Programming Assignment 2

R KAUSHIK — EE15B105

April 25, 2018

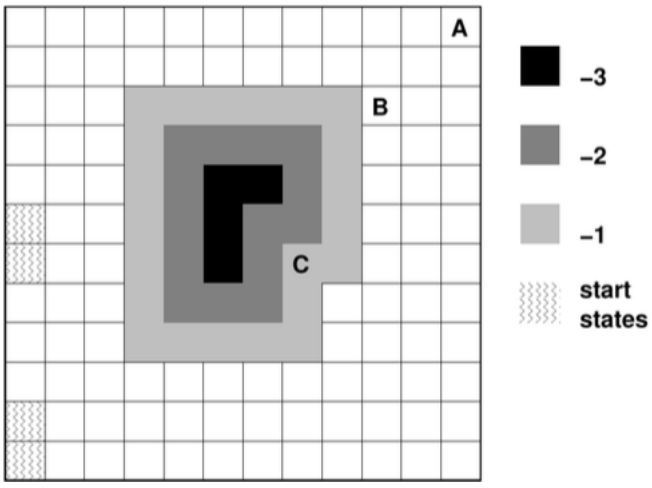


Figure 1: Puddle world Environment

1 Q1

Q-learning

1.1 When the terminal state is at position A

A reward of -0.5 was given for every step that didn't make the agent move to the goal or any of the puddles. If this negative reward was not given, the agent was found to converge to a sup-optimal policy where it kept moving to the top-left corner. More credit should be given to the wind blowing towards the west for this sub-optimal convergence.

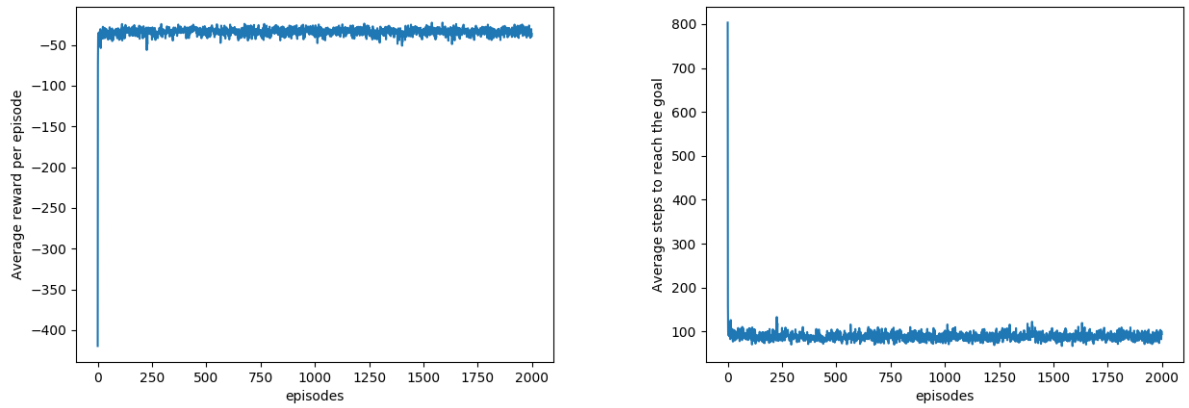


Figure 2: The left plot shows the variation of average reward over episodes. The right lot shows the variation of average number of steps taken to reach the goal per episode when the terminal state is at position A. The averages are taken over 50 independent runs

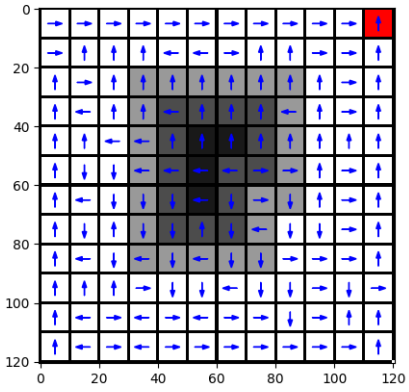


Figure 3: The optimal policy when terminal state is at A

1.2 When the terminal state is at position B

The environment is windy in this experiment too

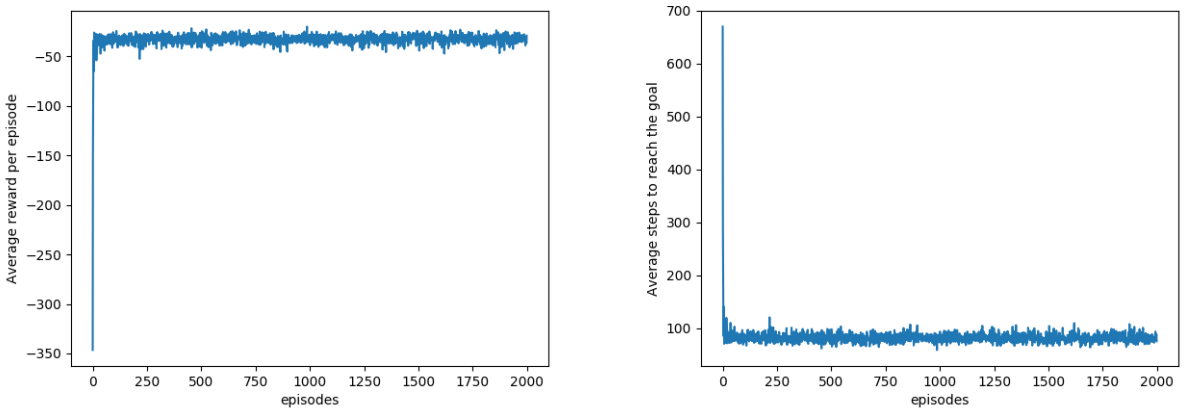


Figure 4: The left plot shows the variation of average reward over episodes. The right lot shows the variation of average number of steps taken to reach the goal per episode when the terminal state is at position B. The averages are taken over 50 independent runs

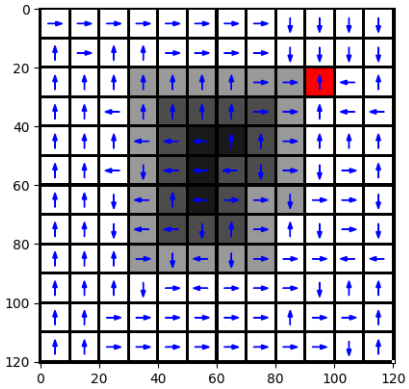


Figure 5: The optimal policy when terminal state is at B after 10000 episodes

1.3 When the terminal state is at position C

The wind is removed for this experiment

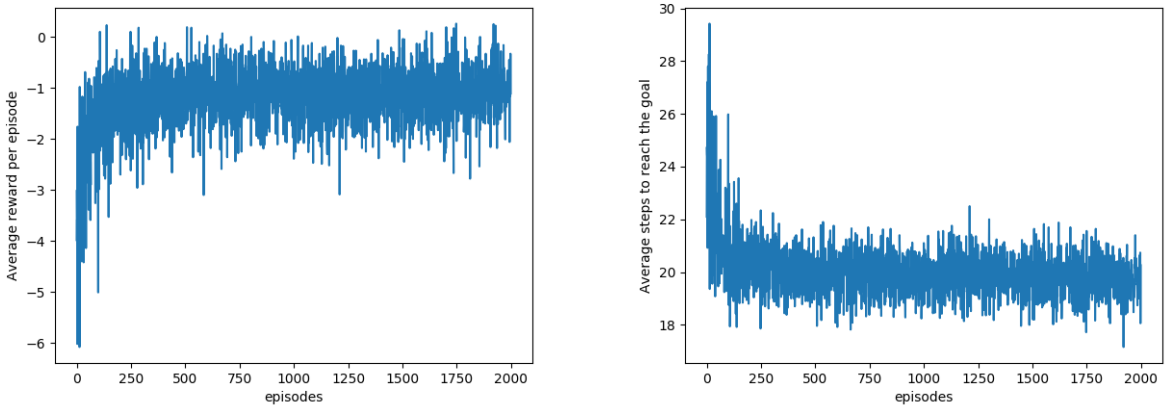


Figure 6: The left plot shows the variation of average reward over episodes. The right lot shows the variation of average number of steps taken to reach the goal per episode when the terminal state is at position C. The wind is turned off in this experiment though. The averages are taken over 50 independent runs

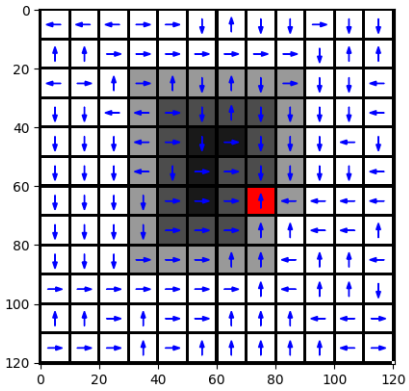


Figure 7: The optimal policy when terminal state is at C after 10000 episodes

SARSA

1.4 When terminal state is at A

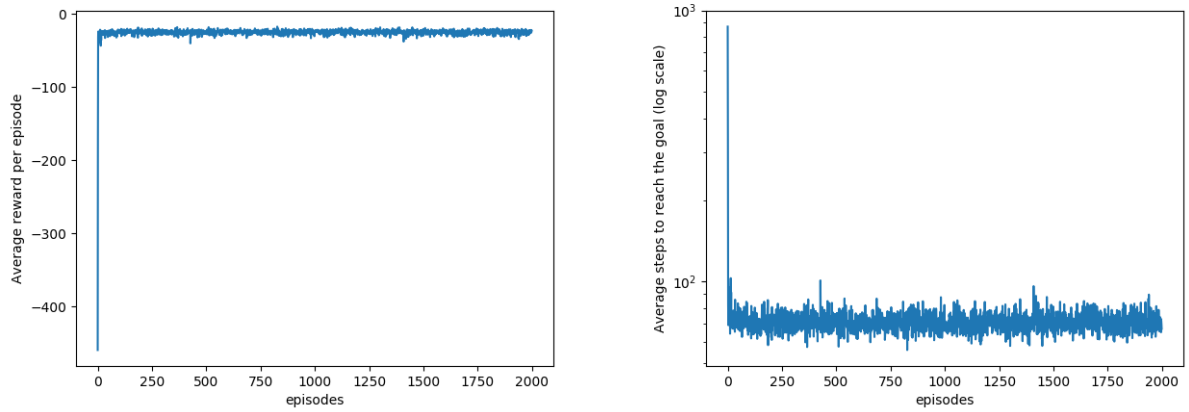


Figure 8: The left plot shows the variation of average reward over episodes. The right lot shows the variation of average number of steps taken to reach the goal per episode when the terminal state is at position A. The averages are taken over 50 independent runs

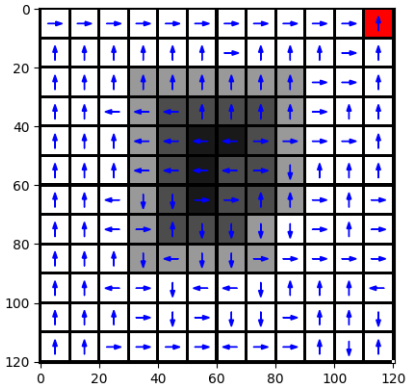


Figure 9: The optimal policy when terminal state is at A after 2000 episodes

1.5 When terminal state is at B

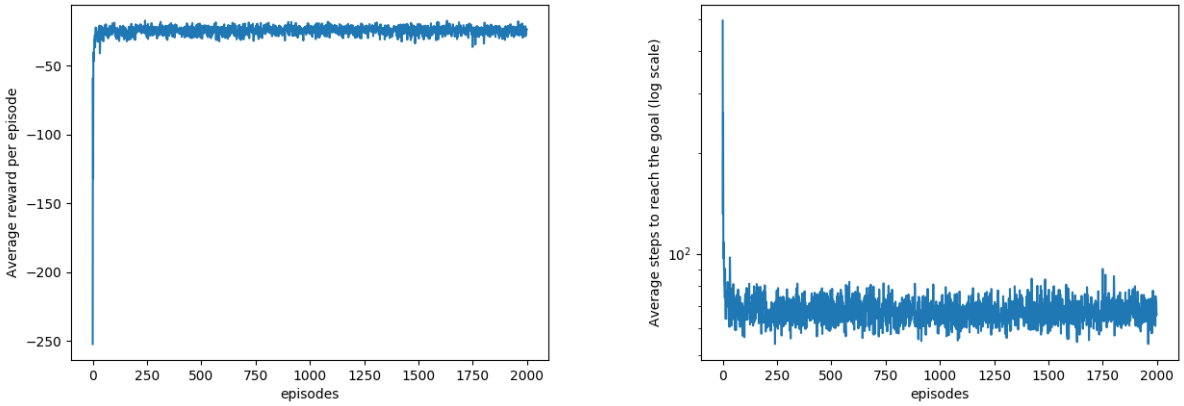


Figure 10: The left plot shows the variation of average reward over episodes. The right lot shows the variation of average number of steps taken to reach the goal per episode when the terminal state is at position A. The averages are taken over 50 independent runs

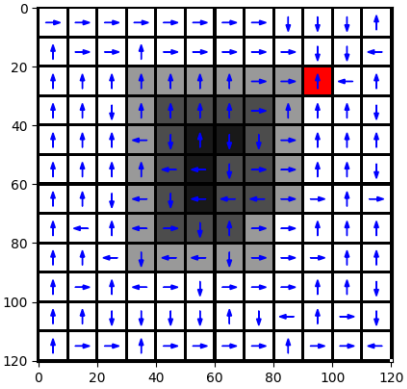


Figure 11: The optimal policy when terminal state is at B after 2000 episodes

1.6 When terminal state is at C

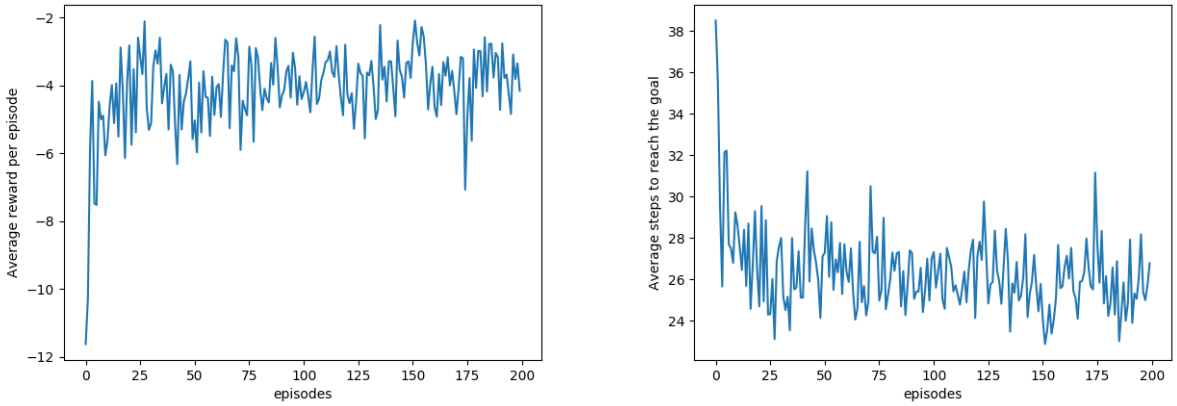


Figure 12: The left plot shows the variation of average reward over episodes. The right lot shows the variation of average number of steps taken to reach the goal per episode when the terminal state is at position A. The averages are taken over 50 independent runs

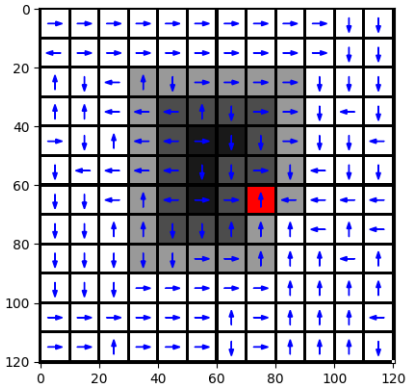
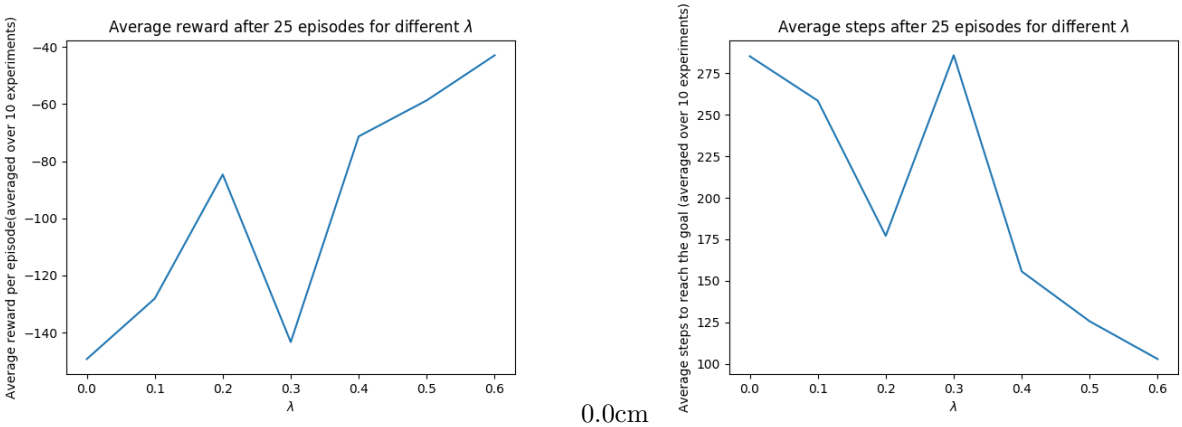


Figure 13: The optimal policy when terminal state is at C after 2000 episodes

1.7 SARSA(λ)



0.0cm

Figure 14: The plots show the performance of SARSA(λ) for $\lambda \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$

The backward view of eligibility trace was used to implement SARSA(λ). The algorithm was too slow for higher values of λ

2 Policy Gradient

Batch update policy gradient was implemented. The terminal state was defined as state where the distance of the agent from the origin is ≤ 0.05. A reward of $+2$ is given when the agent enters the terminal state to encourage the agent more. This didn't affect the performance much. Start state of first episode of each batch was randomly chosen. Each episode of a batch started from the same start state as the first episode of the same batch.
Policy gradient was tested for two environments with different reward structure. The results for two algorithms are presented together in the report.
We can see that the optimal policy for the *chakra* environment is the policy that tries to make the agent move towards the center. Hence the optimal policy parameter

$$\Theta = [[-1.0, 0, 0], [0, -1.0, 0]]$$

. The policy gradient algorihtm implemented was also observed to converge to this value for *chakra* environment.
For *vishamC* environment, the optimal policy parameters was found to converge to some value around

$$\Theta = [[-1.0, 0, 0], [0, -1.0, 0]]$$

after 150 iterations with batch size of 50 and $\gamma = 0.2$

The figure below shows the variation of average reward with iterations

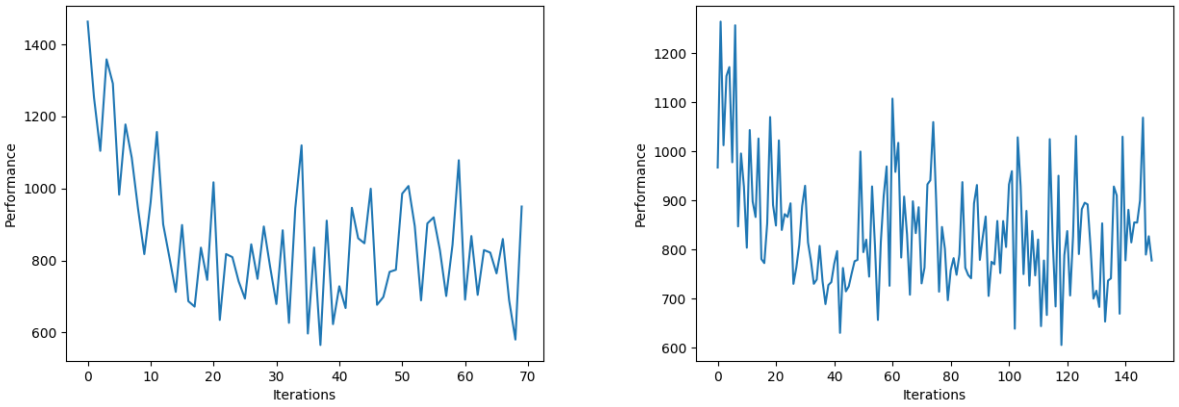


Figure 15: The left plot shows the variation of average reward over episodes in *chakra* environment with batch size = 50 run over 70 iterations and $\gamma = 0.2$. The right plot shows the variation of average number of steps taken to reach the goal per episode in *vishamC* environment with batch size = 50 run over 150 iterations.

2.1 Effect of varying hyperparameters

The following observations applies for both *chakra* and *vishamC* environment. But the examples given corresponds to *chakra* environment.

- **Batch size:** As batch size decreases, it takes more number of iterations to learn the policy that higher batch size agent learns when other hyperparameters were fixed. But in total the number of episodes needed remains same. For example the agent with batch size 10 took more than 173 iterations (1730 episodes) to learn the same policy that agent with batch size 30 did in 57 iterations (1710 episodes), around 32 iterations when batch size is 50 (1600 episodes), around 24 iterations when batch size is 70(1680 episodes) and around 18 iterations when batch size is 90 (1620 episodes). (A policy was said to be learned when the parameter values reached the expected optimal parameter values mentioned above)
- **learning-rate:** As learning rate increases, the solution reaches near the optimal policy faster but it keeps oscillating near the optimal value instead of converging.
- **discount factor:** When the discount factor was decreased, the optimal policy is found faster, in lesser number of episodes. For example with $\gamma = 0.9$ and batch size of 30, the agent took 57 iterations to arrive at the optimal policy. But with $\gamma = 0.5$ and same batch size, the agent found optimal policy in 41 iterations. The same trend was observed for $\gamma = 0.2$ too.

2.2 Value function visualization

The value function of a state is

$$V(s) = E[r|s] + \gamma \sum_{a} \pi(a|s) \sum_{s'} p(s'|s,a) V(s')$$

The transition probability given a state and an action is a unique state with probability 1. Therefore the value function formula reduces to

$$V(s) = E[r|s] + \gamma \sum_{s'} \pi'(s'|s) V(s') \tag{1}$$

$\pi'(s'|s)$ has same shape as $\pi(a|s)$. That is probability of choosing an action given a state is same as probability of choosing a next state given a current state since the every action from a state s leads to a unique next state s'.
$\pi(a|s)$ is a gaussian function centered at $a = -s$. So the function $\pi'(s'|s)$ is a gaussian function centered at $s' = (0, 0)$ for any starting state s. This means the second term of RHS of the equation (1) is same for any starting state s.

Expected reward, $E[r|s] = E[-\|s'\| \, |s]$ where $\|s'\|$ is the euclidean distance of state s' from origin and we know that the distribution of s' is gaussian centered at $(0, 0)$. Therefore Expected reward from a state s is same for all s.
This means the value functions for all the states is same.

2.3 Trajectory of policies

The trajectory was visualized assuming the mean value of the distribution of actions at a state as the actual action taken at that state. Following this, a radial trajectory was obtained for both the environments