CS6700 - Reinforcement Learning

Programming Assignment 2

Report

Neil Ghosh
ME17B060

Q1. Puddle World

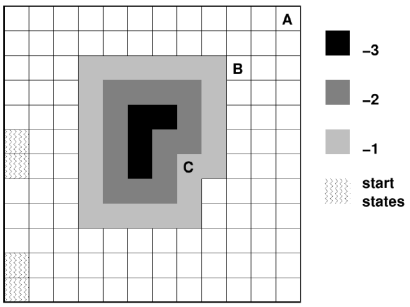
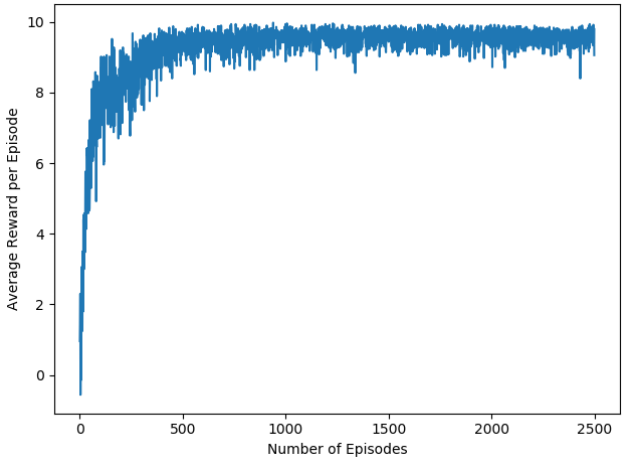
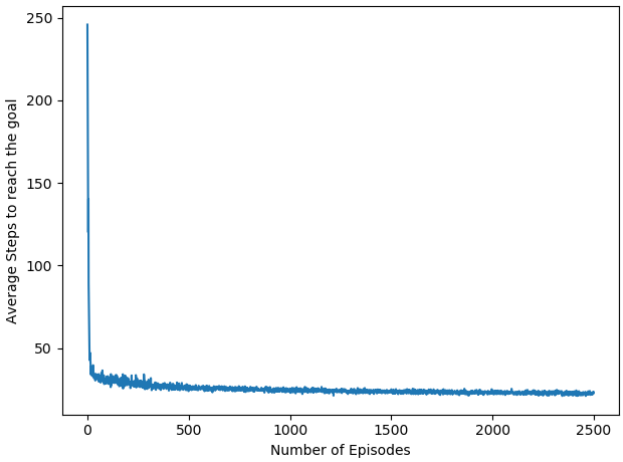


Figure 1: Puddle World

- Q Learning

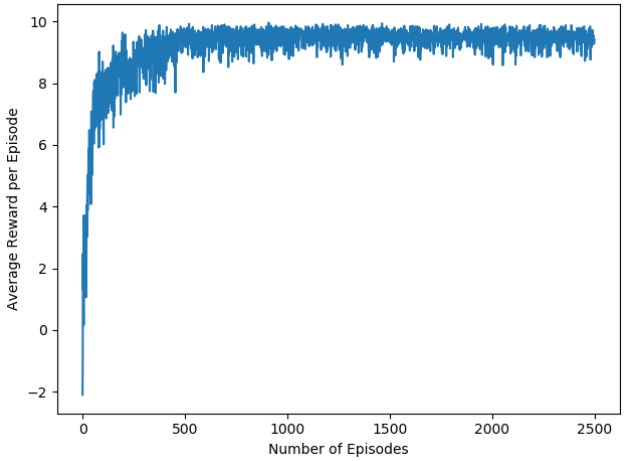


(a) Average Rewards per Episode vs Number of Episodes

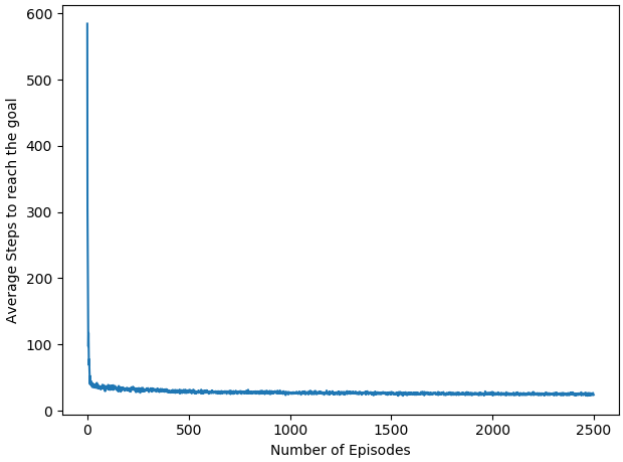


(b) Average Steps per Episode vs Number of Episodes

Figure 2: Performance of Q Learning on the Puddle World when the target is A. Each run contains 2500 episodes and the graphs are averaged over 50 independent runs.

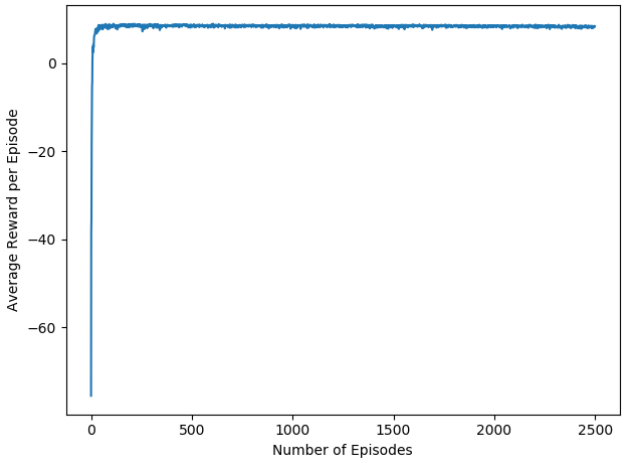


(a) Average Rewards per Episode vs Number of Episodes

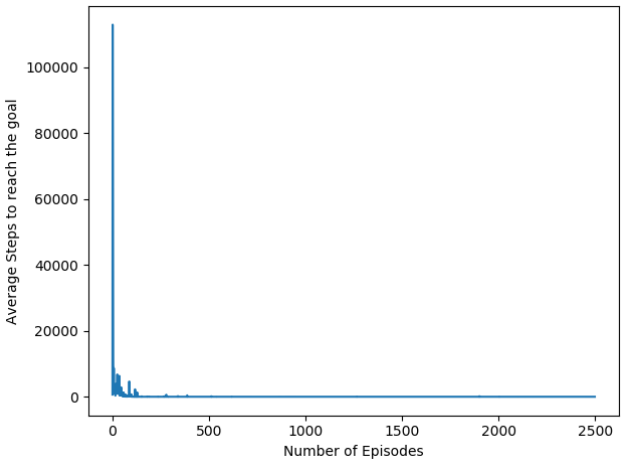


(b) Average Steps per Episode vs Number of Episodes

Figure 3: Performance of Q Learning on the Puddle World when the target is B. Each run contains 2500 episodes and the graphs are averaged over 50 independent runs. The Westerly Wind is considered in this case.

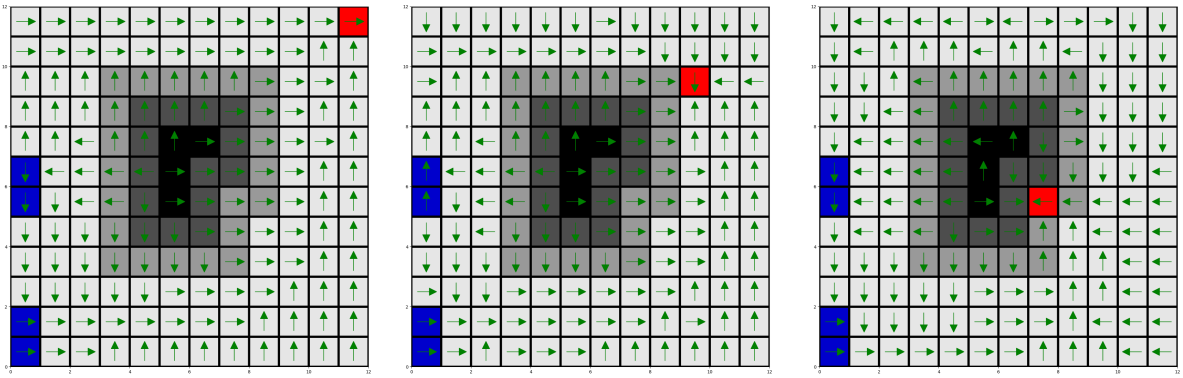


(a) Average Rewards per Episode vs Number of Episodes



(b) Average Steps per Episode vs Number of Episodes

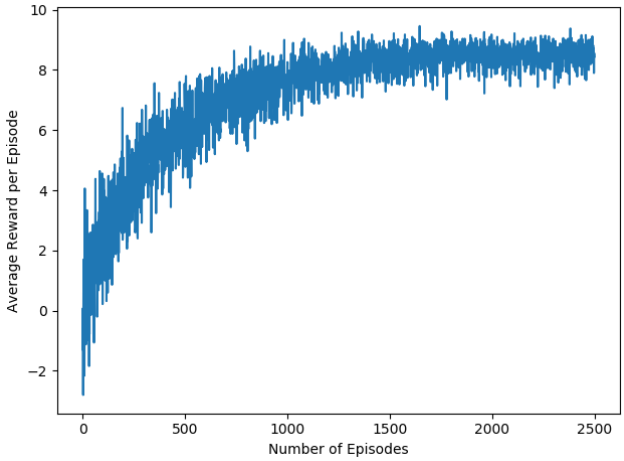
Figure 4: Performance of Q Learning on the Puddle World when the target is C. Each run contains 2500 episodes and the graphs are averaged over 50 independent runs. The Westerly Wind is not considered in this case.



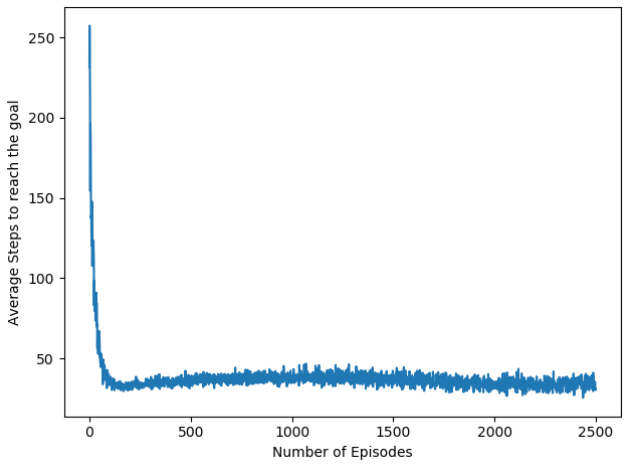
(a) Target A (b) Target B (c) Target C

Figure 5: The optimal policies obtained when Q Learning is used averaged over 50 runs of 2500 episodes each. The red cell is the target and the blue cells are the starting points.

- SARSA

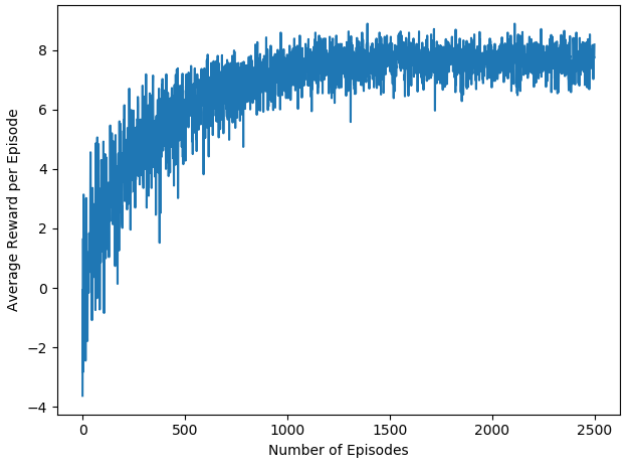


(a) Average Rewards per Episode vs Number of Episodes

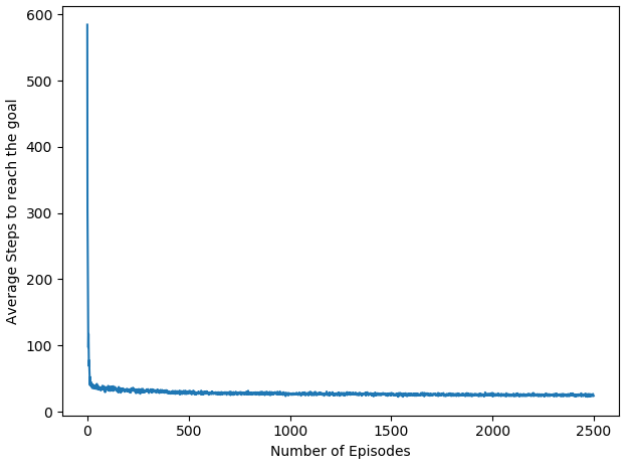


(b) Average Steps per Episode vs Number of Episodes

Figure 6: Performance of SARSA on the Puddle World when the target is A. Each run contains 2500 episodes and the graphs are averaged over 50 independent runs.

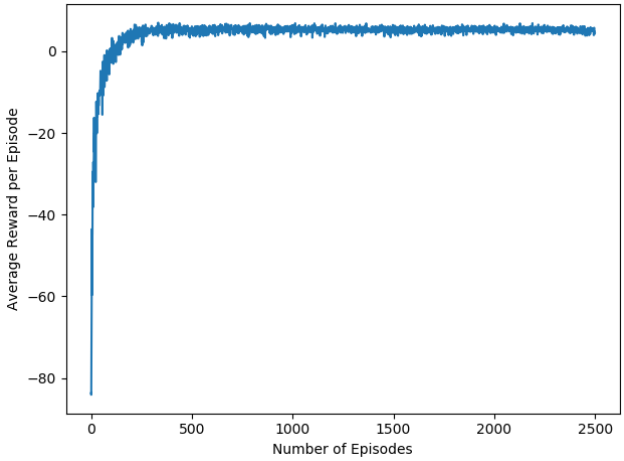


(a) Average Rewards per Episode vs Number of Episodes

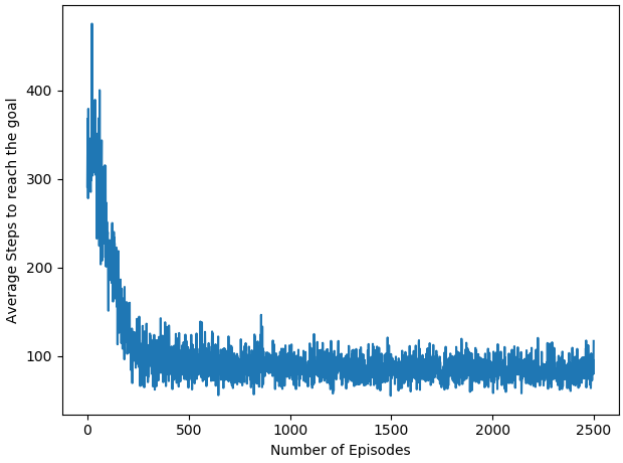


(b) Average Steps per Episode vs Number of Episodes

Figure 7: Performance of SARSA on the Puddle World when the target is B. Each run contains 2500 episodes and the graphs are averaged over 50 independent runs. The Westerly Wind is considered in this case.

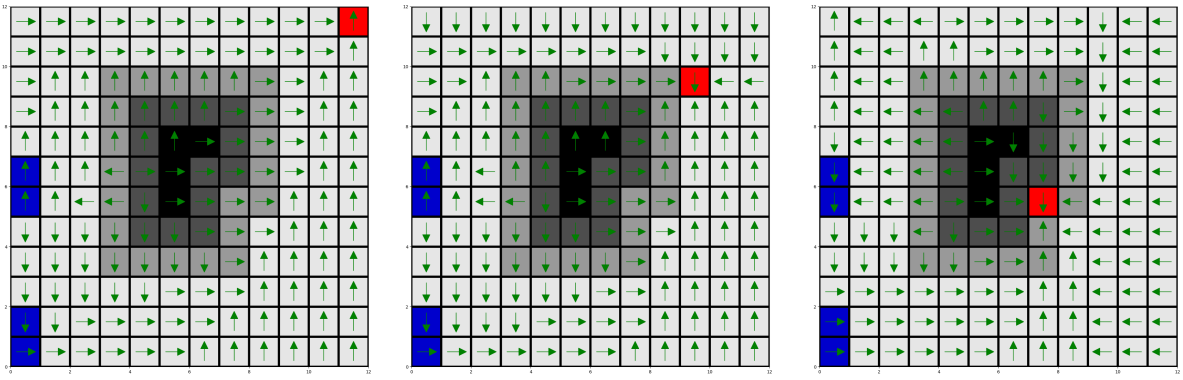


(a) Average Rewards per Episode vs Number of Episodes



(b) Average Steps per Episode vs Number of Episodes

Figure 8: Performance of SARSA on the Puddle World when the target is C. Each run contains 2500 episodes and the graphs are averaged over 50 independent runs. The Westerly Wind is not considered in this case.

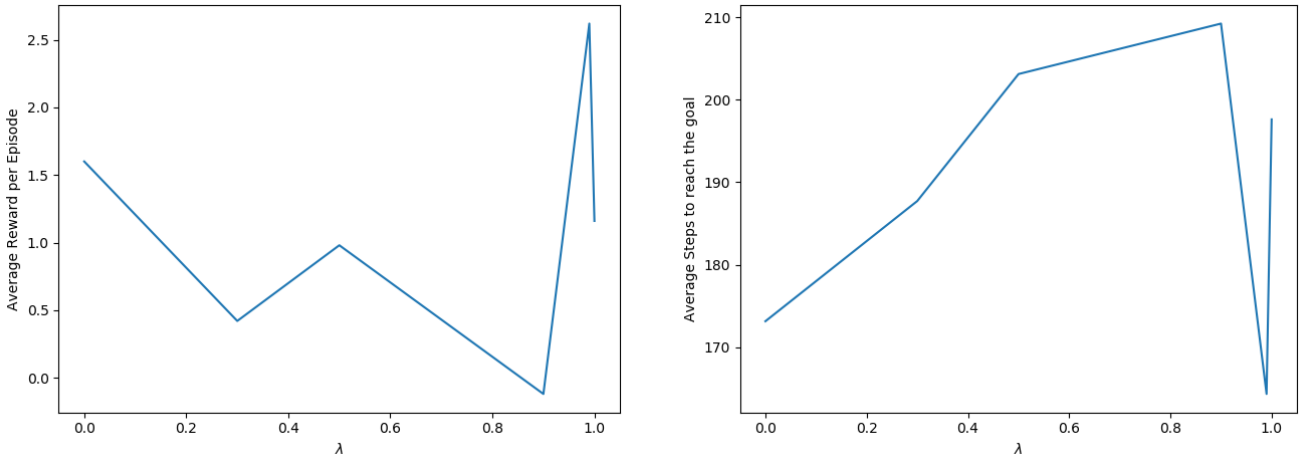


(a) Target A (b) Target B (c) Target C

Figure 9: The optimal policies obtained when SARSA is used averaged over 50 runs of 2500 episodes each. The red cell is the target and the blue cells are the starting points.

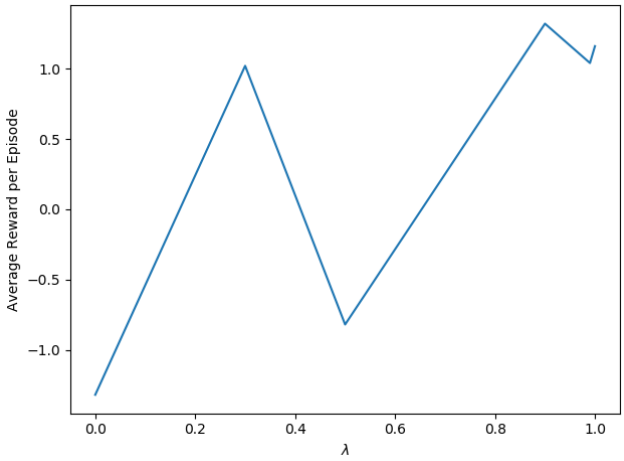
- SARSA (λ)

SARSA (λ) for the following plots were run using $\lambda = \{0, 0.3, 0.5, 0.9, 0.99, 1.0\}$. The performance was averaged over 50 independent runs at the end of 25 episodes each.

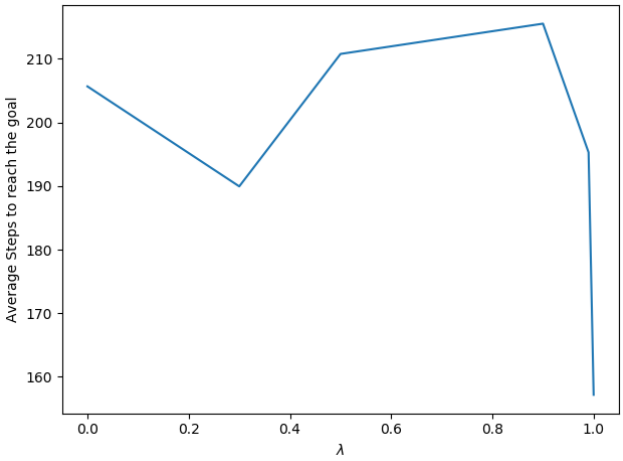


(a) Average Rewards per Episode vs λ (b) Average Steps per Episode vs λ

Figure 10: Performance of SARSA (λ) on the Puddle World when the target is A. Each run contains 25 episodes and the graphs are averaged over 50 independent runs.

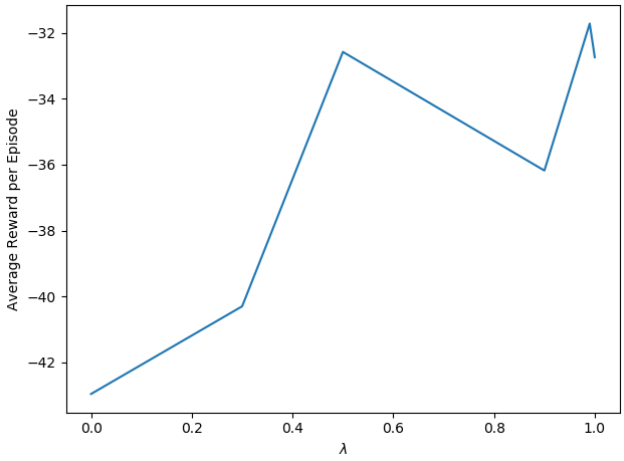


(a) Average Rewards per Episode vs λ

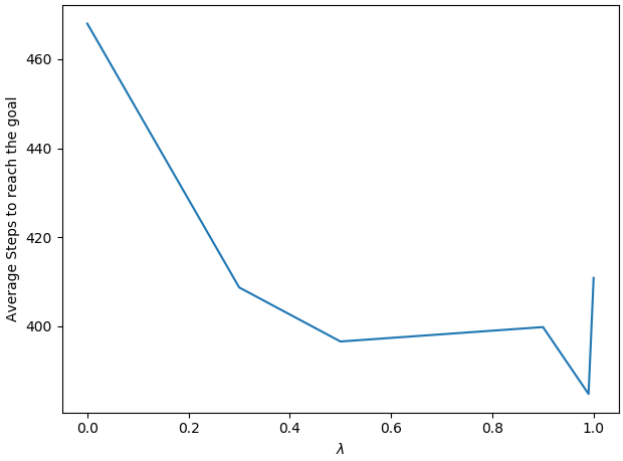


(b) Average Steps per Episode vs λ

Figure 11: Performance of SARSA (λ) on the Puddle World when the target is B. Each run contains 25 episodes and the graphs are averaged over 50 independent runs. The Westerly Wind is considered in this case.



(a) Average Rewards per Episode vs λ

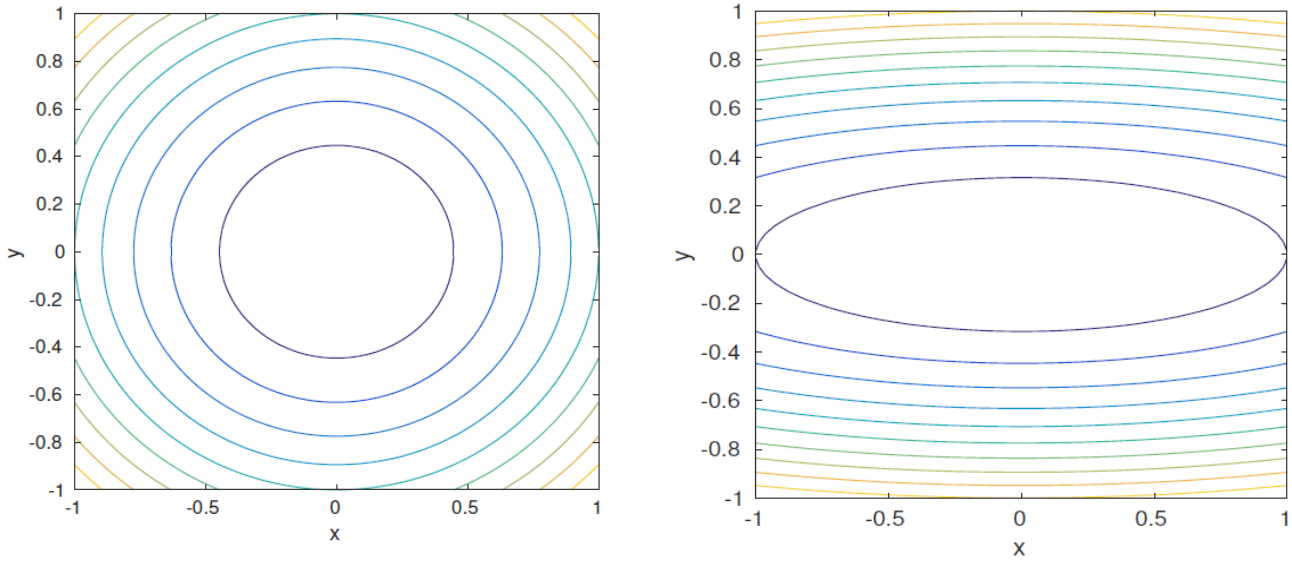


(b) Average Steps per Episode vs λ

Figure 12: Performance of SARSA (λ) on the Puddle World when the target is C. Each run contains 25 episodes and the graphs are averaged over 50 independent runs. The Westerly Wind is not considered in this case.

Q2. Policy Gradient

The problem had two environments - "chakra" and "vishamC". Monte Carlo Policy Gradient (REINFORCE) was implemented on both of them by using a batch update. An epsiode is terminated when the distance of the current state from the origin falls below a limit (0.1). The starting state was chosen randomly from the world. The state was reset to a randomly chosen point inside the world if any action caused the state to leave the world.



(a) Chakra world

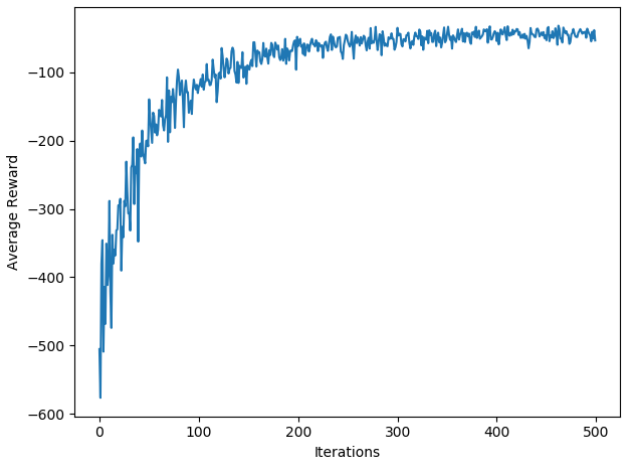
(b) VishamC world

Figure 13: Both the environments are shown above. For chakra, the reward function is simply negative of the distance of current state from the origin. For vishamC, the reward function is a little skewed, and given by : $-(0.5x^2 + \gamma 0.5y^2)$, where γ is 10.

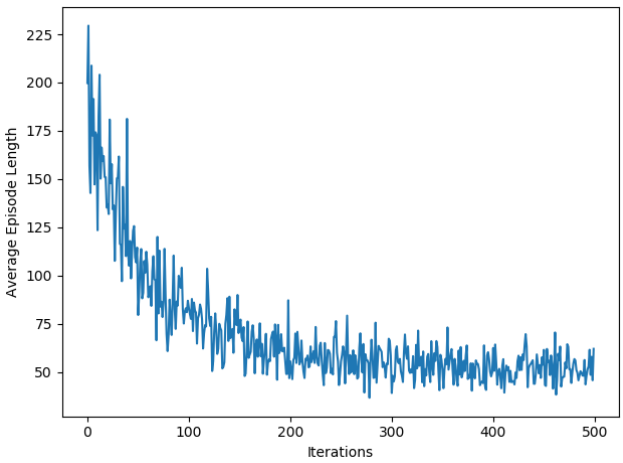
After implementation of the policy gradient algorithm on the chakra world the optimal parameters learnt were approximately - $\theta = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$

The learnt values of θ are correct because the optimal action in any state is to move towards the origin in the chakra world. Similar values of θ are also learnt in the vishamC environment. The results are obtained with a batch size of 50 and $\gamma = 0.9$.

The results on both the environments are shown below -

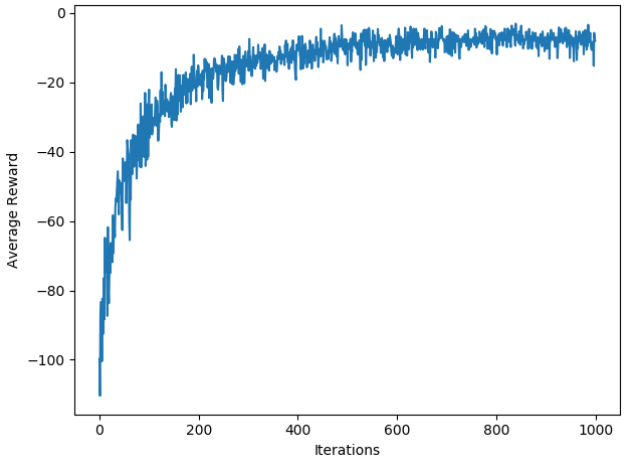


(a) Average Rewards vs Iterations

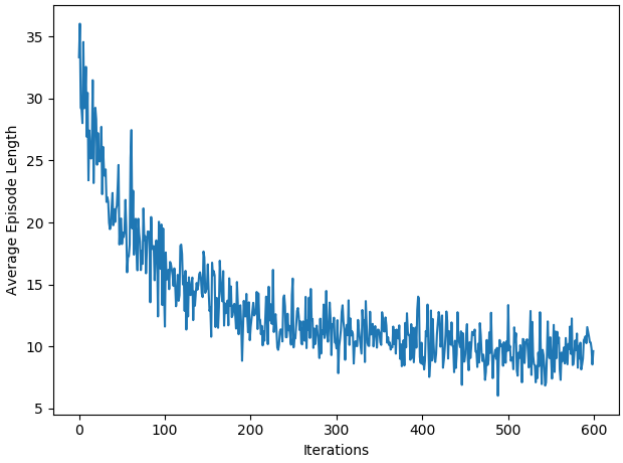


(b) Average Steps for Convergence vs Iterations

Figure 14: The performance of the policy gradient algorithm on the chakra world in shown above. The results are obtained for $\gamma = 0.9$ and $\alpha_\theta = 0.0001$ with a batch size of 50. The rewards are averaged over a single episode while the steps are averaged over a single batch.



(a) Average Rewards vs Iterations

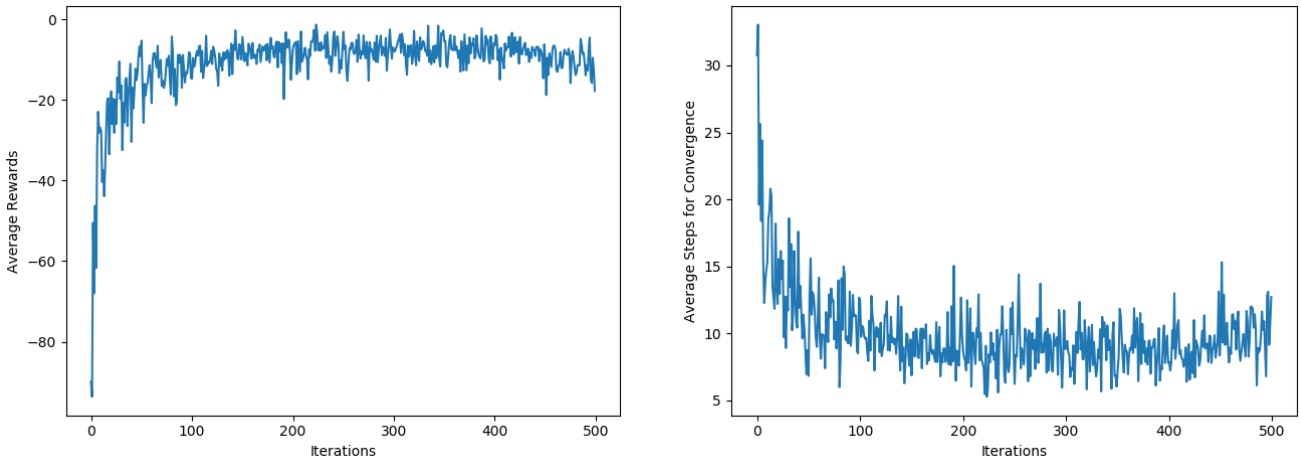


(b) Average Steps for Convergence vs Iterations

Figure 15: The performance of the policy gradient algorithm on the vishamC world in shown above. The results are obtained for $\gamma = 0.9$ and $\alpha_\theta = 0.0001$ with a batch size of 50. The rewards are averaged over a single episode while the steps are averaged over a single batch.

- Hyperparameter Tuning

1. **Batch Size -** The number of iterations taken for the optimal policy to be learnt is almost the same. For a higher batch size, fewer episodes are required for the value of θ to converge as the computed gradient will be more accurate. For a lower batch size, since the gradient will not be as accurate it may take us away from the target for some steps (it will be towards the target on an average). A batch size of 50 was used for both the environments.
2. **Learning Rate -** If the value of the learning rate α_θ is small, the update will be slow and it will take a long time to converge. A small learning rate can also lead to algorithm to get stuck in a local minima instead of the global minima. On the other hand if we use a large learning rate, we can converge quickly to the global minima but, in some situations it may even lead to divergence if the algorithm overshoots the minima. The value of α_θ is chosen as 0.0001.
3. **Discount Factor -** On decreasing the value of the discount factor (γ), the optimal policy is found faster. $\gamma = 0.9$ is chosen for both the environments.



(a) Average Rewards vs Iterations

(b) Average Steps for Convergence vs Iterations

Figure 16: The above graphs are obtained for $\gamma = 0.9$ and $\alpha_\theta = 0.0001$ with a batch size of 25.

- ## Value Function Visualisation

  The value function of the environments is visualized as follows -

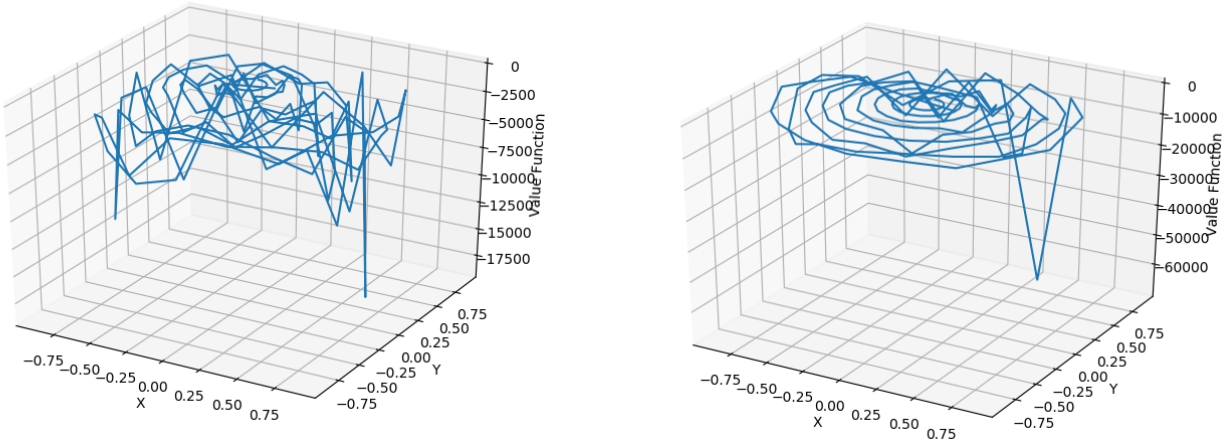


Figure 17: The above 3D plots show the visualisation of the value functions for both the environments. In 2D, the contours should be concentric circles and ellipses for the worlds respectively.

- ## Trajectory of Policy

  The final policy is obtained after using the policy gradient algorithm. A starting state is picked at random and actions from that state are picked using the final value of θ which is obtained.
  For the chakra environment, the trajectory of the policy brings it towards the origin in a straight line. In the vishamC environment, the policy has a trajectory such that the path is a little skewed (because the reward is skewed) and the optimal policy tries to take a larger step in the y direction as compared to the x direction.