REINFORCEMENT LEARNING (CS60077)

TERM-PROJECT (PHASE-1)

PRANAV NYATI

20CS30037

Controlling a Toy Car around a Grid (Project Code: TCV2)

(A) Problem Description:

- **Objective:** Control a car around a discrete square track and complete the lap (starting from the start position and returning after a full lap) without going off the track.
- **Tasks:**
  - Model the state space and reward system for the problem
  - Implement Value Iteration (DP-based approach) to find the Optimal Value function and the Optimal Policy (assuming MDP is known)
  - Implement Model-free control methods such as TD-based methods (SARSA-lambda) and Q-learning algorithms and compare the performance of the two

(B) Environment, State Space, and Reward Formulation:

- **Environment:**
  - A square grid with a pre-defined square track on which the car has to complete the lap. The Environment is simulated using the pygame interface.
- **State Space:**
  - Each state is a tuple of the form **(x_coord, y_coord, direction of car),** where the **(x_coord, y_coord)** denotes the **current position of the car in the grid,** and directions could be any of **'N', 'S', 'E' or 'W'**.
  - I specifically also included the direction of the car in the state characterization and not just the position of the car, as the direction information helps to characterize the states and transitions based on actions more appropriately and also helps to interpret the state values and policy much more clearly.
  - The start state is from the starting x, y coordinate: **(1, grid-size-2)**, with the car pointed in the 'North' direction.
  - Except for the start state, from all other states, there are **3 actions** possible **'left', 'right', and 'forward'** (Backward motion is not allowed), while for the start state **(start position, 'N')**, only 'forward' motion is allowed. Such a constraint on the start state is required for the car to

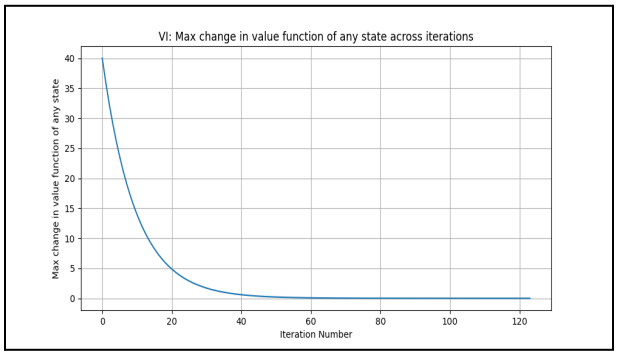
complete the lab, otherwise it will keep oscillating near the start position (which is also the goal) without completing the lap to maximize its reward.
    - **Note:** the top left corner of the grid has the coordinates (0, 0), and the bottom-right one has the coordinates (grid_size-1, grid_size-1)

- **Reward structure:**
    - **Goal reward: +40** for reaching the goal state (after completing the lap)
    - **Forward motion reward: + 20** for taking a forward move from any state that leads to a change of the car's coordinates
    - **Wall hit reward: -5** for going off the track
    - **Rotate reward: -1** if the agent takes a 'left' or 'right' action in a cell and simply remains in the same cell (A negative reward for this is required as; otherwise, the agent will keep rotating in the same cell without achieving the goal)

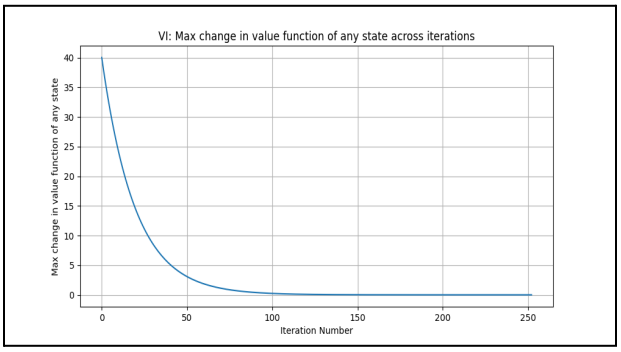
(C) Value Iteration Algorithm (assuming MDP is known)

- It is a DP-based algorithm for finding the Optimal Policy and State Values for a fully-known MDP based on Iterative Application of Bellman Optimality backups.
- **Hyperparams**: **gamma** (discount-factor), **max_iterations**, **precision** (for convergence)
- All experiments were done considering **max-iterations = 2000, prec = 10^{-4},** and experimented with different values of gamma to analyze the rate of convergence of the algorithm to the optimum for different gamma.
- Gamma values taken (0.9, 0.95, 0.99, 1):
    - Gamma = 0.9 converged in around 130 iterations
    - Gamma = 0.95 in around 2250 iterations
    - Gamma = 0.99 in around 1250 iterations
    - Gamma = 1.00 didn't converge in 2000 iterations (and it will never converge as it is the non-terminating case).
    - Thus, we can see convergence is better for lower gamma as it weights the later rewards less
    - Although the optimal state values will be different in each of the above cases (as gamma is different), the optimal policy learned is the same in all the above cases.
- Plots of the optimal value functions and optimal policy for each state, as well as the rate of convergence, have been included in the project zip file.

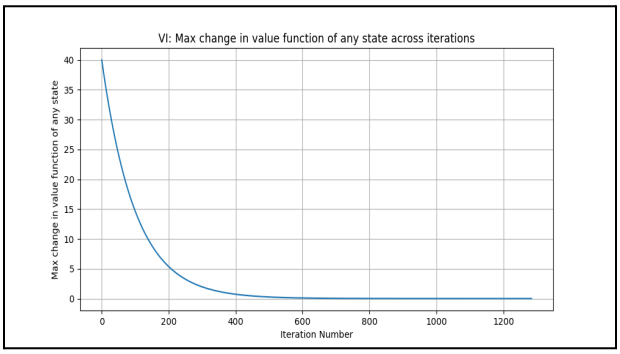
- Plots comparing the rate of convergence for different gamma:



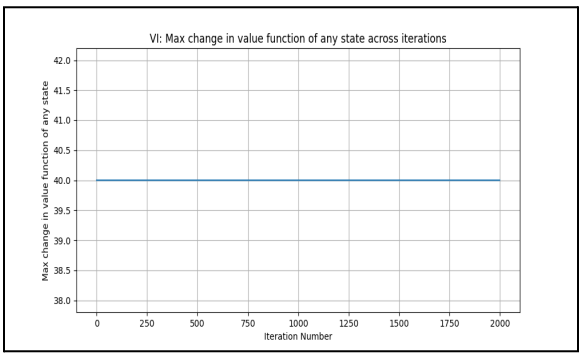
(a) Gamma = 0.9



(b) Gamma = 0.95



(c) Gamma = 0.99



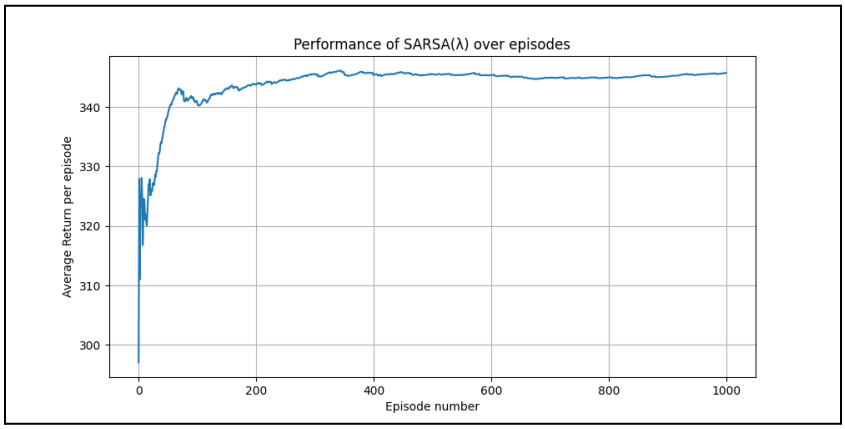
(d) Gamma = 1

(D) Sarsa-Lambda and Q-Learning Algorithms for Model-free Control:

- **SARSA-Lambda:**
  - Sarsa_lambda is an on-policy algorithm for learning the optimal policy for an unknown MDP, meaning that it approximates the state-action values for the current policy, then improves the policy gradually based on the approximate values learned for the current policy. It considers all possible sized returns for approximating the state-action values and weighs them with decaying weights that add up to 1. The policy improvement is done using the epsilon-greedy strategy, and the epsilon can be reduced with each training episode to ensure convergence.
  - **Hyperparams**: **gamma** (discount-factor), **num_iterations**, **epsilon** (for e-greedy strategy), **alpha** (as the learning rate for update rule), **lambda** (for weighing the different step returns)

- All experiments considered gamma = 0.95, epsilon = 0.05, alpha = 0.5, lambda = 0.2. Different numbers of iterations were tried to analyze the extent of learning with different no of episodes, and both the cases of decaying epsilon and non-decaying epsilon were analyzed for each particular number of iteration (Plots included in the zip-file).
- No of iterations considered: 500, 1000, 2000, 4000, 5000, 10000 for both epsilon-decay and without epsilon-decay cases
- **Observations:**
  - The expected return per episode is initially noisy and finally saturates to the near-optimal as the number of episodes increases.
  - The agent learns the near-optimal policy well, even in 500 iterations, and continues to do so for other higher values of num_iterations. Though the state-values obtained using SARSA are closer to the optimal state-values obtained from Value Iteration for higher num_iterations (and in the infinite limit, would approach it), compared to the lower number of iterations
  - Convergence is more speedy for the case of decaying epsilon as after learning the near-optimal policy in the initial few episodes (in some 100-200 episodes), it reduces exploring and exploits more, thus converging to the optimal. For certain cases, the non-epsilon-decaying case took a prohibitively long time and didn't converge, while the decaying case found converged and found the near-optimal policy.

- **Plot of the Learning Curve for Sarsa-lambda:**



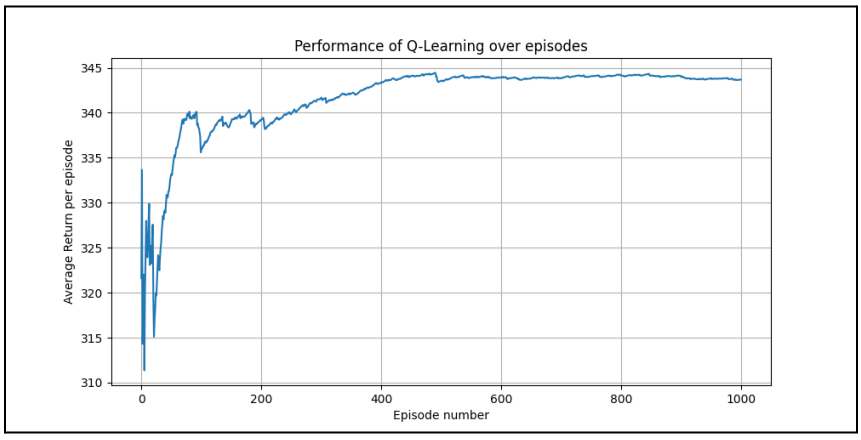
Average return per episode for SARSA for 1000 episodes, w/o decaying epsilon

- **Q-Learning:**
  - Q-learning is an off-policy method, meaning that the policy learned by the method need not be the same as the one used to select actions. In particular, Q-learning learns the greedy policy while it typically follows a policy involving exploratory

actions - occasionally selecting actions that are suboptimal according to the current estimate of state-action values.

- Unlike Sarsa-Lambda, it does not look ahead all the way to the end of the episode in its backup. It only looks ahead as far as the next exploratory action.
- **Hyperparams**: **gamma** (discount-factor), **num_iterations**, **epsilon** (for e-greedy strategy), **alpha** (as the learning rate for update rule)
- All experiments considered gamma = 0.95, epsilon = 0.05, alpha = 0.5. Different numbers of iterations were tried to analyze the extent of learning with different no of episodes, and both the cases of decaying epsilon and non-decaying epsilon were analyzed for each particular number of iteration (Plots included in the zip-file).
- No of iterations considered: 500, 1000, 2000, 5000, 10000, 20000, 40000 for both epsilon-decay and without epsilon-decay cases
- **Observations:**
  - The expected return per episode is initially noisy and finally saturates to the near-optimal as the number of episodes increases.
  - The agent learns the optimal policy well, even in 500 iterations, and continues to do so for other higher values of num_iterations. Though the state-values obtained using Q-Learning are closer to the optimal state-values obtained from Value Iteration for higher num_iterations (and in the infinite limit, would approach it), compared to the lower number of iterations. Also, for num_iterations = 40000 (and non-epsilon decaying case), the state values estimated by Q-Learning are very close to the optimal state values obtained from Value Iteration.
  - Convergence is more speedy for the case of decaying epsilon as after learning the optimal policy in the initial few episodes (in some 100-200 episodes), it reduces exploring and exploits more, thus converging to the optimal faster.

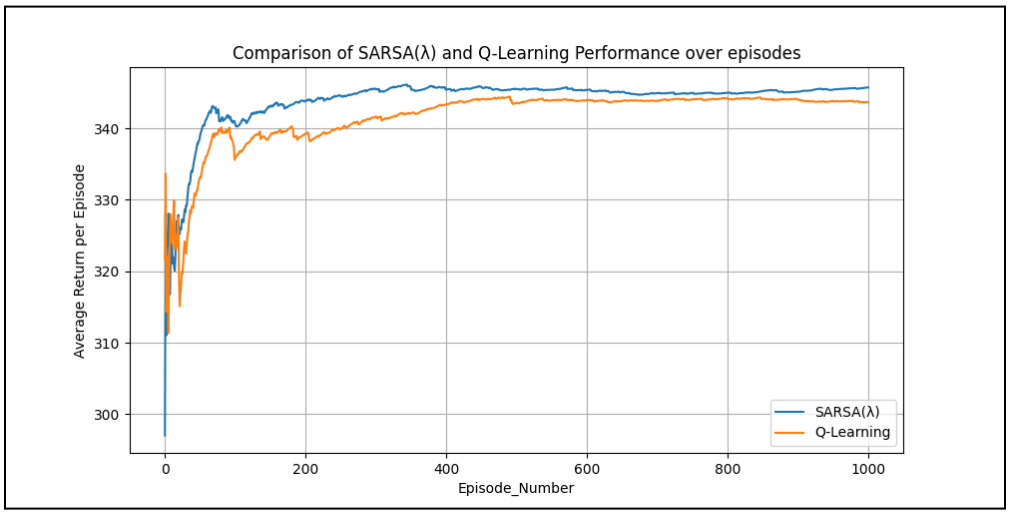
- **Plot of the Learning Curve for Q-Learning:**



Average return per episode for Q-Learning for 1000 episodes, w/o decaying epsilon

- **Comparison of SARSA-Lambda and Q-Learning:**
  - **Observations:**
    - In general, Q-Learning runs considerably faster than SARSA-Lambda for the same number of training episodes but has a worse online performance than SARSA.
    - Q-Learning has a higher variance in average per-episode return as compared to SARSA (can be observed from the above two plots).
    - The state values estimated by Q-Learning for the same number of episodes are much more closer to the optimal state values (as obtained from Value Iteration) compared to those estimated by SARSA for the same number of episodes.
    - In general, the saturated value of average return per episode is higher for SARSA compared to Q-Learning for the same total number of episodes, which can be explained by the fact that SARSA is more conservative. If there is a risk of a large negative reward close to the optimal path, Q-learning will tend to trigger that reward while exploring. In contrast, SARSA will tend to avoid a dangerous optimal path and will prefer a safer path (less negative rewards), only slowly learning to use it when the exploration parameters are reduced.
    - There is no significant difference in the final policies learned by both, which could be because of the very small state space size, thus not leading to much difference in the learning.

  - **Plot comparing the Learning Curve for SARSA and Q-Learning:**



Average return per episode for SARSA and Q-Learning for 1000 episodes w/o epsilon decay