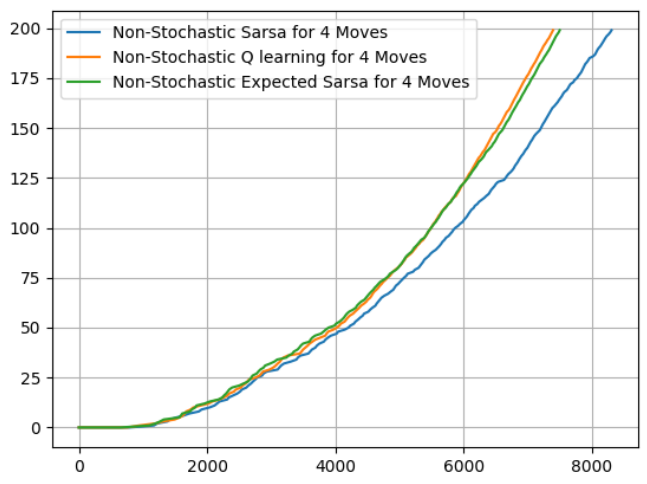

Figure 5: Non-Stochastic gridworld with 4 actions - SARSA, Q-learning, Expected Sarsa

From the plot, we can see that the Expected Sarsa and Q-learning perform significantly better than the Sarsa algorithm. We can observe that the Sarsa algorithm is different from the other two in the sense that it incorporates the next state and next action in the update rule, whereas the other two respectively take the expectation and maximum over all actions. In a way, the latter two do something which is in a sense similar and driving towards the optimal thing in a similar sense. So, this can justify the very similar behaviour of the two. These two do a more sensible thing than SARSA, which has double stochasticty, and thus can be expected to work better.

For a quick comparision and validation of my arguments, here is a plot when the exploration rate is decayed after 100 episodes. Note the increase in performance as well as smoothness of the plots. The performance of each algorithm is very close to optimal for the later timesteps.

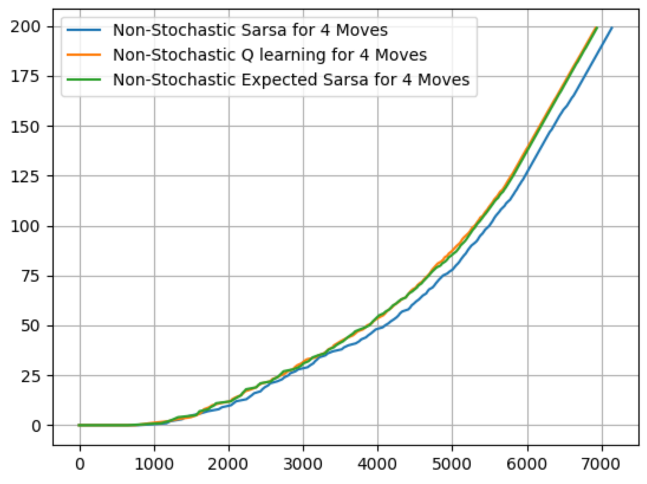

Figure 6: Non-Stochastic gridworld with 4 actions - SARSA, Q-learning, Expected Sarsa - Decaying epsilon

Code Structure and Explanation

The submission directory contains three folders - *report*, *codes* and *plots*. The *report* folder contains this report. The *plots* folder, by default, contains these four plots which have been added to this report. The *codes* folder contains two files, *gridworld.py* and *run.sh*. The shell script *run.sh* can be used to generate and save the required plots to the *plots* folder. Note that this will replace the already existing plots, but those are same as the ones in this report.

The file *gridworld.py* contains the implementation for the algorithms, environment and visualisation. The command line parameters taken by the code are all optional. The default run without any parameters runs SARSA on the normal grid for 200 episodes for the same values of parameters which have been reported, averaged over 10 seeds. The description of the parameters are as follows:

- –environment (-env): by default, set to 'gridnormal'. The acceptable values are 'gridnormal' and 'gridking' for 4 and 8 action cases respectively.
- –stochasticity (-sto): by default, set to False. It can be set to true to enable the stochastic wind environment.
- –algorithm (-algo): by default, set to 'Sarsa'. Multiple algorithms can be called by mentioning them in a comma separated manner in the same string. The accepted elements are 'Sarsa', 'ExpectedSarsa', and 'Qlearning. For multiple algorithms, for example, one can call 'Sarsa,Qlearning' to call these two.
- –epsilon (-eps): set to 0.1 by default. Should be a floating point number in (0,1). This sets the amount of exploration.
- –alpha (-alpha): set to 0.5 by default. Should be a floating point number in (0,1). This sets the learning rate of the algorithms.
- –num_episodes (-num_epi): set to 200 by default. Should be a positive integer. This sets the number of episodes for which the algorithm(s) will run.
- –gamma (-gamma): Set by default to 1. This should be a floating point number in [0,1]. This sets the discount factor.
- –visualise (-vis): Set by default to 0. This should be a non-negative integer. This tells us the number of episodes for which the visualisation needs to be shown while running the algorithm. Note that this will NOT work in docker if GUI settings are not done. By default, there is no visualisation done. If it is set to 1 (in a GUI supported environment), the last episode will be visualised. For any number greater than 1, the episodes to be visualised are exponentially distributed in the range. For example, if the total number of episodes is 32 and this is set to 5, the episode numbers 2,4,8,16,32 will be visualised (in a 1-based indexing).
- –randomseed (-rndm): Set to 10 by default. Should be a positive integer. This tells the number of seeds over which the performance is to be averaged.

A terminal command using all these parameters is shown below: (it is a good idea to not average when you are visualising). Reiterating, while running on docker, -vis should be always set to 0.

```
$ python3 gridworld.py -env gridking -sto True -algo 'Sarsa,ExpectedSarsa,Qlearning' -eps 0.2 -alpha 0.4 -num_epi 100 -gamma 1 -vis 1 -rndm 1
```

Note that in order to generate the plots asked in the assignment, it is sufficient to run the script *run.sh*. For this, (assuming the current working directory is the submission directory):

```
$ cd codes
$ chmod +x run.sh
$ ./run.sh
```

The plots will now be saved to the *plots* folder.