Programming Assignment 3

Sai Praveen B

April 10, 2016

1 QA

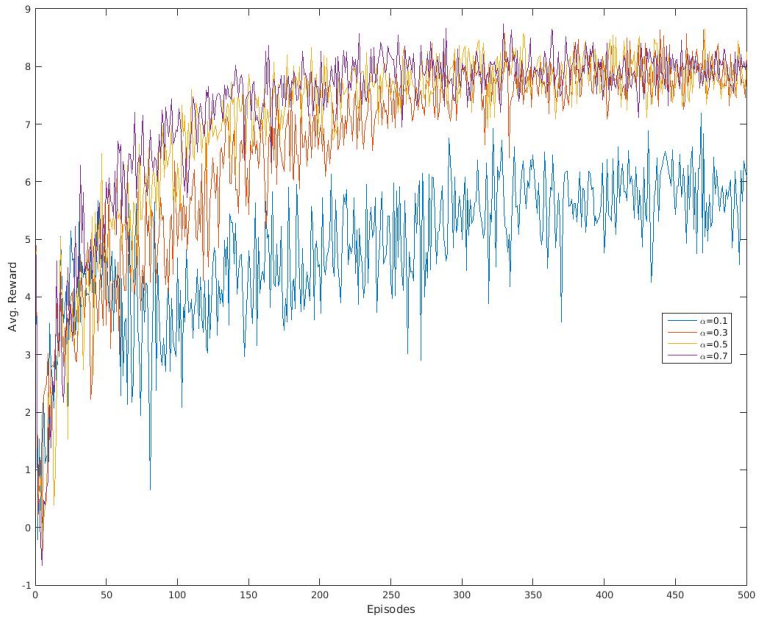
RL-glue for C++ has been used to simulate the agent and environment.

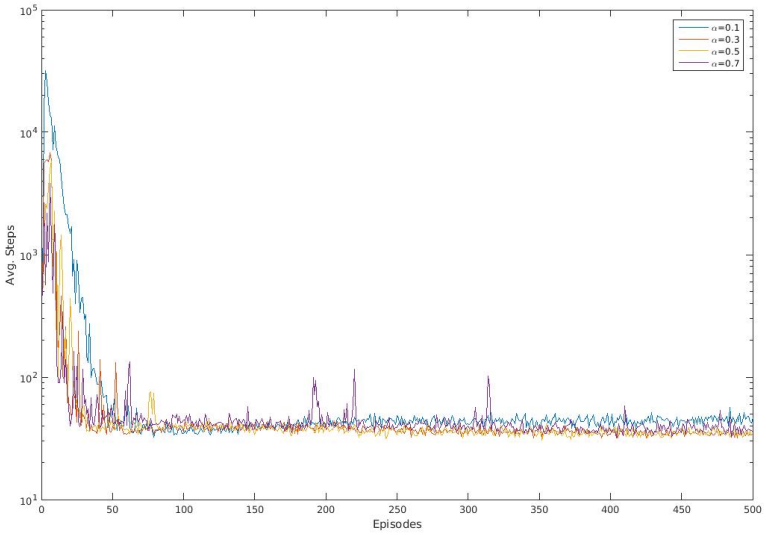
Instead of averaging over 50 runs, Averages are over 200 runs to reduce the noise(most of the methods, especially MC Policy gradient hav very high variance).

1.1 Q-learning

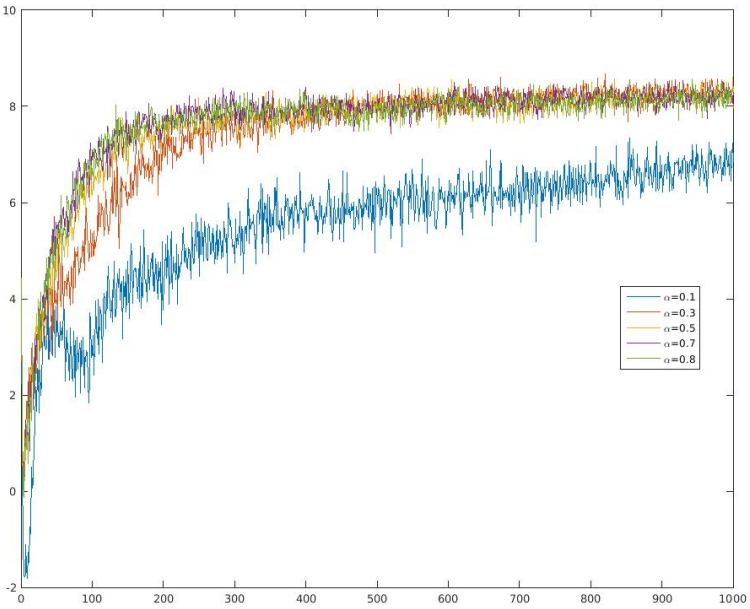
Q - learning has been implemented in C++(File: QAgent.cpp) The performance plots of the agent for each of the three targets are as follows:

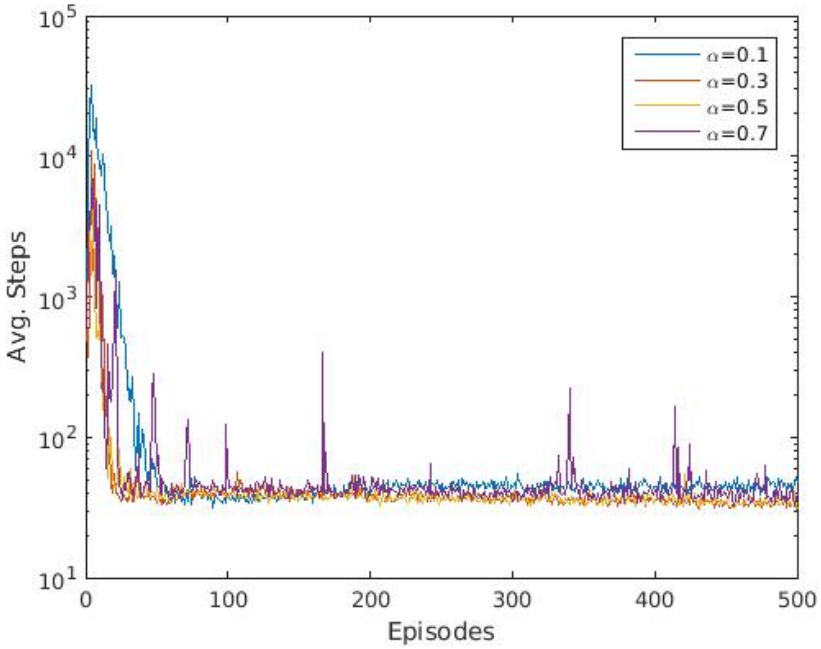
(Averages over 250 runs for various alpha) Target A:



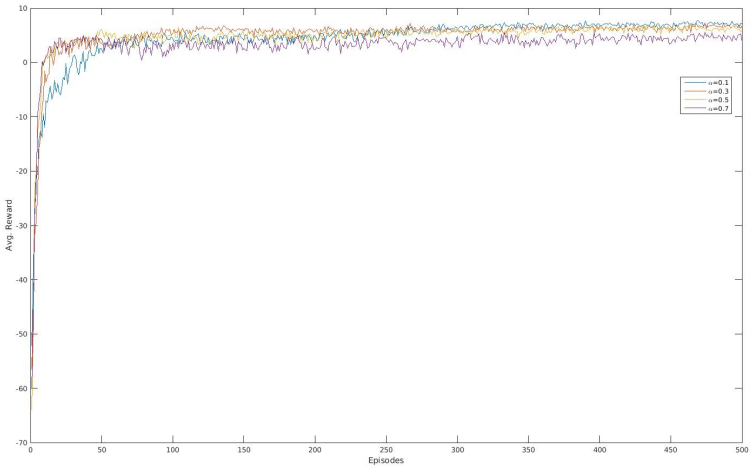


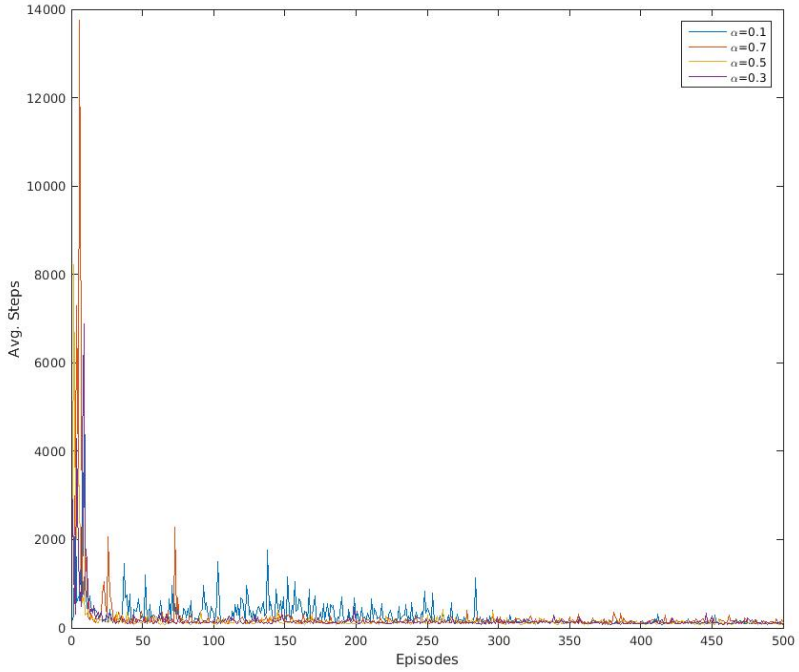
Target B:


10
8
6
4
2
0
-2
0
100
200
300
400
500
600
700
800
900
1000
α=0.1
α=0.3
α=0.5
α=0.7
α=0.8


10^5
10^4
10^3
10^2
10^1
Avg. Steps
0
100
200
300
400
500
Episodes
α=0.1
α=0.3
α=0.5
α=0.7

Target C:

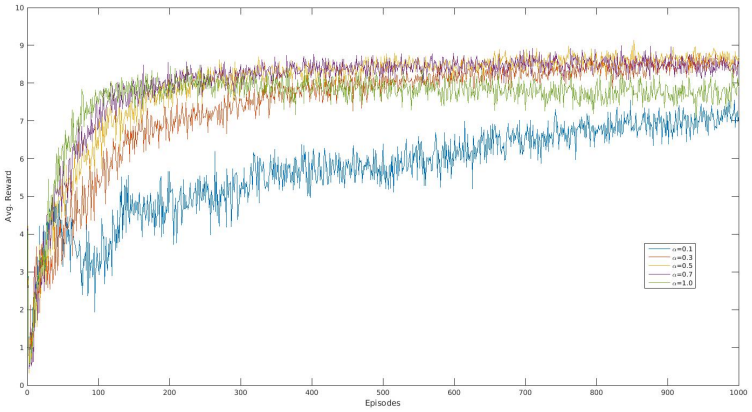


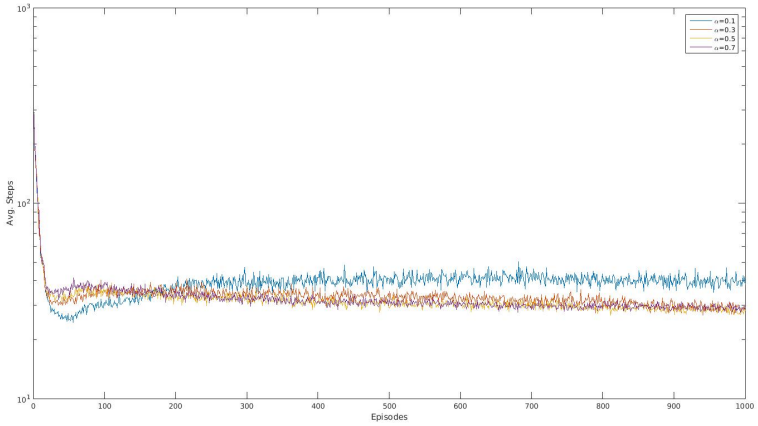


1.2 SARSA

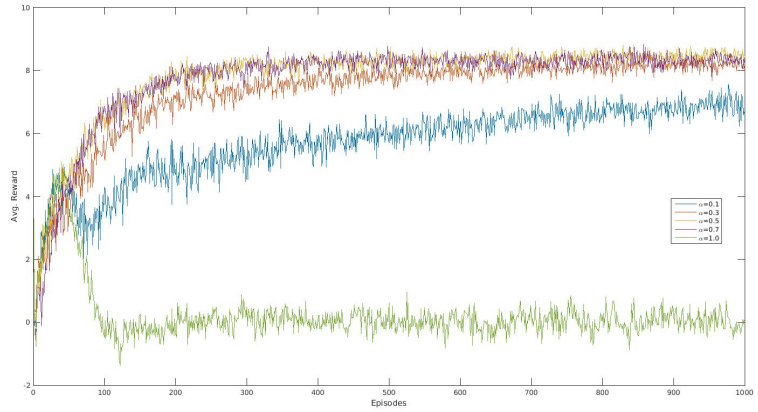
The standard SARSA algorithm has been implemented in C++(File: SARSAAgent.cpp)

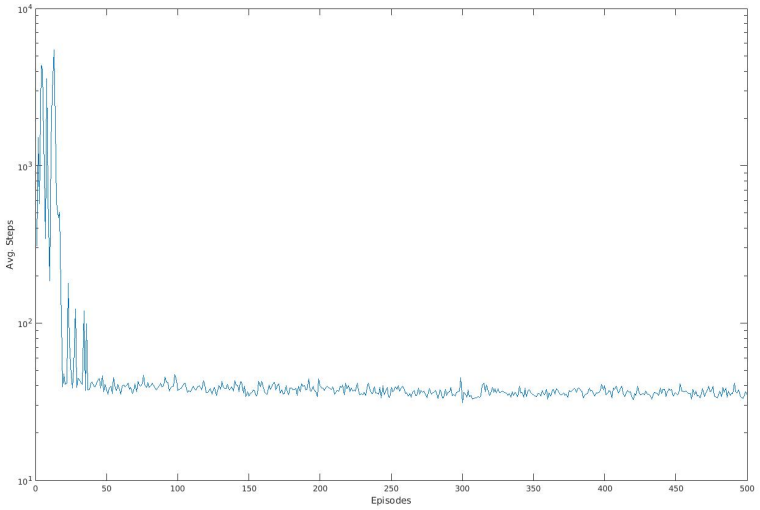
Target A:

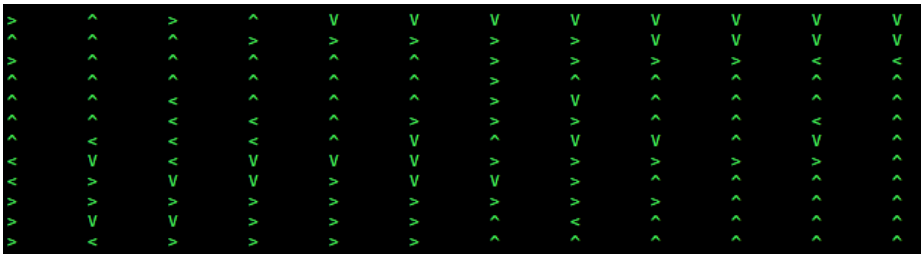

Avg. Reward
Episodes
α=0.1
α=0.3
α=0.5
α=0.7
α=1.0


Avg. Steps
Episodes
α=0.1
α=0.3
α=0.5
α=0.7

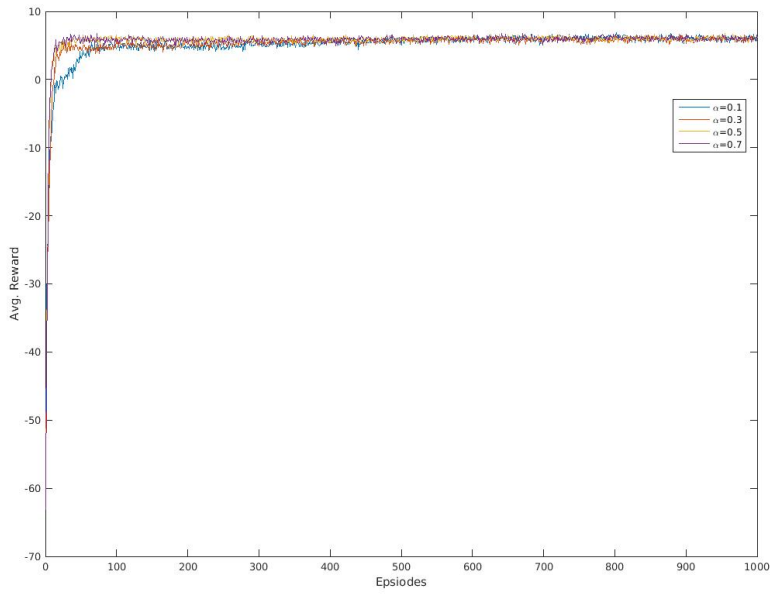
Target B:

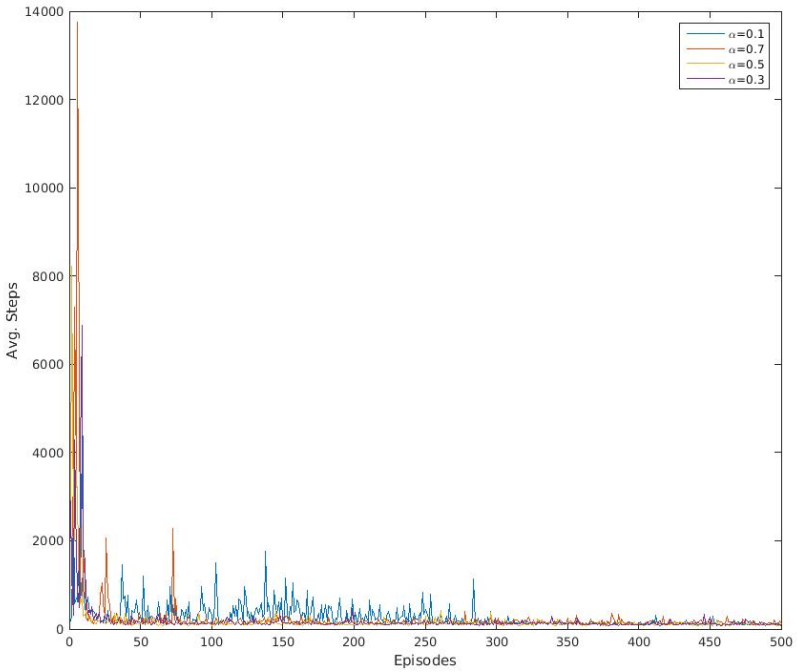






Target C:


Avg. Reward
Epsiodes
α=0.1
α=0.3
α=0.5
α=0.7

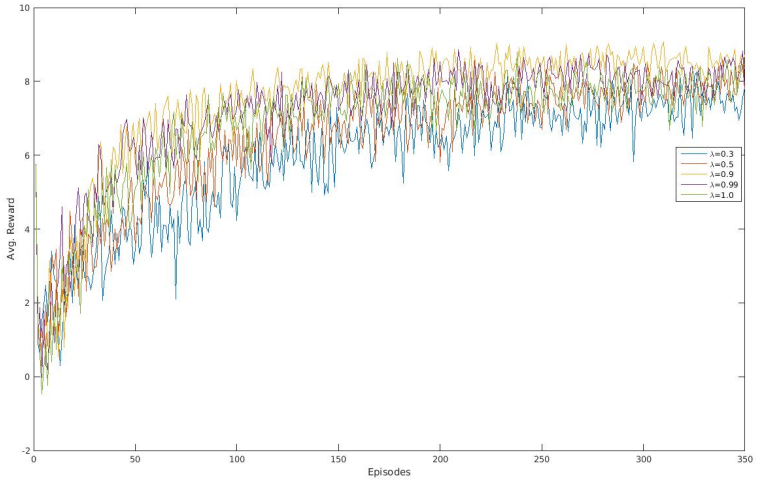

Avg. Steps
Episodes
α=0.1
α=0.7
α=0.5
α=0.3

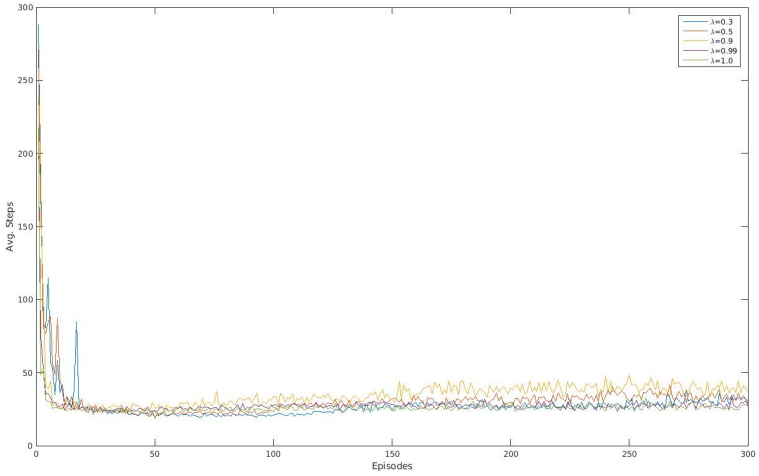
1.3 SARSA Lambda

SARSA Lambda has been implemented with 2 methods. The first one uses a history list of episodes and thus implements Accumulating traces.(File: SARSALambdaAgent.cpp) The second one uses a matrix of eligibilities and currently implements Replacing traces. (File: SARSAFSLambdaAgent.cpp)

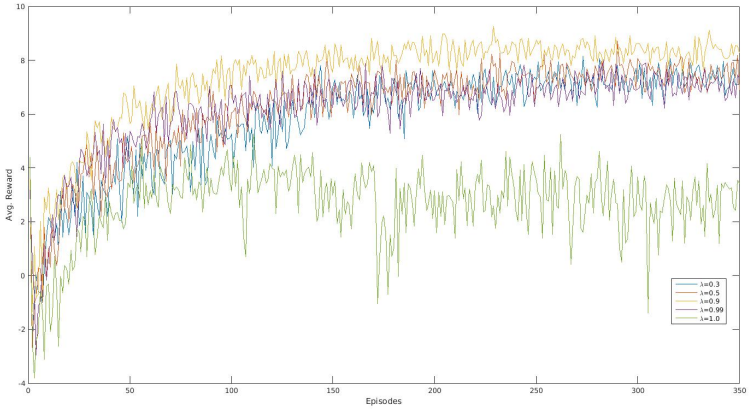
The results shown below are averaged over 50 runs for Replacing Traces only.

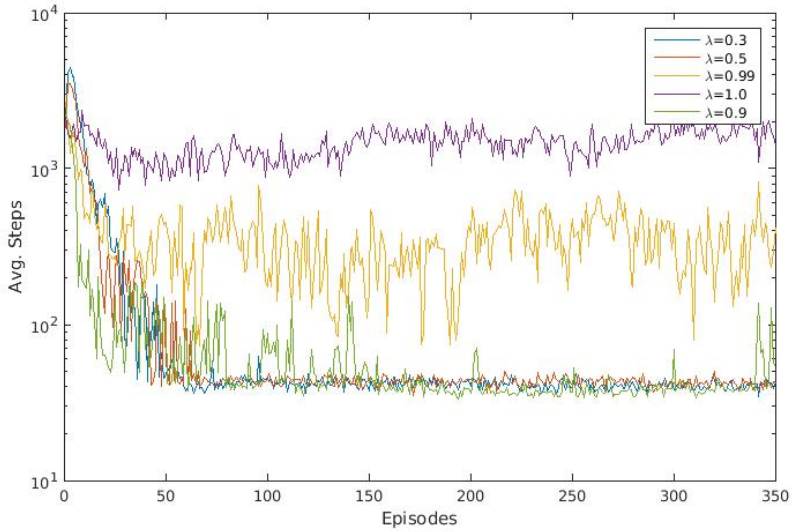
The results have been plotted for 5 values of $\lambda = 0.3, 0.5, 0.9, 0.99, 1.0$ Target A:



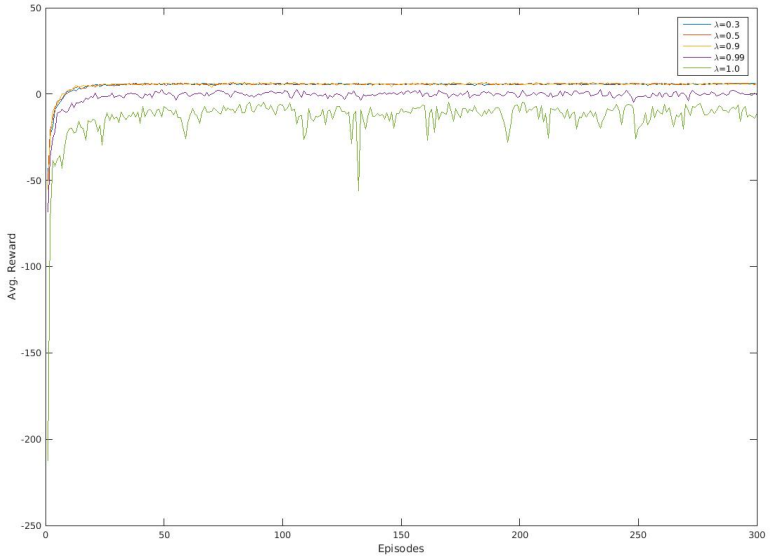

λ=0.3
λ=0.5
λ=0.9
λ=0.99
λ=1.0
Avg. Steps
Episodes
0
50
100
150
200
250
300

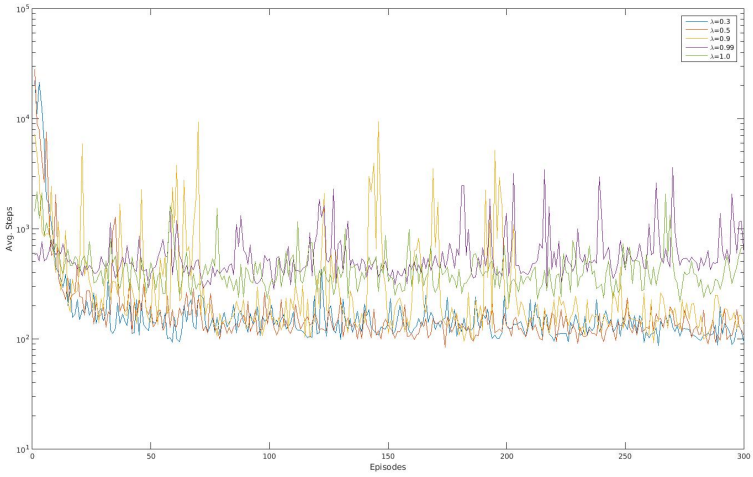
Target B:


λ=0.3
λ=0.5
λ=0.9
λ=0.99
λ=1.0
Avg. Reward
Episodes
-4
-2
0
2
4
6
8
10
0
50
100
150
200
250
300
350


λ=0.3
λ=0.5
λ=0.99
λ=1.0
λ=0.9
Avg. Steps
Episodes

Target C:


λ=0.3
λ=0.5
λ=0.9
λ=0.99
λ=1.0
Avg. Reward
Episodes



1.4 MC Policy Gradient

A Monte Carlo Policy gradient technique has been implemented in MCGradientAgent.cpp

We derive the update rules to the REINFORCE-like equations as follows: Our function is:

$$\pi_\theta(s_t, a_t) = \frac{e^{\theta_{s_{x,t},a_t} + \theta_{s_{y,t},a_t}}}{\sum_b e^{\theta_{s_{x,t},b} + \theta_{s_{y,t},b}}}$$

Our target algorithm:

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

We now differentiate to get the gradient as:

$$\Delta_{\theta_{s_{x,t},a_t}} \pi_\theta(s'_t, b_t) = 1 - \pi_\theta(s_t, a_t), s_{x,t} = s'_{x,t}, a_t = b_t$$

$$\Delta_{\theta_{s_{x,t},a_t}} \pi_\theta(s'_t, b_t) = 1 - \pi_\theta(s_t, a_t), s_{x,t} = s'_{x,t}, a_t = b_t$$

$$\Delta_{\theta_{s_{x,t},a_t}} \pi_\theta(s'_t, b_t) = 0, otherwise$$

The same rules apply to y as well.

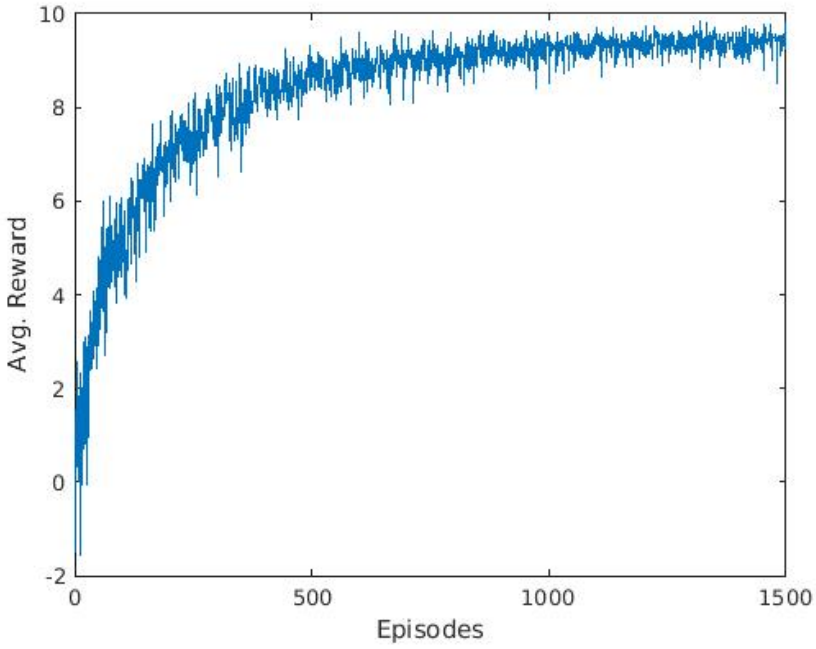
The gradient is factored with an α (learning rate) which has been fixed at $\alpha = 0.012$ after much trail and error. The method uses a per row/column per action parametrization which is inherently bad for the general case.

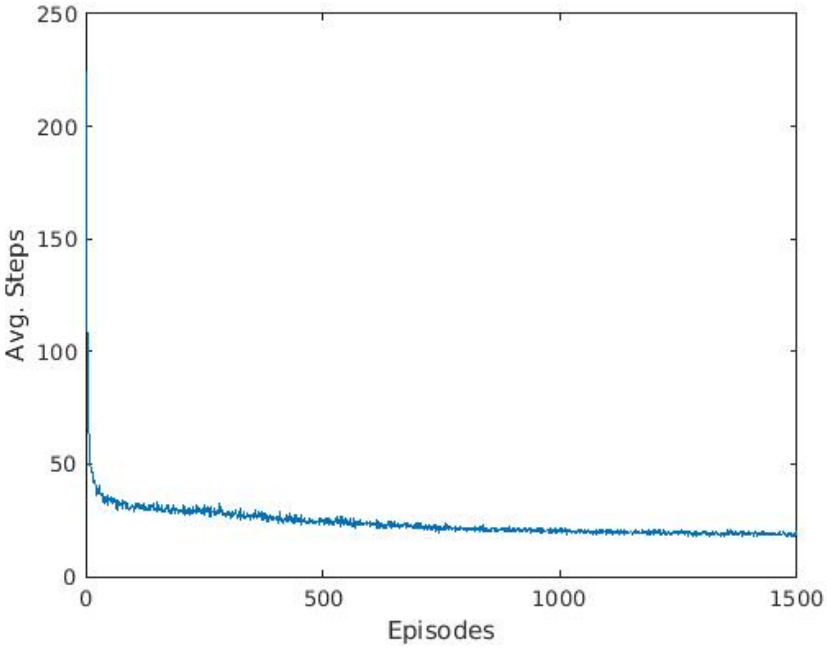
Target A is very easy and has an optimal solution that can be represented using the parametrization given. It gets an average reward of around 9. Avg. Step count of around 20.

Target B is considerably tougher given that the row has other negative cells along it and an attempt to optimize near the target can lead to bad decisions elsewhere. It manages an average reward of around 7 and an average step count of around 50.

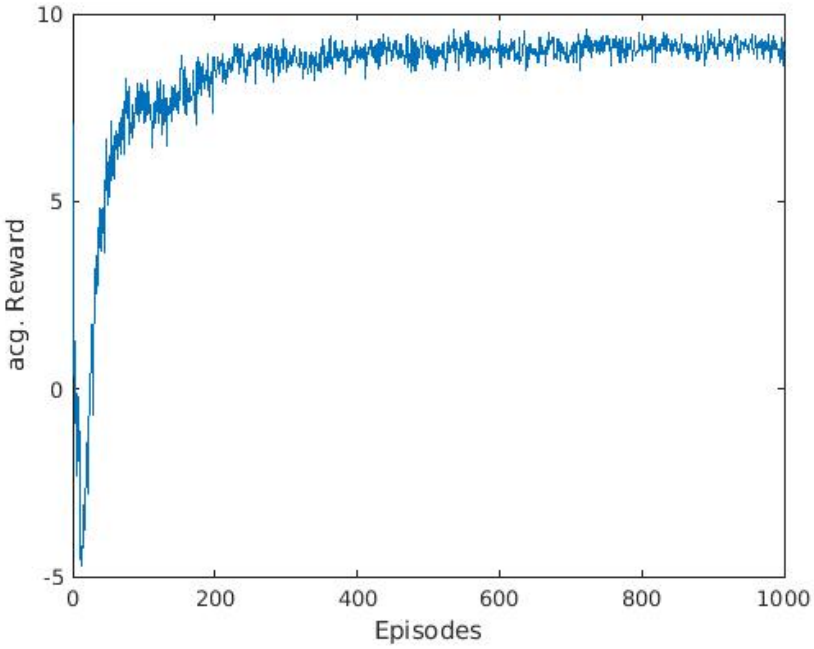
Target C is very difficult as it's in the middle of the puddle with high negaties in both the row and column. This leads to a variety of issues including the agent often getting stuck due to bad loops formed due to ripples that occur because of the non-local nature of the parametrization. It only manages a reward of around 4 and an even worse step count of around 400.

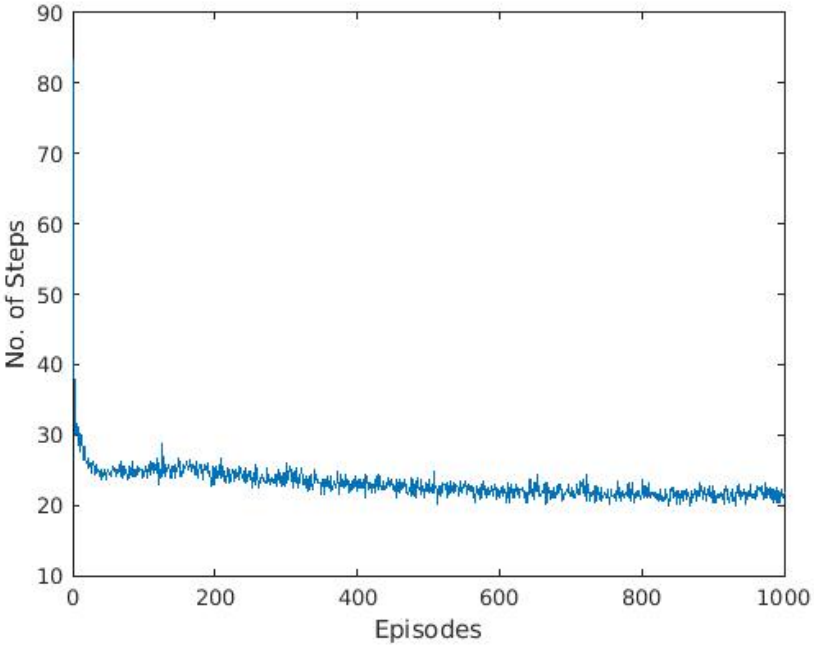
Target A:

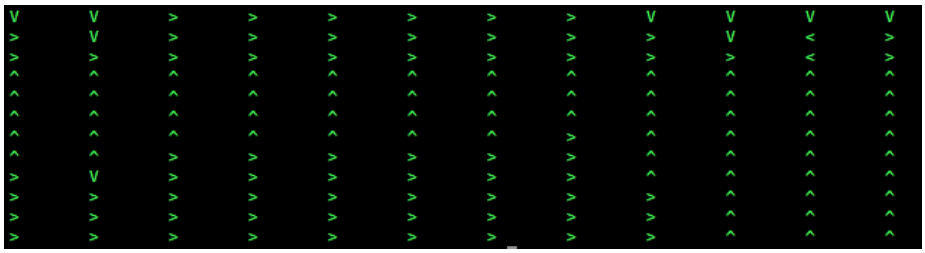




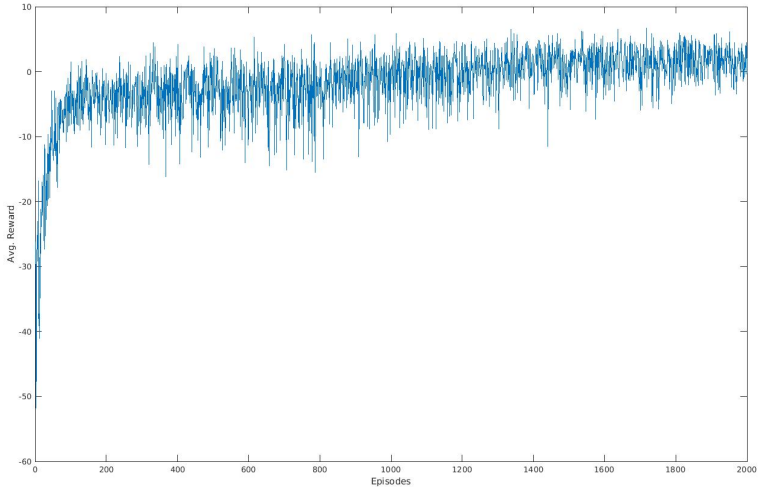
Target B:

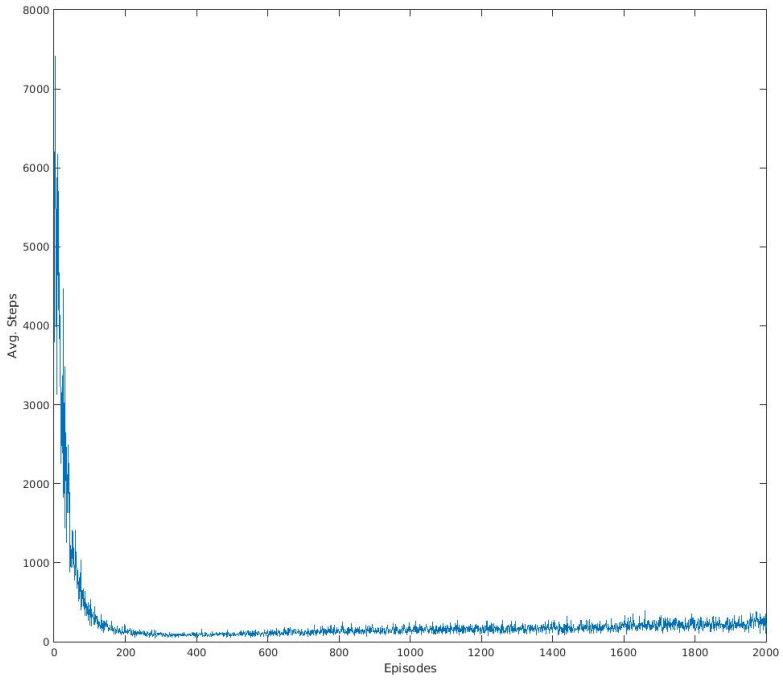

10
5
0
-5
acg. Reward
0
200
400
600
800
1000
Episodes

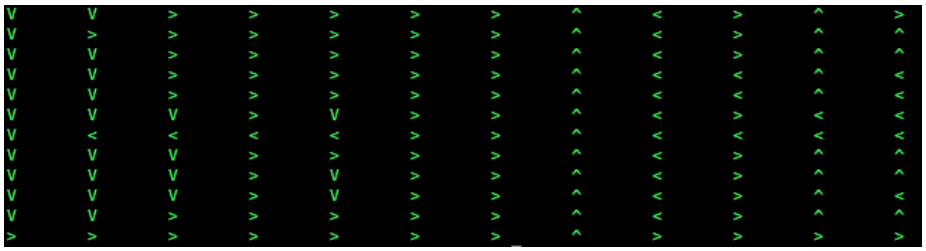

90
80
70
60
50
40
30
20
10
No. of Steps
0
200
400
600
800
1000
Episodes



Target C:


Avg. Reward
Episodes
10
0
-10
-20
-30
-40
-50
-60
0
200
400
600
800
1000
1200
1400
1600
1800
2000





The parametrization is quite bad as it oscillates in the case of C sometimes and sometimes causes the agent to get stuck in a loop.