Assignment 2

Ajay Subramanian

July 2019

Written Assignment

Question 1

(a) TD(0)

- $v(A) = \frac{\frac{3}{5}+\frac{4}{5}+\frac{9}{5}}{3} = \frac{16}{15}$
- $v(B) = \frac{3}{5}$
- $v(C) = \frac{4}{5}$

MC

- $v(A) = \frac{4}{3}$
- $v(B) = \frac{3}{5}$
- $v(C) = \frac{4}{5}$

(b) MDP Diagram

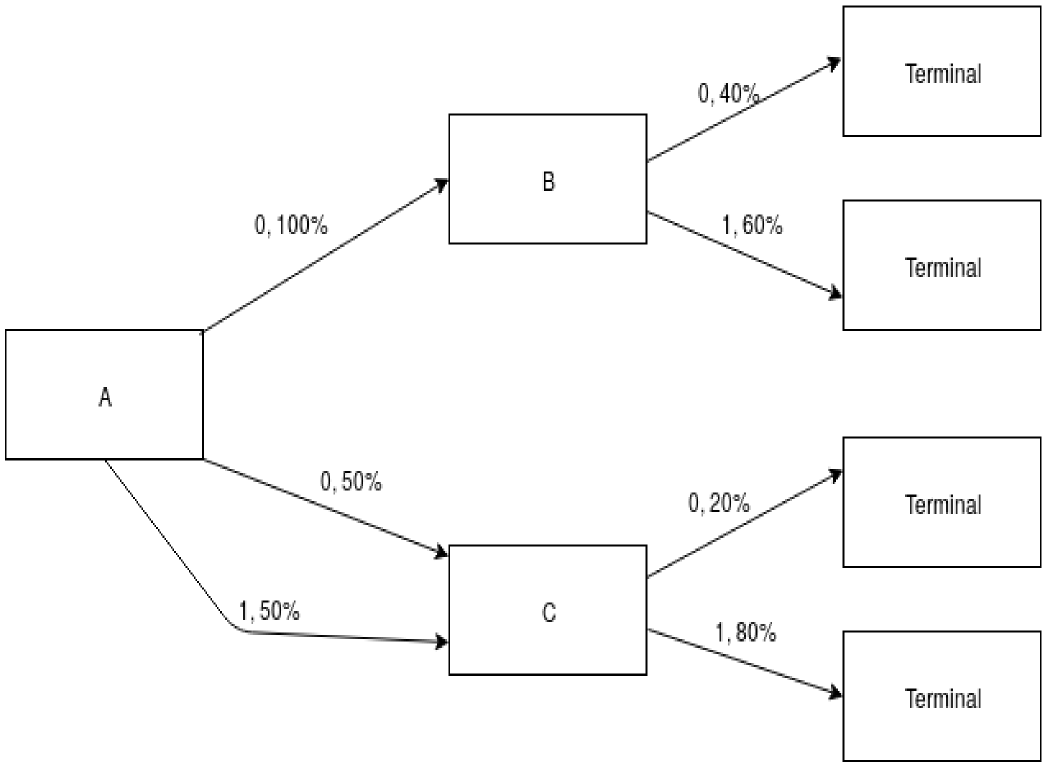


Figure 1: Markov Decision Process for the given situation

(c) TD(0)

$$\frac{1}{3}\left[2\left(\frac{16}{15}\right)^2+\left(\frac{1}{15}\right)^2\right]+\frac{1}{5}\left[3\left(\frac{2}{5}\right)^2+2\left(\frac{3}{5}\right)^2\right]+\frac{1}{5}\left[4\left(\frac{1}{5}\right)^2+\left(\frac{4}{5}\right)^2\right]=1.16$$

MC

$$\frac{1}{3}\left[\left(\frac{1}{3}\right)^2+2\left(\frac{4}{3}\right)^2\right]+\frac{1}{5}\left[3\left(\frac{2}{5}\right)^2+2\left(\frac{3}{5}\right)^2\right]+\frac{1}{5}\left[4\left(\frac{1}{5}\right)^2+\left(\frac{4}{5}\right)^2\right]=1.6222$$

TD is truer to the training data for this problem since its MSE is lower. For large amoounts of training data, MC will produce a lower MSE than TD(0).

(d) TD(0) is truer to the Markov assumption since we use bootstrapping which implicitly creates a dependency between the current state and the next. In MC, the value function is found by averaging returns from multiple sample trajectories. Hence, it will more closely fit the training data.

(e) Since the problem is Markovian in nature, TD(0) will produce a lower error on future data. This is because uses the expected reward for updates rather than samples, thereby reducing the chance of updating with outlier reward values.

Question 2

(a)

$$\begin{aligned} G_t =& \left(\frac{R_{t+1} + R_{t+2} + ... + R_{t+\tau}}{\beta} + R_{t+\tau+1}\right) + \\ & \gamma\left(\frac{R_{t+\tau+2} + R_{t+\tau+3} + ... + R_{t+2\tau}}{\beta} + R_{t+2\tau+1}\right) + \\ & \gamma^2\left(\frac{R_{t+2\tau+2} + R_{t+2\tau+3} + ... + R_{t+3\tau}}{\beta} + R_{t+3\tau+1}\right) + ... \\ & ; \beta \geq 1, \gamma \leq 1 \end{aligned}$$

It can be observed that this formulation is a modified version of τ-step truncated return. Here, β is used to give a higher importance to the reward at timestep $\tau + 1$ since that's when the action is applied to the environment.

(b) $v_{new}(s) = v(s) + \alpha \left[\frac{R_{t+1}+R_{t+2}+...+R_{t+\tau}}{\beta} + \gamma v(s_{t+\tau}) - v(s_t)\right]$

Question 3

(a)

$$E_t(s) = \begin{cases} max(E_{t-1}(s) - \gamma\lambda, 0) & s_t \neq s \\ E_{t-1}(s) - \gamma\lambda + 1 & s_t = s \end{cases}$$

(b)

$$E_t(s) = \begin{cases} max(E_{t-1}(s) - \gamma\lambda, 0) & s_t \neq s \\ 1 & s_t = s \end{cases}$$

(c) The linear decay formulation gives importance only to recent states (until $t - \gamma\lambda$) and hence does not waste time on computation for states that temporally occur before that. But in the geometric decay case, the coefficient of all states would be non zero and hence not ideal for large state spaces (would give high variance). Another advantage is that the traces always decrease at the same rate hence making traces of different states more comparable.

Question 4

Using TD(0) in such a case would not be ideal. TD(0) works on the assumption that the next state's value function is solely influenced by the current state and hence holds this state responsible for its performance. This assumption is not true for non-Markovian environments. One way to solve this issue partially is by using a TD(λ) approach whence varying λ between 0 and 1 influences the strength of the Markovian assumption.

Question 5

$a < 0 \rightarrow$ straight to T1 for k

$$a \geq 0 \rightarrow \begin{cases} a = 0 & \text{straight to T1 for k} = 0 \\ 0 < a < 5 & \text{straight to T1 for k} = \frac{a}{5} \\ a \geq 5 & \text{keep oscillating between * and middle state for k} = \frac{1}{\sqrt{2}} \end{cases}$$

Question 6

Episode length = $M.K$ steps
Every K steps, the dynamics of the problem change i.e. $P(s', r|s, a)$ changes
Problem representation:
$s \in S \times M$
$a \in A$
Since the problem dynamics change every K timesteps, M times, before repeating, we introduce another parameter for the transition probability equation which represents the phase (of length K) from 0 to M-1 that the problem is currently in. Hence the equation can be written as $P(s', r|s, a, (t//K)\%M)$ where // represents an integer division and % represents a modulo operation.

Question 7

Yes. Q-learning can be made on-policy by sampling an action from current policy and using the obtained reward in the update, rather than seeking the optimal reward. It will take longer to converge since using the policy for action selection constrains exploration. Using an off-policy method enables us to evaluate using the optimal policy while exploring using the estimation policy. Yes. We can use importance sampling / weighted importance sampling which use only the ratio of the policy probabilities as a coefficient for the value update equation. This does not require any feedback from the environment/simulation model and hence enables us to compute it off-policy while executing the optimal policy.

Question 8

(a) state set = $\{L, S\}$
action set = $\{O \wedge I, O \wedge \neg I, \neg O \wedge \neg I, \neg O \wedge I\}$
Reward and state transition diagram:

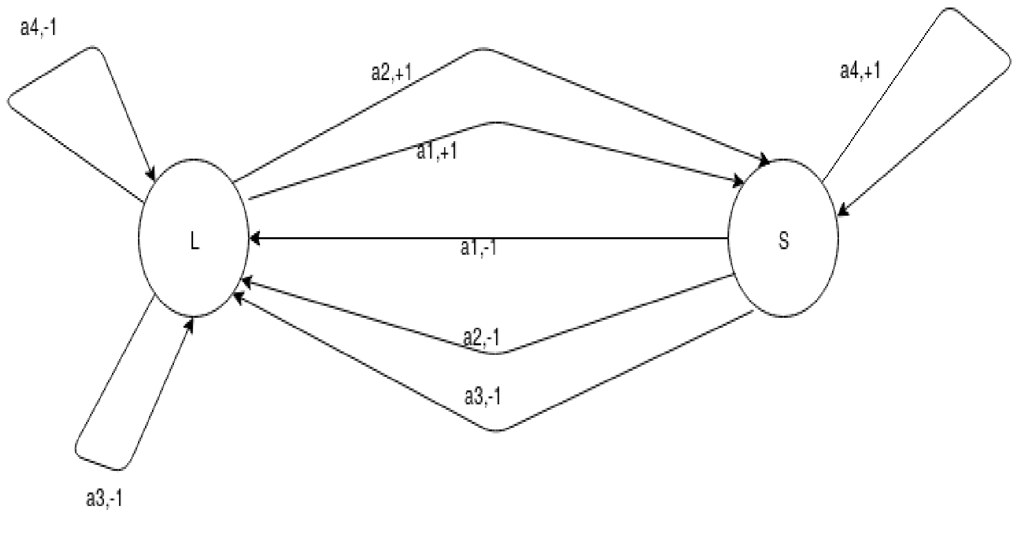


Figure 2: MDP with transitions for the given situation

(b) **Policy iteration:**

1)

$$
\begin{aligned}
v_\pi(L) &= -1 + 0.9(0) = -1 \\
v_\pi(S) &= 1 + 0.9(0) = 1 \\
\pi(L) &= \underset{a}{\text{argmax}}[1 + 0.9(1), 1 + 0.9(1), -1 + 0.9(-1), -1 + 0.9(-1)] = a1 \\
\pi(S) &= \underset{a}{\text{argmax}}[-1 + 0.9(-1), -1 + 0.9(-1), -1 + 0.9(-1), 1 + 0.9(1)] = a4
\end{aligned}
$$

2)

$$
\begin{aligned}
v_\pi(L) &= 1 + 0.9(1) = 1.9 \\
v_\pi(S) &= 1 + 0.9(1) = 1.9 \\
\pi(L) &= \underset{a}{\text{argmax}}[1 + 0.9(1.9), 1 + 0.9(1.9), -1 + 0.9(1.9), -1 + 0.9(1.9)] = a1 \\
\pi(S) &= \underset{a}{\text{argmax}}[-1 + 0.9(1.9), -1 + 0.9(1.9), -1 + 0.9(1.9), 1 + 0.9(1.9)] = a4
\end{aligned}
$$

Value iteration:

1)

$$
\begin{aligned}
v(L) &= max[1, 1, -1, -1] = 1 \\
v(S) &= max[-1, -1, -1, 1] = 1
\end{aligned}
$$

2)

$$
\begin{aligned}
v(L) &= max[1 + 0.9, 1 + 0.9, -1 + 0.9, -1 + 0.9] = 1.9 \\
v(S) &= max[-1 + 0.9, -1 + 0.9, -1 + 0.9, 1 + 0.9] = 1.9
\end{aligned}
$$

$$
\begin{aligned}
\pi(L) &= \underset{a}{\text{argmax}}[1 + 0.9(1.9), 1 + 0.9(1.9), -1 + 0.9(1.9), -1 + 0.9(1.9)] = a1 \\
\pi(S) &= \underset{a}{\text{argmax}}[-1 + 0.9(1.9), -1 + 0.9(1.9), -1 + 0.9(1.9), 1 + 0.9(1.9)] = a4
\end{aligned}
$$

(c) Optimal state action values are:
L - [2.71,2.71,0.71,0.71]
S - [0.71,0.71,0.71,2.71]

(d) If you hear laughter, blow organ. If its silent, don't blow the organ and burn incense.

Programming Assignment Report

Introduction

Two popular kinds of policy optimization algorithms are policy gradient algorithms, those that directly optimize the policy based on its performance on the environment, and value function based methods, those that use a value function to guide the policy towards preferred states and actions. Both these methods have their own set of advantages and disdvantages.

Policy gradients are usually better for high dimensional action spaces (since you can't estimate a value function for every possible case), and have guaranteed convergence). They work well with stochastic policies as well, while value function based methods cannot solve such problems. However, there is also the possibility of them converging to local maxima since they don't have the future-sight that value function based methods offer. Policy gradient methods are also harder to explore with since they're usually on-policy.
Value function based methods generally exhibit faster convergence and offer more flexibility in terms of exploration. However a big downside of these methods is their incapability in solving large/continuous action spaces, and environments where the optimal policy is stochastic. In fact, TD learning works on an inherent assumption that the environment can be solved by a deterministic policy!

SARSA and Q-Learning

Environment

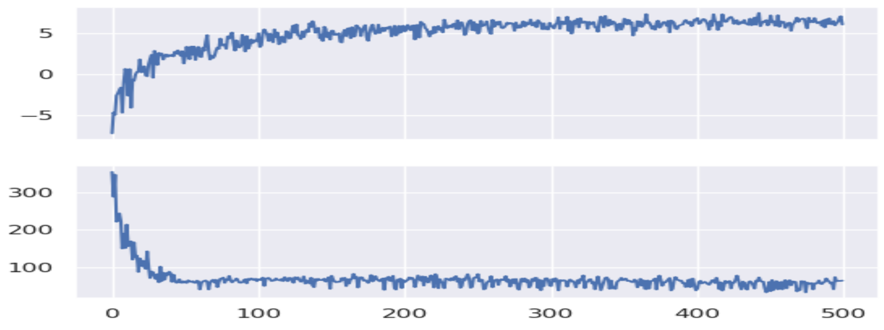
The two algorithms were compared on the puddle world environment (as described in the question) over 3 variants of the problem with different goals. C is more difficult than A and B since the goal is just on the boundary of the puddle making it harder to solve, given that all actions on the environment have a stochastic component to them.

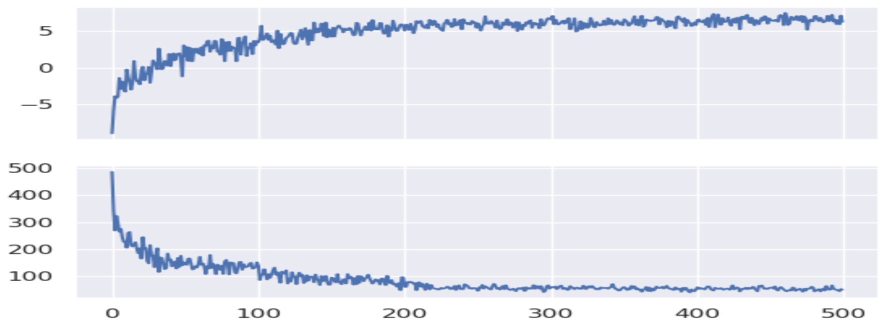
SARSA

SARSA was run 50 times with 500 episodes each, on each variant of the problem. The resulting trends in the cases of A and B were quite similar with the optimal policy being identified faster in A than in B. In C however, SARSA fails to find a policy that reaches the goal. This is because it takes into account the exploration while updating the value function. Hence it takes a safer routes and fails to converge. Its performance on such environments can be improved in two ways:

- Reducing the exploration coefficient: This will probably lead to convergence, but at the cost of exploring multiple policies.
- Introducing a small negative reward for every step it takes in the environment. I experimented with this on the C problem with a reward of -0.25 on every step. In this case SARSA converges (even faster than it did in A and B without the extra reward).

The plots for the observed trends are given below.





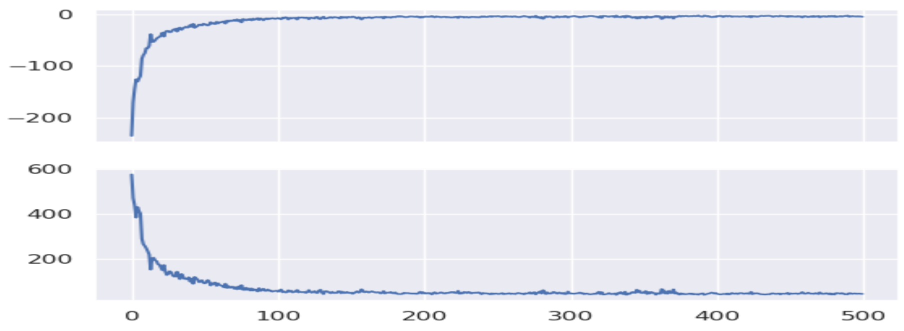
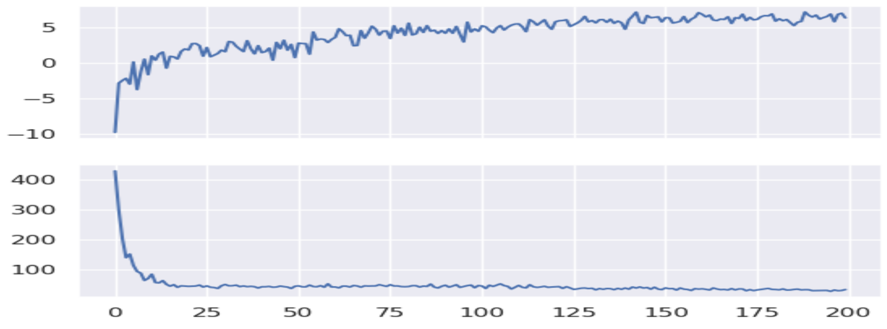
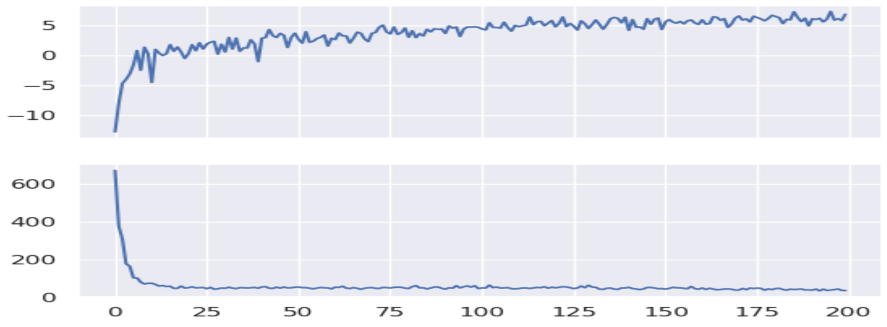


Figure 3: Reward-time and steps-time plots for SARSA on variants A,B and C of the problem.

Q-Learning

Q-Learning was run 50 times with 200 episodes each, on each version. On problems A and B it converges pretty fast , and slightly faster than SARSA does. But on problem C, it converges unlike SARSA, to the policy shown below. This is because it's value function updates are off-policy and hence not affected by the tendency of exploration. Hence it is more likely to find 'dangerous' yet successful policies, at the cost of more 'deaths' while training.





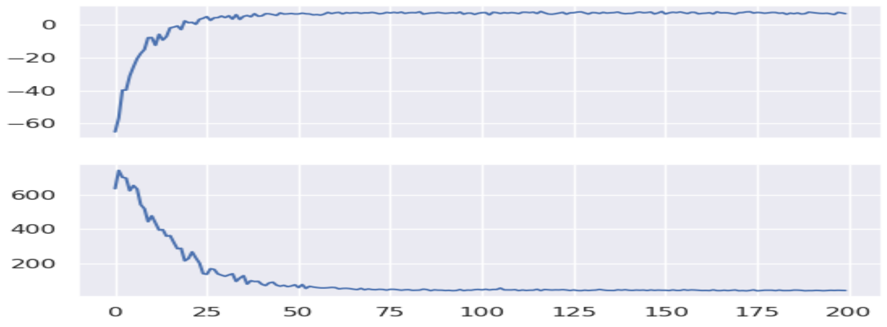
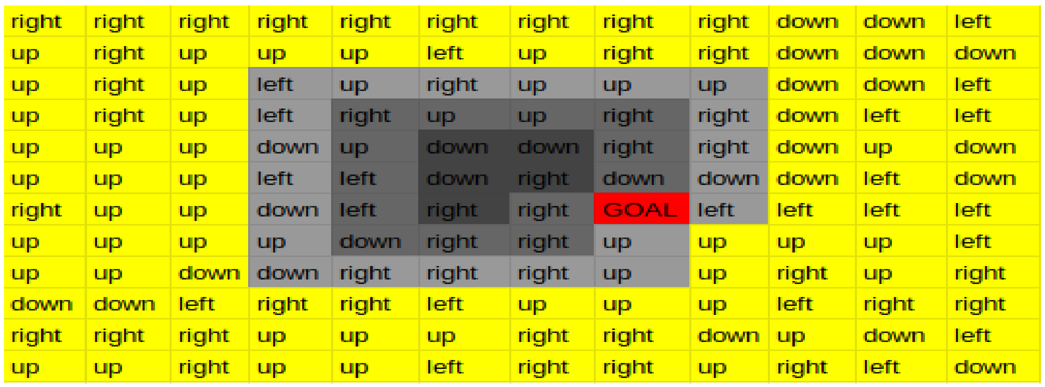
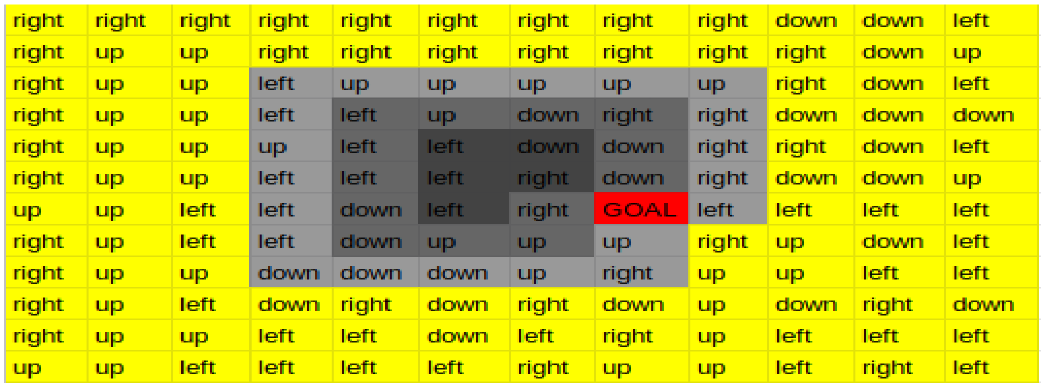


Figure 4: Reward-time and steps-time plots for Q-Learning on variants A,B and C of the problem.

Due to this difference in control equations, I looked into the learned policies of the two algorithms (with -0.25 reward in the SARSA case) on environment C. They are as shown below. We can see that, even having the per-step punishments in SARSA, Q-Learning identifies a slightly shorter policy. This is due to the reasons discussed above. Another interesting observation from these images, the learned policies on the southern half of the grid lead to the top half rather than along the southern boundary of the puddle. This is in fact a problem with TD methods (or Dynamic Programming methods in general) wherein their value function updates are biased on their initial learning parameters. So, once the northern states get a high value function, the optimal actions at the southern states will be towards the north, which is not ideal.



right	right	right	right	right	right	right	right	right	down	down	left
up	right	up	up	up	left	up	right	right	down	down	down
up	right	up	left	up	right	up	up	up	down	down	left
up	right	up	left	right	up	up	right	right	down	left	left
up	up	up	down	up	down	down	right	right	down	up	down
up	up	up	left	left	down	right	down	down	down	left	down
right	up	up	down	left	right	right	GOAL	left	left	left	left
up	up	up	up	down	right	right	up	up	up	up	left
up	up	down	down	right	right	right	up	up	right	up	right
down	down	left	right	right	left	up	up	up	left	right	right
right	right	right	up	up	up	right	right	down	up	down	left
up	up	right	up	up	left	right	right	up	right	left	down

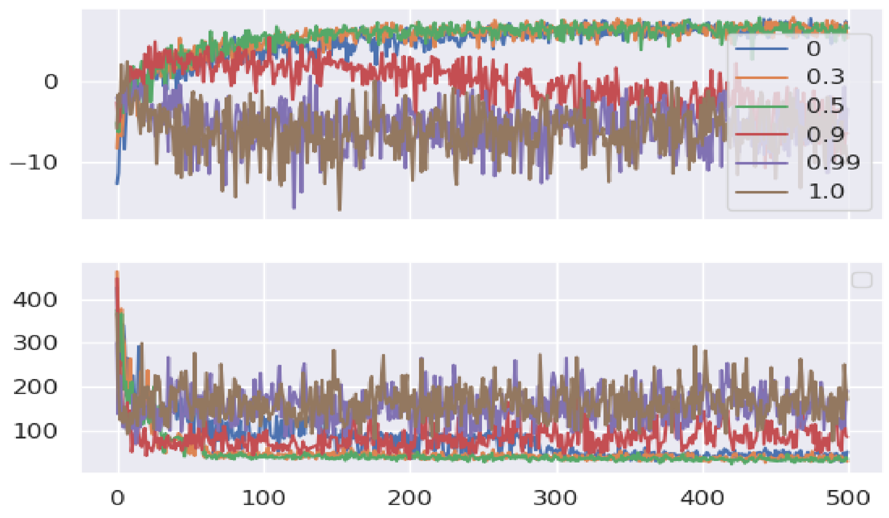


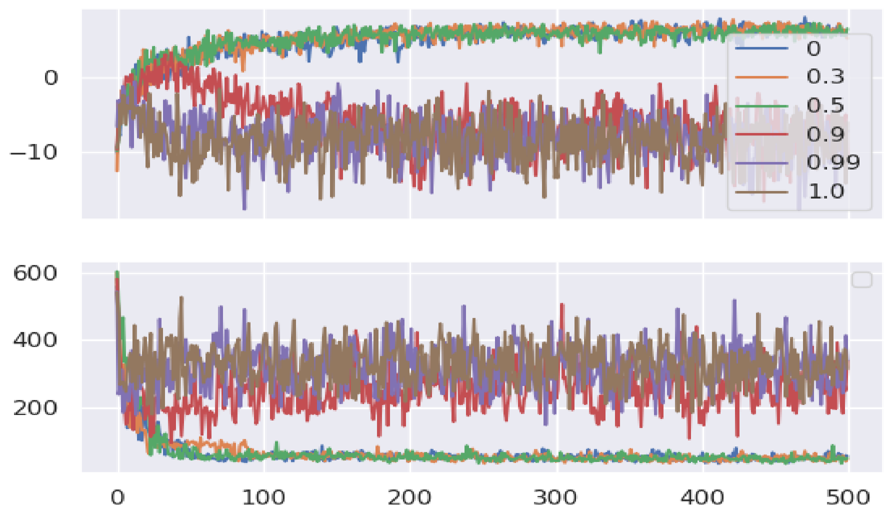
right	right	right	right	right	right	right	right	right	down	down	left
right	up	up	right	right	right	right	right	right	right	down	up
right	up	up	left	up	up	up	up	up	right	down	left
right	up	up	left	left	up	down	right	right	down	down	down
right	up	up	up	left	left	down	down	right	right	down	left
right	up	up	left	left	left	right	down	right	down	down	up
up	up	left	left	down	left	right	GOAL	left	left	left	left
right	up	left	left	down	up	up	up	right	up	down	left
right	up	up	down	down	down	up	right	up	up	left	left
right	up	left	down	right	down	right	down	up	down	right	down
right	up	up	left	left	down	left	right	up	left	left	left
up	up	left	left	left	left	right	up	up	left	right	left

Figure 5: Learned policies for SARSA (with punishments)[top] and Q-Learning [bottom]

SARSA(λ)

SARSA(λ) is a TD(λ) control algorithm that can be thought of as a spectrum from TD(0) control to MC control (by Forward view), dependent on the value of λ. The only aspect of its equations that is different from the TD(λ) evaluation algorithm is that the eligibility traces are computed for state-action pairs rather than just the state, since we want to optimize the policy. The SARSA(λ) algorithm has been run on the environment for each variant of the problem (C with punishments). Below are the results.


0
−10
0
0.3
0.5
0.9
0.99
1.0
400
300
200
100
0
100
200
300
400
500


0
−10
0
0.3
0.5
0.9
0.99
1.0
600
400
200
0
100
200
300
400
500

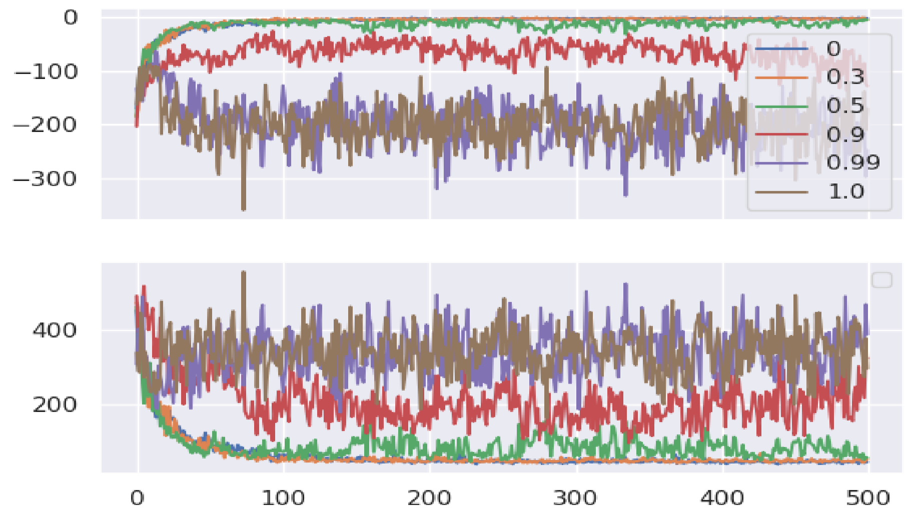


Figure 6: Reward-time and steps-time plots for SARSA(λ) for different values of λ, on variants A,B and C of the problem.

$\lambda = 0, 0.3, 0.5$ perform the best on all of the versions. This means that, for this environment, the optimal policy at any state is more dependent on temporally closer states i.e the environment is more towards the Markovian end of the spectrum.

Policy Gradients

The vanilla policy gradient (sample-based) was implemented on the chakra and vishamC environments given in the question. As mentioned earlier policy gradient algorithms are useful in problems with large/continuous action spaces. They also guarantee convergence (by Policy gradient theorem). The most popular way to implement exploration in policy gradients is to use a stochastic policy. Using stochastic policy formulations is also useful in cases where translation invariance is desired. Value function based methods cannot learn stochastic policies. The policy had a mean proportional to a set of learnable parameters θ and the current state. The variance was fixed at 1.0. A higher variance makes it harder to learn a good policy since there's a lot of noise between the decision and the action. After computing an action from the policy, the values along both axes was clipped to 0.025, before applying on the environment.

Policy gradient RL methods can viewed from a supervised learning angle too (from an Andrej Karpathy blog). For each parameter update, we take multiple trajectories (episodes in this case) and obtain the returns for each step. The trajectories are analogous to training data and the rewards, to labels (i.e. how good the agent performs). We then perform a parameter update based on the loss function computed on this batch of trajectories.

Why does the loss function have a logarithmic formulation? In the policy gradient algorithm, the loss function used is the expected return from a given state i.e product of probability of taking a trajectory, and corresponding return for each possible trajectory. The second term is easy to calculate directly. The first term reduces to a summed product of policy and transition probabilities. In order to obtain the value of the transition probabilities, we need a dynamics model, don't we? This is were using a log becomes useful. It reduces the product into a sum of two large terms; sum of policies and sum of transition probabilities. The second is constant wrt θ and hence disappears when we take the gradient! (from a Pieter Abbeel lecture)

Answers to Questions and Deliverables

1. In these environments the ideal training conditions are (from a few tests: small batch size (since pattern is simple), small learning rate (since there's a lot of variance in the action selection), high discount factor. Also the ideal trajectory length would be = episode length.

2. A value function for the learned policy would be inversely proportional to the distance norm (used for rewards; differs for chakra and vishamC) of the state.

3. The learned policy trajectories point in the direction towards the origin , perpendicular to the reward contour boundaries. But due to the variance in the policy, the observed motion is zigzag towards the centre with the component parallel to the boundaries cancelling out.