Programming Assignment 3

Karan Agarwalla
180050045

Task1:

Class *gridWorld* implements Windy Gridworld as an episodic MDP. The constructor takes one argument: *gridParams* (a dictionary containing all information regarding the grid: numMoves, numRows, numCols etc.). The instance of class maintains the current state *currState*.

function *nextMove(action)*: Returns the next state and reward for given action and current state. The next state is calculated by first moving, possibly out of bounds, including wind and then truncating back to edge.

Task2:

Sarsa(0) agent is implemented in function *sarsaAgent()* in main.py. The function takes in arguments *epsilon*, *learning rate*, *episodes*, *gridWorld object* and *seed.* The algorithm in procedural form is presented below.

def *getAction*(Q, epsilon, currState):
    Returns ϵ-greedy action with respect to action value function, Q from current state *currState.*

Sarsa(0) Agent for estimation Q:

Parameters: learning rate α and epsilon ϵ
Initialise $Q(S, A)$=0 for $\forall S, A$
Loop for each episode:
    Initialise S = *grid*.startState
    Choose A from S using ϵ-greedy policy derived from Q
    Take action A, observe R, S'
    Loop until S' is not *grid*.endState:
        Choose A' from S' using ϵ-greedy policy derived from Q
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + Q(S',A') - Q(S,A)]$
        $S \leftarrow S', A \leftarrow A'$
        Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R - Q(S,A)]$ since $Q(S',A')$ *is* 0

Hyperparameters: $\epsilon = 0.1, \alpha = 0.5, episodes = 170, seeds = 50$
To obtain the plots use the bash-script $run.sh$

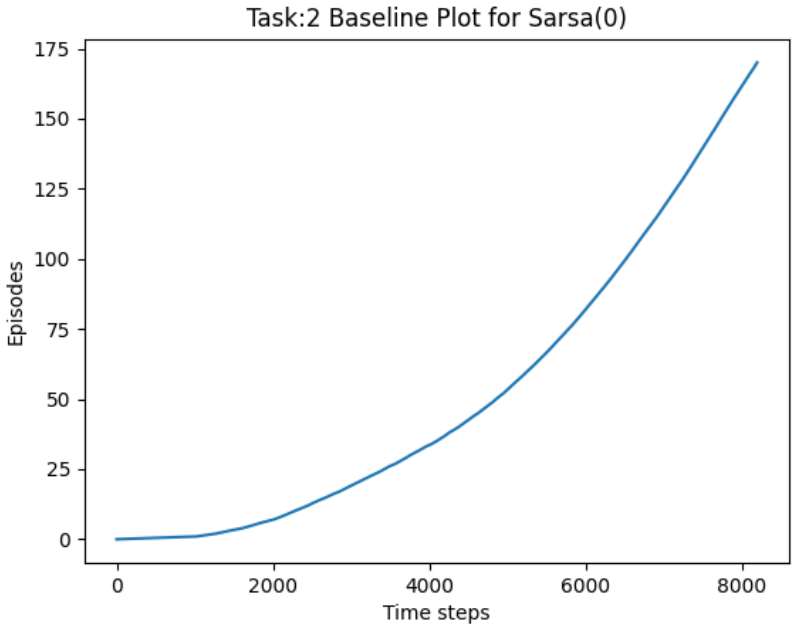


Figure 1: Baseline Plot for Sarsa(0)

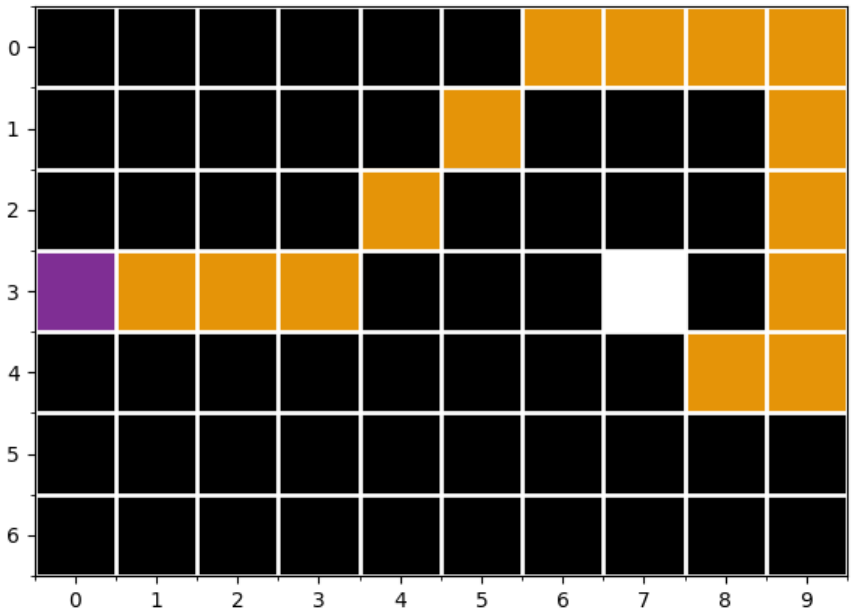


Figure 2 Path obtained for Standard Moves

The figure above shows the path obtained by greedily following the average Q across all seeds. One sees the path is indeed optimal in its length.

Task3:

The argument to *gridWorld* object is *gridParams,* which contains *numMoves* as an argument. In case of King's Moves, *numMoves* is set to 8. The moves are indexed as:
0: North, 1: East, 2: South, 3: West, 4: North East, 5: South East, 6: South West, 7: North West.

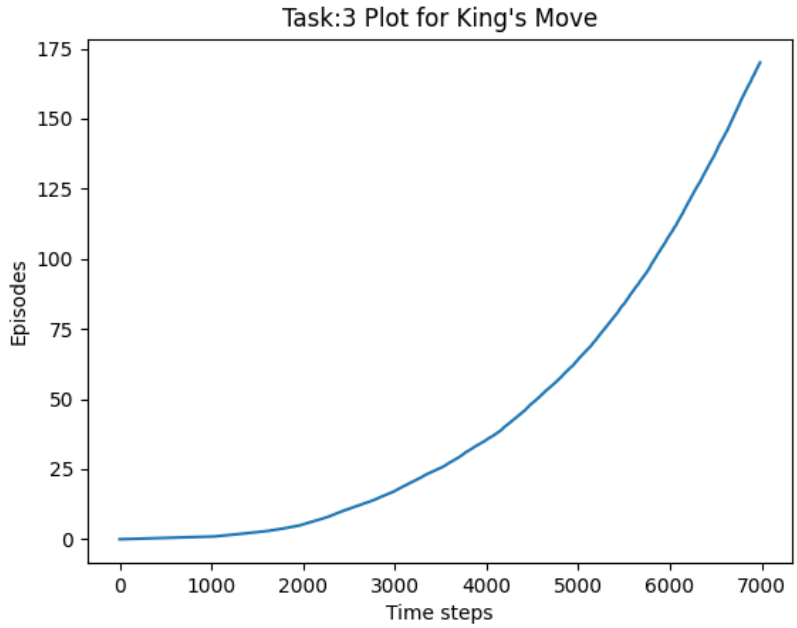


Figure 3 Plot for King's Move with Sarsa(0) and No Stochasticity

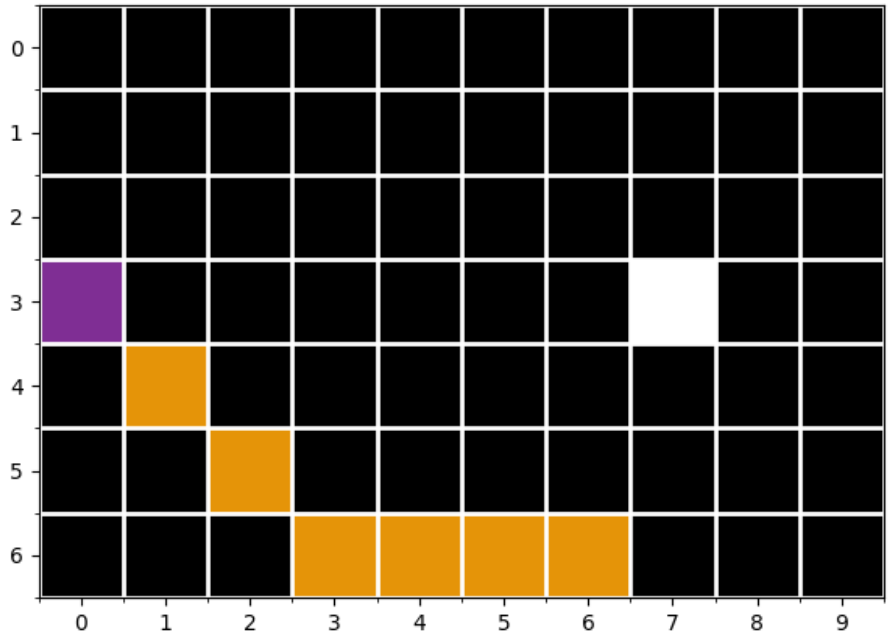


Figure 4 Path obtained for King's Moves

The figure above shows the path obtained by greedily following the average Q across all seeds. The path is optimal in its length. Also, we observe from the first curve that the timesteps needed for 170 episodes is less than that of 4-actions gridWorld. Note a stark difference in length of optimal path with King's Move enabled (7) and disabled (15).

Task4:

The *stochasticity* is a parameter(through *gridParams*) to the *gridWorld* constructor. The *windSpeed* is calculated as varying by 1 from mean values for each column(**if windStrength[self.curState[1]]>0**):

windSpeed = *self*.windStrength[*self*.currState[1]] + np.random.randint(-1, 2)

Figure 5 Plot for King's Move with Sarsa(0) with Stochasticity

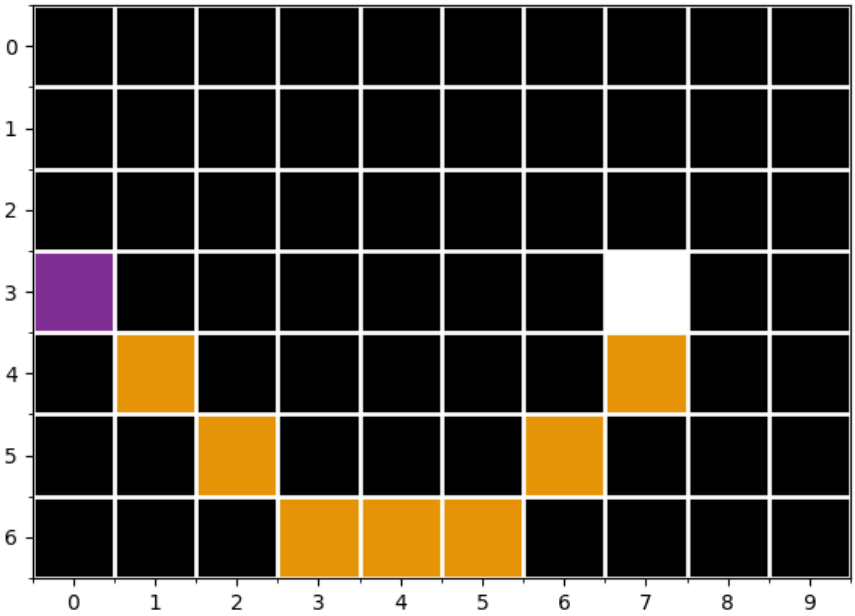


Figure 6 Path obtained for Stochastic King's Moves

The figure above shows the path obtained by greedily following the average Q across all seeds. The path is not optimal in its length as the shortest path is of length 7(See Figure 4: for stochasticity 0). However, the worst-case scenario path for stochastic case with value always equal to +1 is 14(Optimal Path: E, E, E, E, E, E, E, E, S, S, S, SW, SW).

Task5:

Q-Learning Agent for estimation Q:

Parameters: learning rate α and epsilon ϵ
Initialise $Q(S, A)$=0 for $\forall S, A$
Loop for each episode:
    Initialise S = *grid*.startState
    Loop until S is not *grid*.endState:
        Choose A from S using ϵ-greedy policy derived from Q
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \max_{A'} Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$

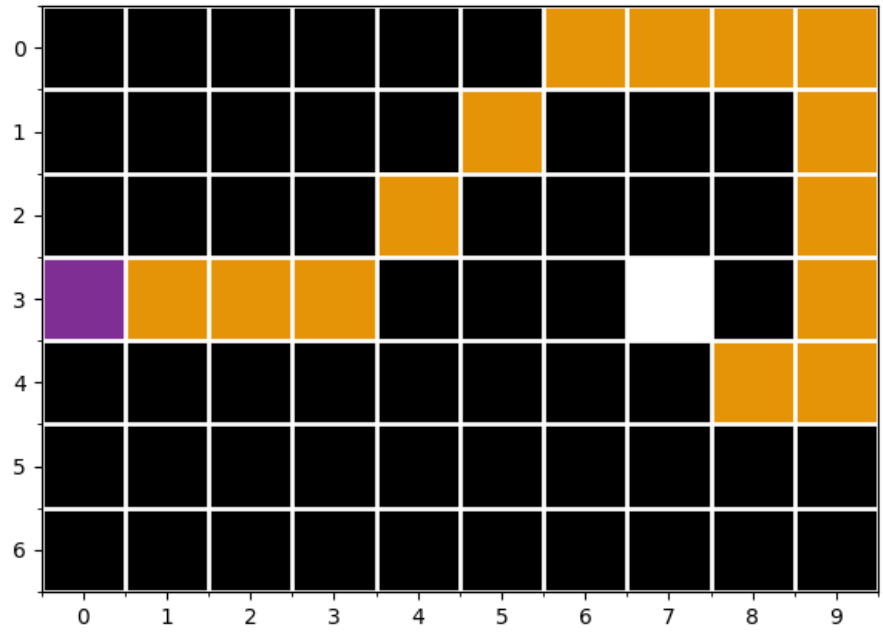


Figure 7 Path obtained for Q-Learning

Expected Sarsa Agent for estimation Q:

Parameters: learning rate α and epsilon ϵ
Initialise $Q(S, A) = 0\ \forall S, A$
Loop for each episode:
    Initialise S = *grid*.startState
    Loop until S is not *grid*.endState:
        Choose A from S using ϵ-greedy policy derived from Q
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \sum_{A'} \pi(S',A') * Q(S',A') - Q(S,A)]$
            where $\pi(S',A') = \frac{\epsilon}{|A|} + 1 - \epsilon\ if\ A'\ is\ optimal\ else\ \frac{\epsilon}{|A|}$
        $S \leftarrow S'$

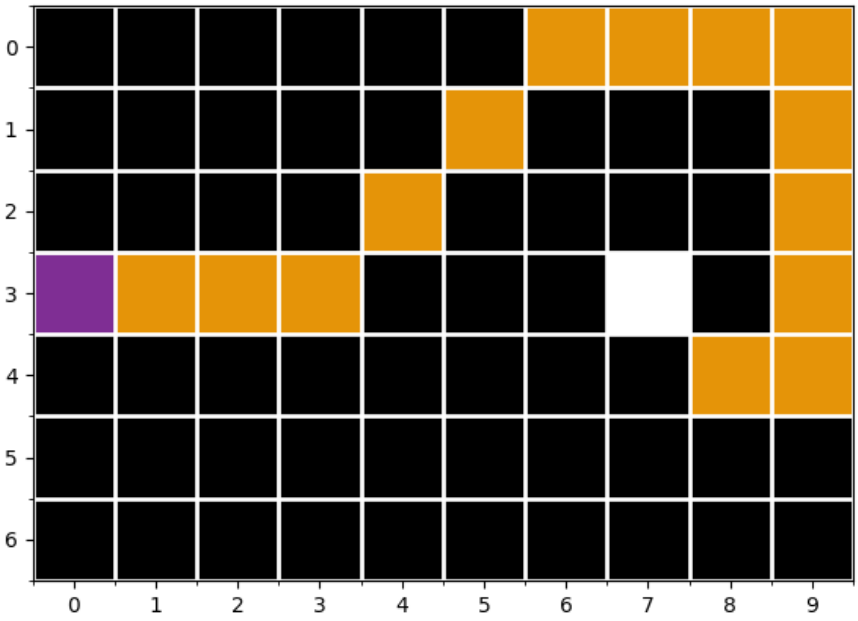


Figure 8 Path obtained for Expected Sarsa Agent

Both Expected Sarsa and Q-Learning take the optimal path.

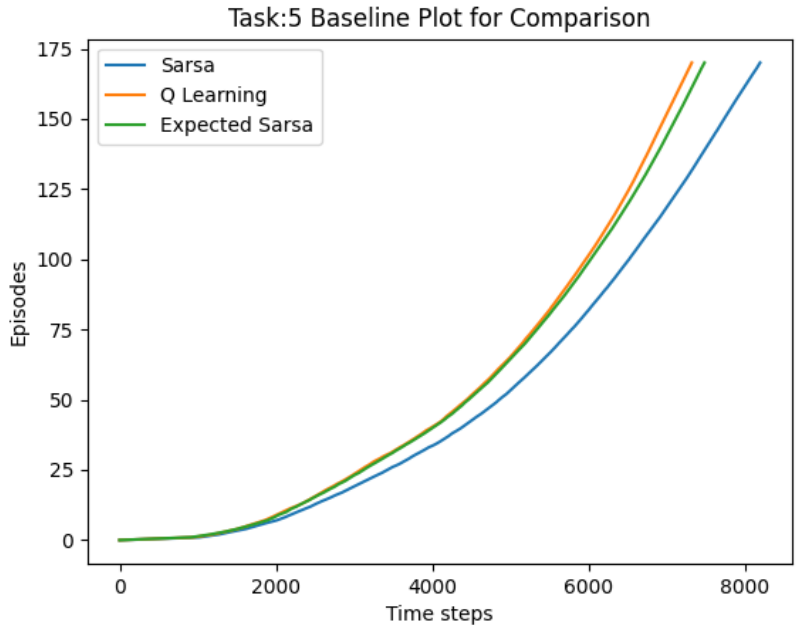


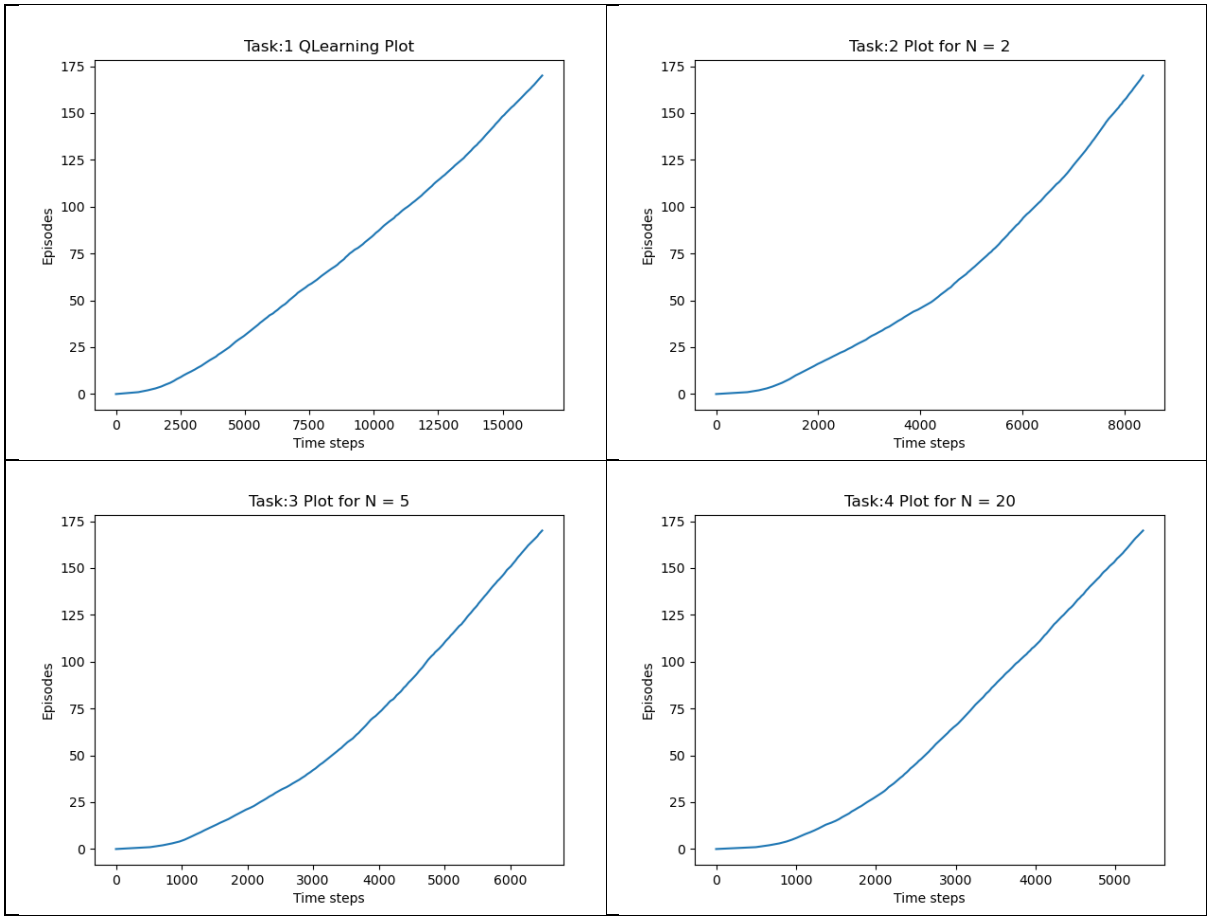
Figure 9 Comparative Baseline Plot for different algorithms

Advanced Agents Task-6

DynaQ-Learning Agent:

Parameters: learning rate α and epsilon ϵ
Initialise $Q(S, A)$=0 for $\forall S, A$
Initialise *Model(S, A)*
Loop for each episode:
    Initialise S = *grid*.startState
    Loop until S is not *grid*.endState:
        Choose A from S using ϵ-greedy policy derived from Q
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \max_{A'} Q(S',A') - Q(S,A)]$
        $UpdateModel(S, A, S', R)$ $\#Stochastic\ Update$
        $S \leftarrow S'$
        Loop repeat N times:
            $\bar{S}, \bar{A} \leftarrow any\ previously\ visited\ pair$
            $\bar{R}, \bar{S'} \leftarrow Model(\bar{S}, \bar{A})$
            $Q(\bar{S},\bar{A}) \leftarrow Q(\bar{S},\bar{A}) + \alpha[\bar{R} + \max_{A'} Q(\bar{S'},A') - Q(\bar{S},\bar{A})]$

Plots:



Observations:

Task 1-5
We observe that all of the three algorithms give the optimal policy following Q greedily after the learning process. However the optimality is driven by choice of epsilon, learning_rate and number of episodes. Changing the parameters sometimes results in a non-optimal path.

In general, we see in the above plot that Q-learning performs the best, followed by Expected Sarsa and Sarsa(0). The same trend is also observed in King's Moves.

Expected Sarsa is expected to perform better than Sarsa as Expected Sarsa update step guarantees to reduce the TD Error whereas Sarsa does that only in expectation. Q-learning directly learns the optimal policy, while Sarsa learns a near-optimal policy whilst exploring. This gives an intuition why Q-Learning learns the optimal policy the fastest followed by Expected Sarsa and Sarsa. Also, If $\pi^t = \pi$ (time-invariant) and it still visits every state-action pair infinitely often, then $\lim_{t\to\infty} \hat{Q}$ is Q^{π} for Sarsa and Expected Sarsa, but is $Q^{\star}$ for Q-learning!

For Dyna-Q **Task-6**:
We observe that 170 episodes complete for QLearning in ~16000 timesteps, for N = 2 in ~8500 timesteps, for N = 5 in ~6500 timesteps and for N = 20 in ~5500 timesteps. Thus the performance of the agent improves considerably by using a Model in addition to modelling the Action Value Function.