CS698R: Deep Reinforcement Learning

Assignment #1

Name: Samrudh BG
Roll NO.: 20128409

Solution to Problem 1: Multi-armed Bandits

1. The 2-armed Bernoulli bandit was created using a discrete action space of size 2, and a discrete observation space of size 3 with 2 terminal states (1,2) and 1 non terminal state(0). The agent starts off at state 0, and action 0 takes the agent to the state 1 with reward=1 and action 1 takes the agent to the state 2 with reward=1. However, this is not deterministic and the agent reaches the state 1 with a probability α and state 2 with a probability $1 - \alpha$ on taking action 0. Similarly, on taking action 1, the agent reaches state 2 with a probability β and state 1 with probability $1 - \beta$. The agent only receives the positive reward on reaching state 1 with action 0 and state 2 with action 1. The environment is formally defined as, R_0 $Bernoulli(\alpha)$ and R_1 $Bernoulli(\beta)$. The alpha and beta parameters can range between 0 and 1 and are taken as input for initialisation (default:(0.7,0.7)). The default seed used for the environment is 31 and remains so further below, unless specified otherwise.

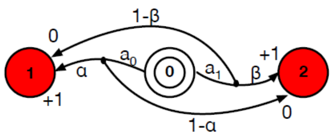


Figure 1: Two Armed Bernoulli Bandit.

The following values for the parameters were used in the simulation, (α, β) = (0,0); (1,0); (0, 1); (1,1); (0:5,0:5),

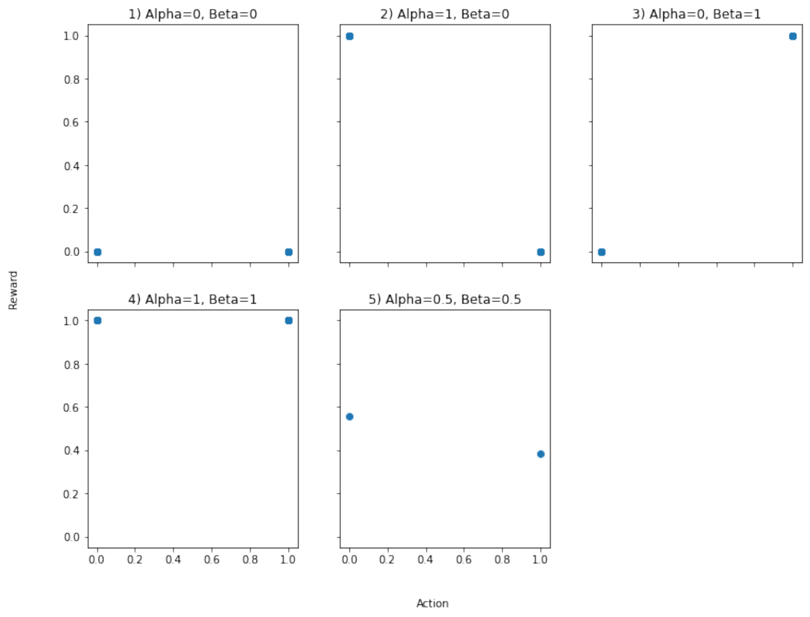


Figure 2: 2-armed Bandit test cases.

1) Since alpha=0 and beta=0, the agent will always end up going in the opposite direction of the proposed action. Therefore, action 0 will take the agent to 2 and give no reward (reward=0) and action 1 will the agent to 1 and give no reward.

2) Since alpha=1 and beta=0, the agent will always end up at state 1. This is because probability of going towards 2 is dependent on (1-alpha) and beta which are both 0. Therefore action 0 will end up with reward=1 and action 1 will end up with no reward.

3) Since alpha=0 and beta=1, the agent will always end up at state 2. This is because probability of going to state 1 is dependent on alpha and (1-beta) which are both 0. Therefore action 1 will end up with reward=1 and action 0 will end up with no reward.

4) Since alpha=1 and beta=1, the agent will always end up going towards the direction of the proposed action. Therefore both actions 0 and 1 will always have a reward=1.

5) Since alpha=0.5 and beta=0.5, there is an equal probability of landing at states 1 and 2 irrespective of what action is taken. Therefore, the expected long term reward will be 0.5 for each action.

2. The ten armed Bernoulli bandit was created using a discrete action space of size 10 and observation space of size 11. The environment consists of 1 non-terminal state (0) and ten terminal states (1-10). The agent therefore starts at state 0 and reaches any one of the non-terminal states (depending on the policy) per episode. A central Gaussian reward distribution is created with mean and standard deviation as input parameters (default: $\mu = 0, \sigma = 1$). Each of the ten terminal states also have a Gaussian reward distribution of their own with standard deviation=1.Ten samples are taken from the central distribution, and the ten samples serve as the mean for the terminal states' distributions. The seed was set to 31 by default for the environment and remains so elsewhere unless specified otherwise.

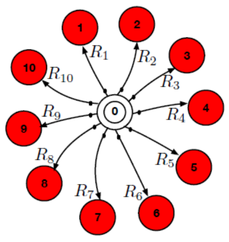


Figure 3: Ten Armed Gaussian Bandit.

The following values for the input parameters were used in the simulation (μ, σ)=(-20,0),(-20,2),(-20,5),(-20,10),(-20,50),(20,0),(20,2),(20,5),(20,10),(20,50).

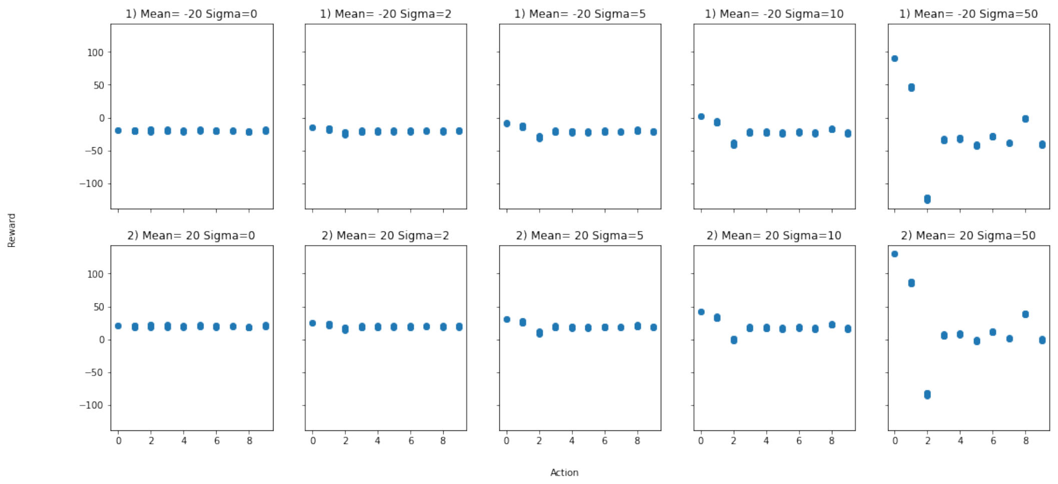
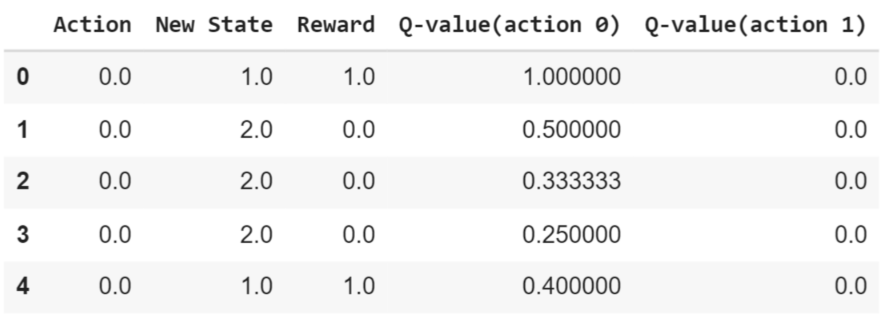


Figure 4: 10-armed Bandit test cases.

By increasing σ values, the variance in the reward received for each action also increases. This is because the reward distribution for each of the k terminal states is dependent on the central reward distribution $q^*(k)$. By changing the mean values of the central reward distribution, the average reward received from each terminal state increases and decreases accordingly.

3. Six action selection strategies were implemented and run in the 2 armed Bernoulli bandit environment. The results are presented below for each strategy.

   (a) The first strategy that was implemented was the **Pure exploitation (greedy) strategy**. In the strategy, the agent always selects the argmax of the current states' Q-values as the action to be taken in the environment.The parameter values used here are $(\alpha, \beta) = (0.7, 0.7)$.The maximum number of episodes was set to 5.



	Action	New State	Reward	Q-value(action 0)	Q-value(action 1)
0	0.0	1.0	1.0	1.000000	0.0
1	0.0	2.0	0.0	0.500000	0.0
2	0.0	2.0	0.0	0.333333	0.0
3	0.0	2.0	0.0	0.250000	0.0
4	0.0	1.0	1.0	0.400000	0.0

Figure 5: Pure exploitation strategy results.

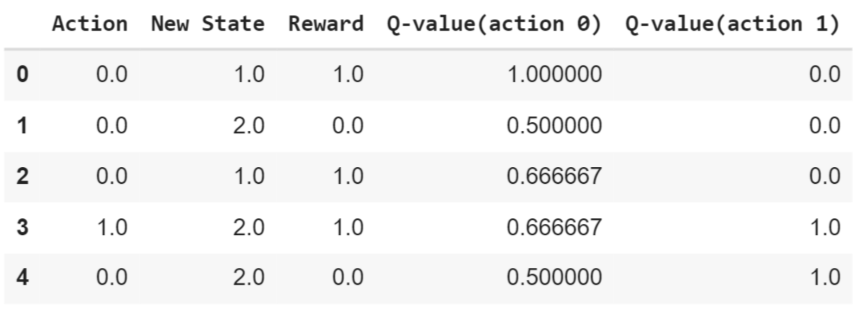
The agent initially took an action towards 0, reached state 1 and received reward=1. Therefore the $Q(a_0)$ is updated to 1. After the positive Q-value updation for action 0, since the agent is taking the argmax of the Q-values, the agent is now bound to take action 0 until $Q(a_0) < Q(a_1)$.

The agent then repeats the action but due to the stochasticity in the environment reaches state 2 and receives no reward. Therefore $Q(a_0)$ is updated to 0.5 ($\frac{\sum R_0=1}{N_e(a_0)=2}$).

In the third episode, the agent's action and outcome is repeated again and $Q(a_0)$ is updated to 1/3. The fourth episode has the same pattern of action and outcome and $Q(a_0) = 1/4$. In the final episode, the agent takes action 0 and reaches state 1 and receives reward=1. The final $Q(a_0)$=2/5 ($\frac{\sum R_0=2}{N_e(a_0)=5}$). Since a_1 is never selected, $Q(a_1)$ remains 0.

We can see that the Q-values of the agent [0.4,0] are far away from the actual expected Q-values [0.7,0.7]. The greedy strategy is therefore sub-optimal and the Q-values will not converge to the true values in many situations.

   (b) The second strategy implemented is the **Pure Exploration strategy**. In the strategy, the agent selects the action randomly for each episode with an equal probability for each of the available actions. The parameter values used here are $(\alpha, \beta) = (0.7, 0.7)$. The maximum number of episodes was set to 5.



	Action	New State	Reward	Q-value(action 0)	Q-value(action 1)
0	0.0	1.0	1.0	1.000000	0.0
1	0.0	2.0	0.0	0.500000	0.0
2	0.0	1.0	1.0	0.666667	0.0
3	1.0	2.0	1.0	0.666667	1.0
4	0.0	2.0	0.0	0.500000	1.0

Figure 6: Pure exploration strategy results.

The agent initially chooses action 1 and reaches state 1 and receives reward=1. Therefore, $Q(a_1)$ is updated to 1. The agent then repeats the same action by chance and due to the transition probabilities, reaches state 2 and receives a reward=0, $Q(a_1)$ is now updated to 0.5 ($\frac{\sum R_0=1}{N_e(a_0)=2}$). The agent then selects action 0 randomly, reaches state 1 and receives reward=1, $Q(a_0)$ is now updated to 2/3. The agent then chooses action 1 randomly and reaches state 2 and receives reward=1. The

$Q(a_1)$ is now updated to 1 ($\frac{\sum R_1=1}{N_e(a_1)=1}$). In the final episode, agent then chooses action 0 randomly and reaches state 2 with no reward. $Q(a_0)$ is now updated to $\frac{2}{4} = 0.5$.

(c) The third strategy implemented is the $\epsilon-$ **Greedy strategy**. The strategy is a combination of the above two strategies such that a random strategy is selected with $p(\epsilon)$ and a greedy strategy is selected with $p(1-\epsilon)$ in every episode. The epsilon value is taken as the input parameter in the function and the default value is set to 0.05.The maximum number of episodes was set to 1000.

Ten ϵ values were used in the simulation and were equally distributed between 0 and 1.The default values for (α, β) are used (0.7,0.7).

Two actions can be chosen in the environment (0,1). The difference between the actions chosen for each value of epsilon was calculated as, $\frac{abs(N_e(a_0)-N_e(a_1))}{N_e(a_0)+N_e(a_1)}$.

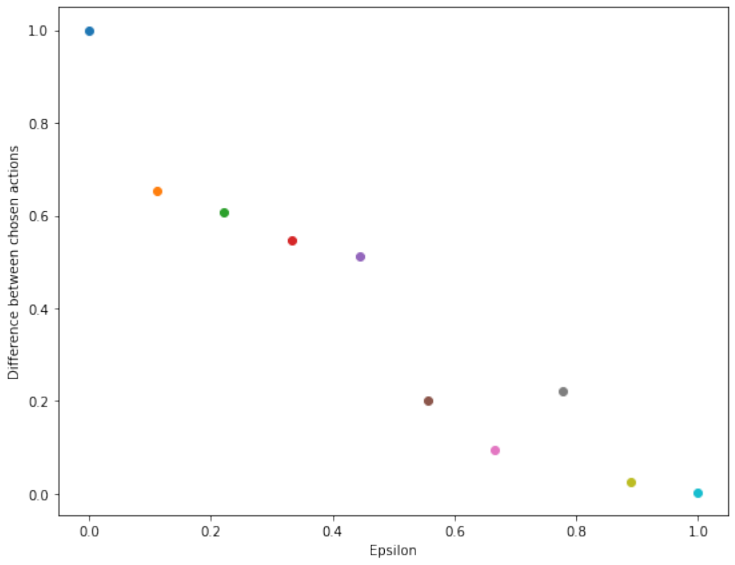


Figure 7: ϵ-Greedy strategy results.

The above plot shows that as epsilon increases, the difference between the chosen actions (1 and 2) reduces. An epsilon of 0 is synonymous to the always greedy strategy (pure exploitation) where only one option is almost always chosen while an epsilon of one is synonymous to the random strategy (pure exploration) where there is an equal probability of choosing both actions on every episode.

(d) The fourth strategy implemented is the **decaying** $\epsilon-$ **Greedy strategy**. In the earlier episodes where the agent has no knowledge of the environment, it makes more sense to explore the available options and evaluate the returns obtained from each. During the later stages once the agent has explored the environment sufficiently, it makes more sense to go to those states which have the highest rewards. Being greedy throughout might make the agent miss out on better opportunities while being random throughout prevents the agent from maximising rewards. Therefore the decaying $\epsilon-$ Greedy strategy ensures that the agent explores the environment more initially and gradually starts exploiting the maximum rewarding states.

Two types of decay are taken as an input parameter in the function, linear and exponential. The decay rate is also taken as an input parameter and the default is set to 0.05. The following plots were created with 1) Exponential decay, decay rate=0.05 and 2) Linear decay, decay rate=0.01 The agent was run for 100 episodes in the environment.

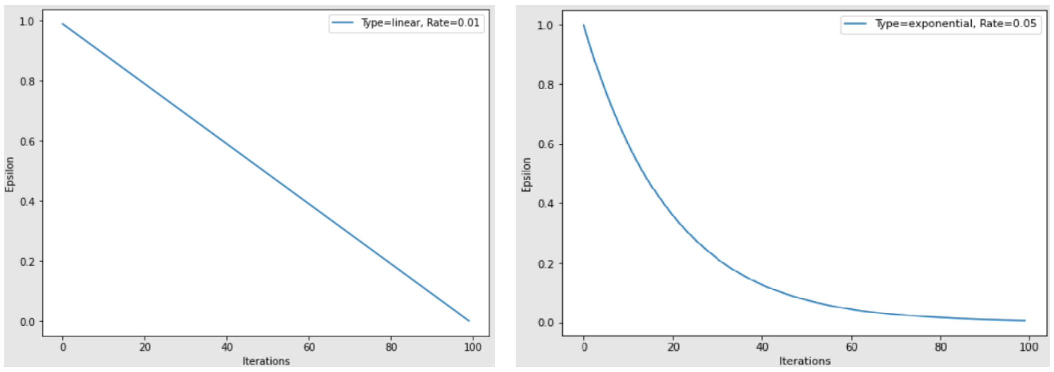


Figure 8: Decaying ϵ-Greedy strategy results.

(e) The fifth strategy implemented is the **Softmax strategy**. The strategy selects the action probabilistically, such that the $p(A)$ is dependent on $Q(A)$. A temperature hyper-parameter is also added such that for higher values of temperature, the Softmax probabilities are very similar for the actions and for lower values of temperature the converse is true . The formula for the policy using Softmax is $\pi(a) = \frac{\exp(\frac{Q(a)}{T})}{\sum_{b\epsilon A}\exp(\frac{Q(b)}{T})}$
The temperature was initialised to 1 and decayed linearly to 0.01 for 100 time steps. The difference between the chosen actions is calculated using the same formula as above. Default values for α and β were used here.

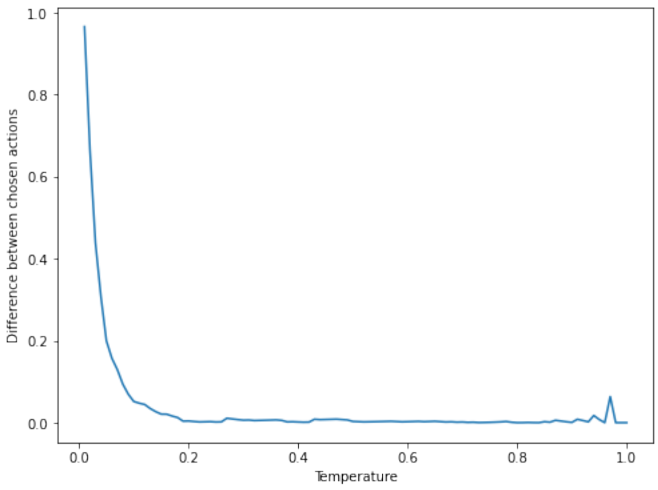


Figure 9: Softmax strategy results.

The plot shows that for smaller values of temperature there is a high difference between the chosen actions, synonymous to the exploratory approach. For higher values of temperature, there is a smaller difference between chosen actions, synonymous to the greedy approach. Decaying the temperature in such a way mimics the effects of the decaying epsilon strategy in the exploration vs exploitation trade-off. The change in difference between chosen actions is also exponential for the current environment.

(f) The sixth strategy implemented is the **UCB strategy**. The UCB strategy considers the uncertainty in decision making while selecting actions. Uncertainty is quantified as $c\sqrt{\frac{\ln e}{N_e(a)}}$ and added to the estimated Q values at each episode. The argmax of the sum of Q-values and uncertainty is taken as the selected action. Uncertainty is dependent on the number of times an action is taken such that when an action is selected a smaller number of times, the uncertainty would increase and vice-versa. Adding the uncertainty bonus helps the agent select the different actions a sufficient number of times. The c hyper-parameter helps decide the impact of uncertainty in the decision making process. The default c value is set to 0.5.
Eleven c values are used in the simulation and are uniformly distributed between 0 and 1. The maximum number of episodes was set to 100. The difference between the chosen actions is calculated

using the same formula used previously. Default values for α and β were used here.

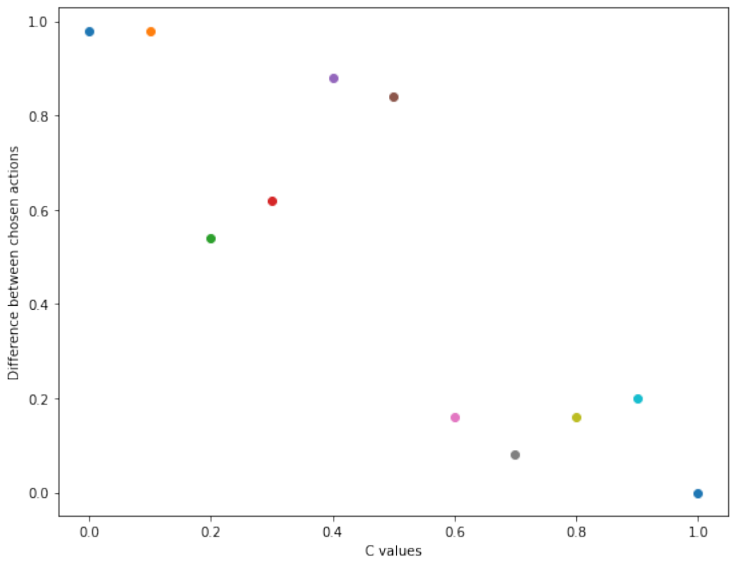


Figure 10: UCB strategy results.

When $c = 0$,there is no uncertainty added to the Q values and the action with the maximum Q-value is always chosen. The UCB strategy now mimics the greedy strategy (pure exploitation). As the weightage to uncertainty is increased through increasing the c value, both actions are chosen with greater frequency and the function approaches the always random paradigm.

4. α and β values were sampled from a uniform distribution fifty times. The following values were obtained using a seed of 31:

   α=array([0.61471817, 0.71962363, 0.22320241, 0.4483353, 0.05817954, 0.88157663, 0.86203608, 0.60524667, 0.72031531, 0.83853032, 0.03890314, 0.0750046 , 0.64680219, 0.51844817, 0.05944246, 0.49668287, 0.39962314, 0.4618744 , 0.96373968, 0.25780816, 0.57074063, 0.05256957, 0.83381663, 0.21977808, 0.45591319, 0.7125237 , 0.08273294, 0.6974497 , 0.09254781, 0.15187653, 0.38373962, 0.72069634, 0.92920944, 0.41075802, 0.50095043, 0.09315417, 0.77169725, 0.72412309, 0.68031976, 0.65444124, 0.04805003, 0.07437124, 0.51131814, 0.63497384, 0.19623165, 0.17040153, 0.99196725, 0.41272464, 0.55010062, 0.7121246])

   β= array([0.46417449, 0.24894652, 0.16032079, 0.04626118, 0.1763093 , 0.59167691, 0.44082034, 0.60332425, 0.87581628, 0.81527974, 0.21740816, 0.59440562, 0.4219051 , 0.75530295, 0.61070866, 0.1722158 , 0.31549807, 0.72953347, 0.03382733, 0.1737228 , 0.46807463, 0.62482716, 0.93452347, 0.90165523, 0.53705651, 0.71998815, 0.64156466, 0.64144008, 0.64131673, 0.98243316, 0.9050769 , 0.50194762, 0.48970175, 0.26372871, 0.588701 , 0.41978833, 0.26325977, 0.50636844, 0.59565725, 0.28464933, 0.46073129, 0.9717104 , 0.34840039, 0.20420903, 0.64517641, 0.6410167 , 0.31472018, 0.26608609, 0.02516422, 0.00919859])

   A combination of α and β values were supplied as input parameters for the six algorithms and were run for 1000 episodes each. The reward received per episode was averaged out over the 50 iterations and the averaged reward was plotted for each episode. The following plot was obtained using the default hyper-parameter values mentioned previously.

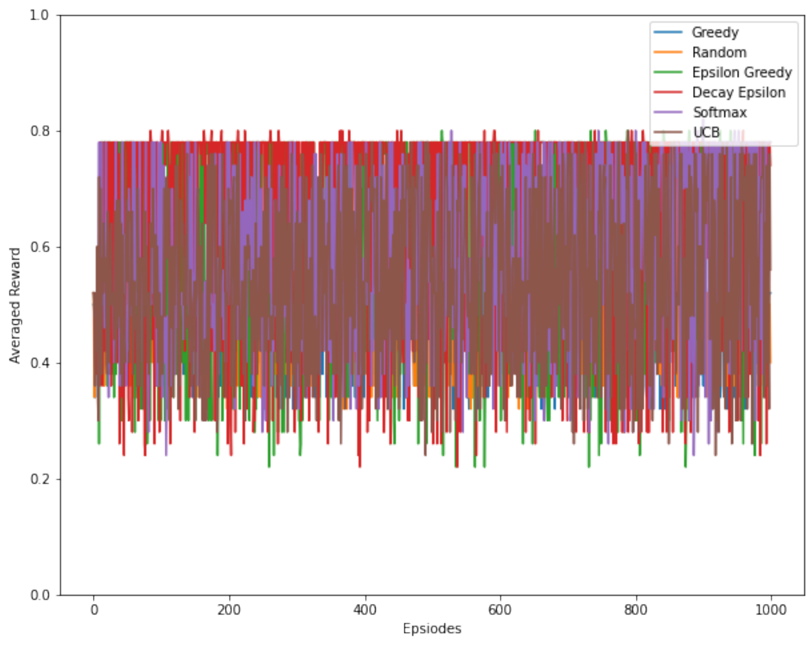


Figure 11: Averaged reward for the six strategies (2 Armed Bandit).

There seems to be a large amount of variance in the plots with all strategies performing fairly similarly. Further hyper-parameter tuning would be required for a better analysis of performance.

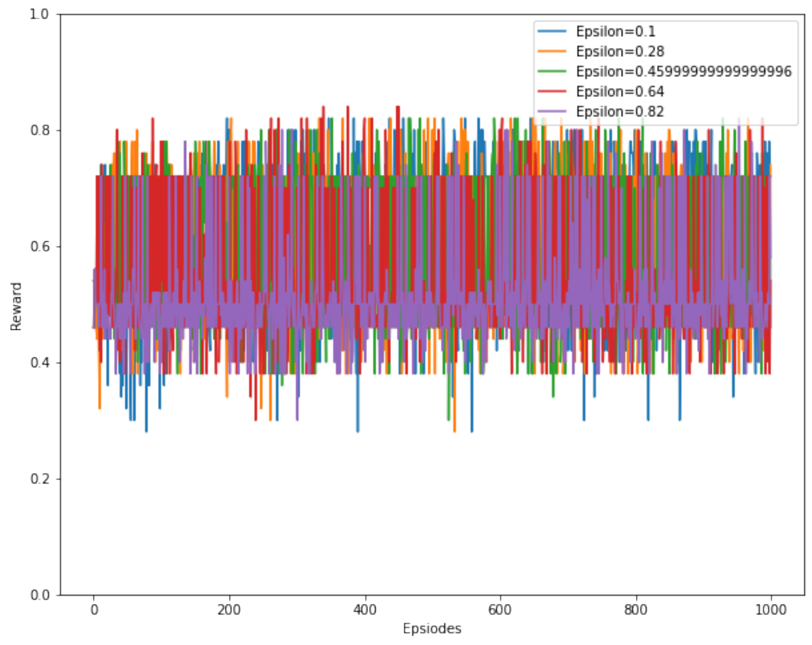


Figure 12: Tuning of epsilon values.

(a) Looking at the values for epsilon it is evident that larger values have lower performance. The variance also seems to be large for all the values. Epsilon values in the middle range seem to be performing ideally. This can be due to low values performing with an always greedy approach and high values performing with an always random approach which may not be optimal.

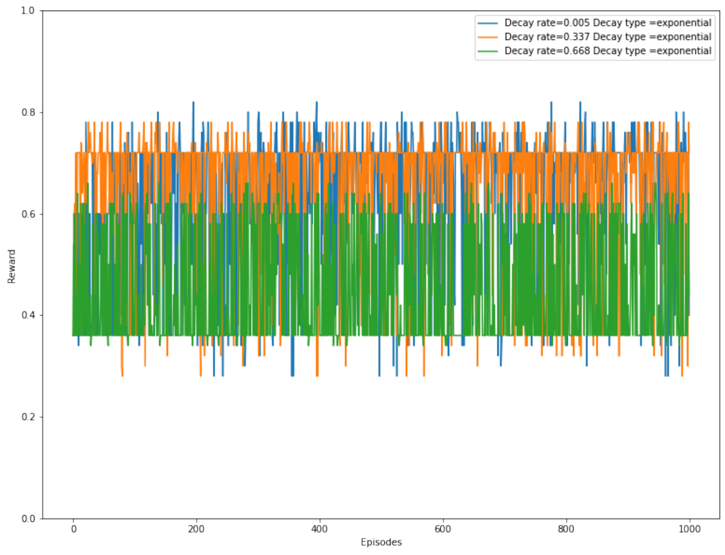


Figure 13: Tuning of decay rates.

(b) Looking at the values for the decay rates, it is evident that lower rates perform better than higher rates with a drop in performance for higher rates. This could be due to the fact that for higher decay rates, epsilon will decay very quickly and the agent will not get a sufficient number of time steps to explore the environment.

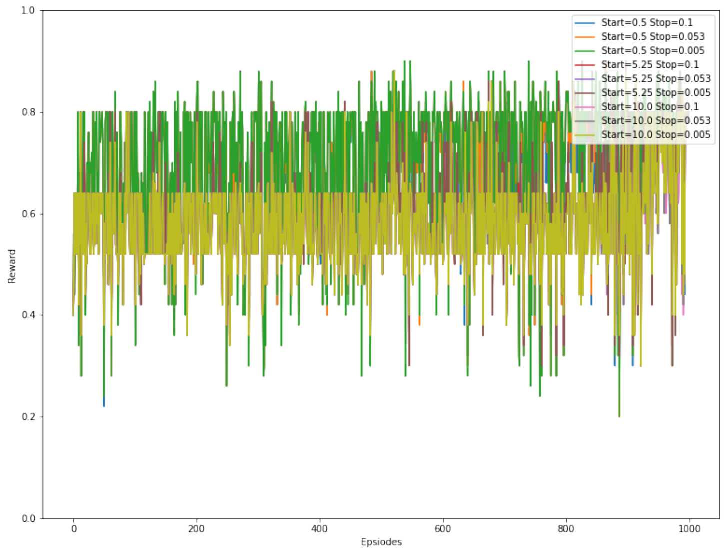


Figure 14: Tuning of temperature values.

(c) The maximum and minimum values for temperatures was manipulated and it is seen that very small minimum values increase variance and an ideal starting values for the temperatures are around 1 and stopping values are around 0.05.

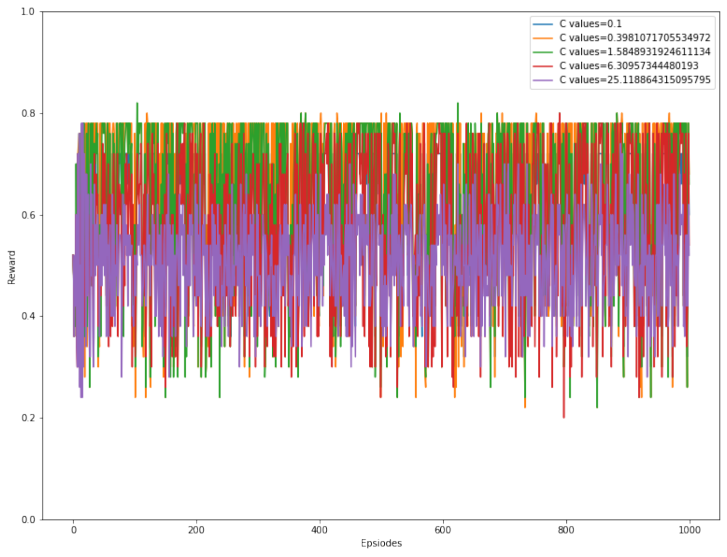


Figure 15: Tuning of c values.

(d) For the current environment, ideal c values seem to lie between 0.1-1 and higher c values seem to cause a decrease in performance and increase in variance. Very high c values could cause random choices as Q values will range between 0 and 1 and uncertainty could overpower the importance of Q-values of the states.

An overall comparison of the strategies seem to suggest that using the optimal hyper-parameter values would cause the agent to hover around an averaged reward of 0.6-0.8. There is no clear evidence that one technique is performing much better than the other.

5. A similar format was followed to compare the strategies on the 10 armed bandit environment. 50 Mean values were sampled from a uniform distribution ranging from 0 to 1. The following mean values were obtained,
μ= array([0.66222663, 0.19591212, 0.0361948 , 0.87614122, 0.5153978 , 0.03630033, 0.52447228, 0.18095247, 0.76337228, 0.51241942, 0.77236872, 0.56772826, 0.0360072 , 0.23922614, 0.95351221, 0.1622063 , 0.77319626, 0.03722349, 0.26599721, 0.00941974, 0.8679402 , 0.1205254 , 0.50385285, 0.08609814, 0.46497075, 0.9568295 , 0.54522058, 0.1211234 , 0.89790223, 0.05055027, 0.90697382, 0.42873311, 0.40640575, 0.76834406, 0.47375092, 0.81741927, 0.20304699, 0.08274335, 0.56732047, 0.40808253, 0.53652813, 0.78251042, 0.53957628, 0.62191418, 0.04084702, 0.86067441, 0.21787194, 0.25609105, 0.60616868, 0.36846659])

The rewards received were averaged over the 50 mean values for each of the 1000 episodes.

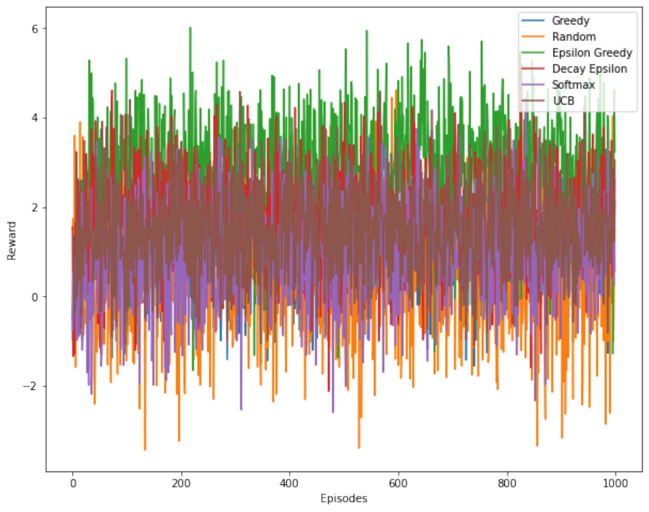


Figure 16: Averaged reward for the six strategies (10 Armed Bandit).

It is much easier to discern the performance of the techniques in the 10 armed bandit. The strategies seem to perform in the following hierarchy in descending order or performance: $\epsilon-$ greedy, decaying ϵ greedy, UCB, Softmax, greedy and random.

6. Regret is a measure of how well the agent could have done, by taking the optimal action instead of the action taken at the particular episode. It is measured as the difference between the optimal v value, (v_*)) or the true q value of the best action (argmax $(q_*(a))$, and the true q value of the action taken in that episode $(q_*(A_e))$. Since the agent is unaware of the true q values, it must learn them through experience. The most efficient agent would learn the true q values the fastest and take the optimal actions the earliest, having minimum regret.

   The true q values for the 2 armed bandit are synonymous to the α and β values and the optimal action is the maximum of the α and β values for the particular environment. Using this, regret was calculated for the six strategies depending on the action the agent took in each episode.

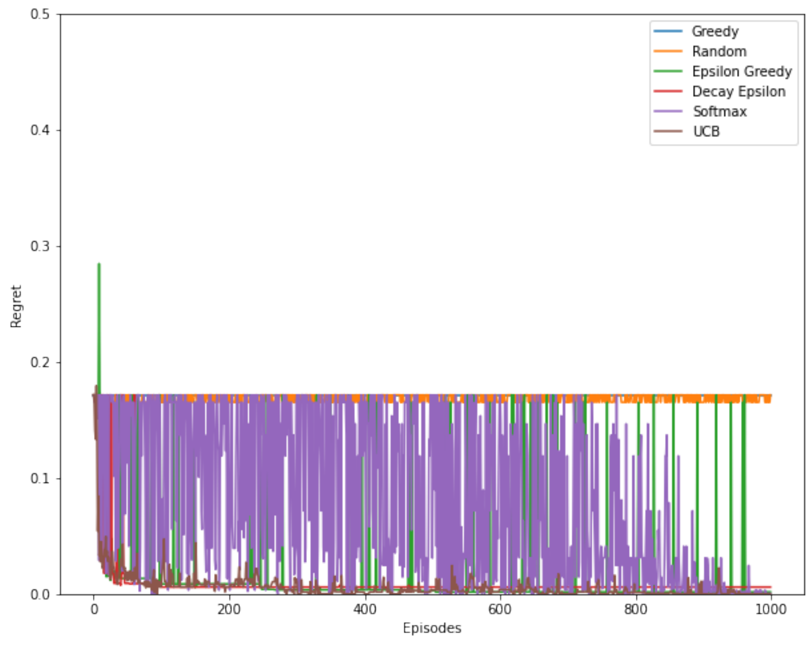


Figure 17: Regret over time (2 Armed Bandit).

   Since the pure exploration strategy selects actions randomly, it is logical that regret would not decrease as the action selection is not dependent on the q-values. Since greedy selection will always end up taking action 0 in the current environment, regret will be constant for the strategy. The $\epsilon-$ greedy strategy's regret values will fluctuate as well as actions are selected randomly with $p(\epsilon)$. The Softmax strategy seems to have high regret values initially but eventually decreases. This can be due to the exploratory nature induced initially due to the high temperature values which eventually decrease, allowing for exploitation. The $\epsilon-$ decay strategy seems to work much better and finds the optimal action quickly, minimising regret. The UCB strategy seems to be working well by minimising regret very quickly as well.

7. In the 10 Armed Gaussian Bandit, the actual q values for the actions were obtained through the central reward distribution for the environment. After obtaining the actions taken by the agent in each episode one-hot encoding was used to arrive at the true q values for the actions taken. Regret was calculated using the same formula as above and the values over time were obtained for the six strategies.

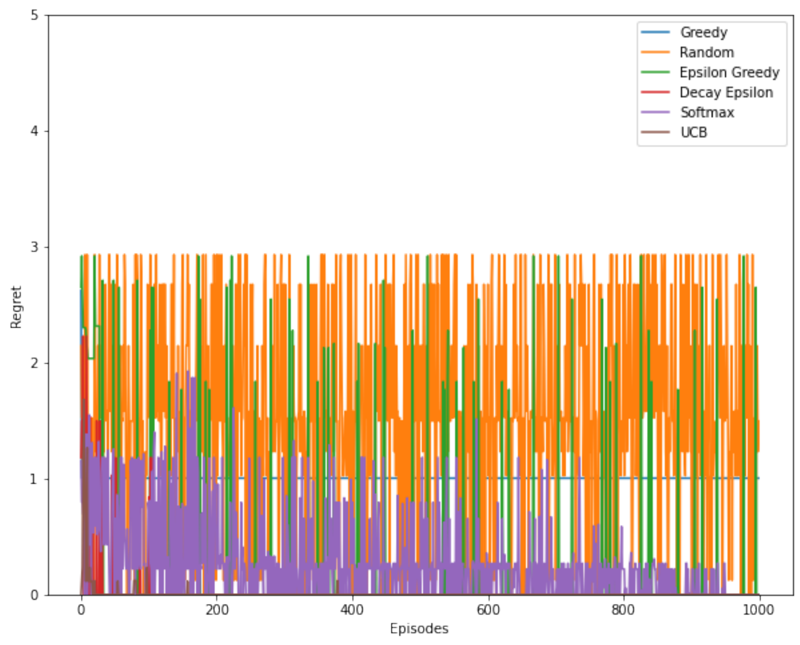


Figure 18: Regret over time (10 Armed Bandit).

Similar trends are observed in the 10 armed bandit environment compared to the 2 armed bandit for the different strategies. The Softmax strategy however seems to be minimising regret much faster in the 10 armed bandit condition compared to the 2 armed bandit. The greedy and random strategies do not display any learning as expected, and the $\epsilon-$ greedy strategy seems to randomly choose actions which are sub-optimal, presumably due to the random selection induced by the ϵ value. The $\epsilon-$ decay and UCB strategies seem to be performing optimally by minimising regret quickly. The plot is further evidence that an epsilon decay strategy works much better than an epsilon decay strategy, considering that the decaying parameters are selected appropriately.

8. The percentage of optimal actions was calculated by finding the number of times the agent took the optimal action divided by the total number of episodes.

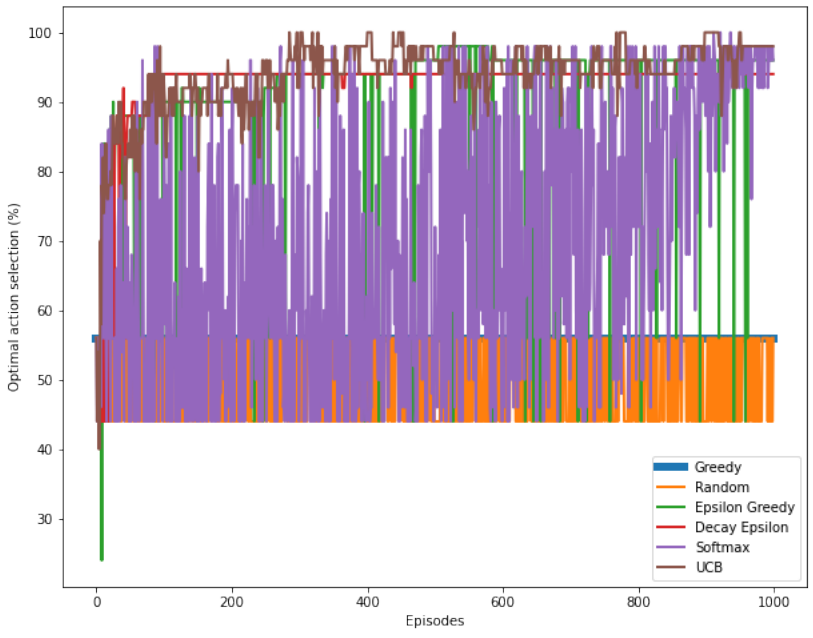


Figure 19: Optimal actions selected over time (2 Armed Bandit).

The regret and optimal actions plots are related and the random and greedy strategies seem to display no increase in the percentage of times they take the optimal action over time. The $\epsilon-$ greedy strategy shows a steep ascent but randomly drops off in certain episodes due to the stochasticity in action selection. The softmax strategy seems to display a slow increase in accuracy over time, by selecting the optimal action more frequently in the later episodes. A lower initialisation and/or quicker decay of temperature values seem to make sense in the current simplistic environment, to prevent excess exploration. Both the $\epsilon-$decay and UCB strategies quickly move towards the maximum accuracy and are performing the best out of the six strategies.

9. The percentage of optimal actions taken by the agent was calculated using a similar measure as mentioned above.

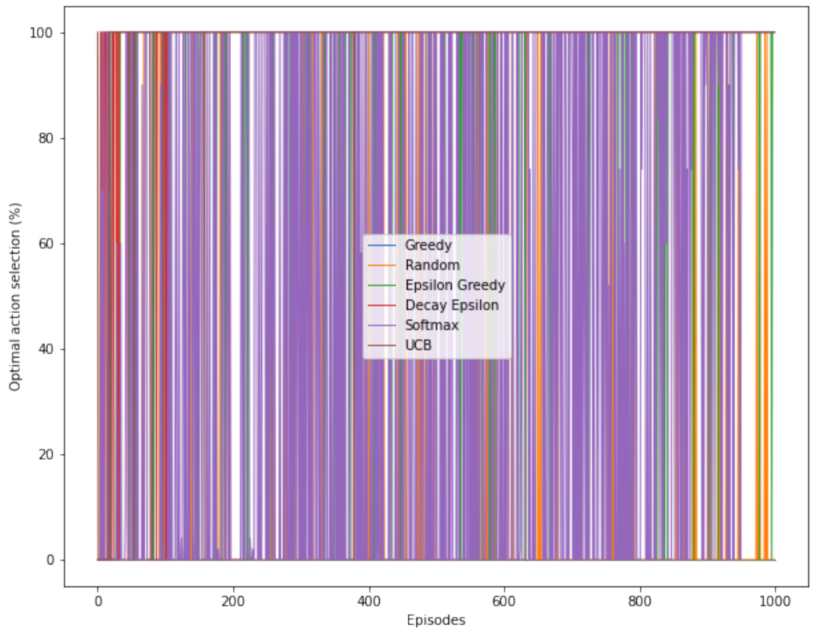


Figure 20: Optimal actions selected over time (10 Armed Bandit).

Although the high variance due to the complexity in the environment makes the plot slightly difficult to read, it is evident that the optimal actions chosen follows the regret over time. The random, greedy strategies seem to have high variance throughout while the softmax strategy's variance in the optimal actions chosen decreases towards the later episodes. The $\epsilon-$ greedy strategy has random bursts of variance across episodes. The UCB and $\epsilon-$ decay strategies seem to be performing highly in the current environment as well as they quickly reach higher percentages of optimal action choices.

Solution to Problem 2: MC Estimates and TD Learning

Random Walk Environment

The random walk environment was created using a discrete action space of size 2, where action 0 would take the agent to the left state and action 1 would take the agent to the right state. The observation space used in the environment was a discrete space of size 7 which consisted of 2 terminal states (0 and 6) and 5 non-terminal states (1-5). The starting state in the environment was set to state 3.

Irrespective of the action taken by the agent, the agent has an equal probability to land up in the state to the right or left of the current state. The agent receives a reward of 1 only on reaching the state terminal state 7 and receives a reward of 0 in all other states.

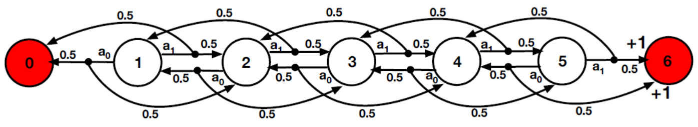


Figure 21: Random walk environment.

1. To implement the RL algorithms, first a helper function was created to generate the trajectory for every episode. The function generates a experience tuple containing (s,a,r,s') for every step in the episode, where s-current state, a-action, r-reward, s' -new state. The function is also capped at a certain number of maximum steps where if the episode runs for more than the specified steps, it is terminated, the current trajectory is discarded and an empty list is returned.

   The environment is initialised at state 3 and the maximum number of steps is set to 50. An example trajectory is generated with a policy of only left (action=0). The function terminates and returns the trajectory when the agent reaches state=0.

   [[3, 0, 0, 4], [4, 0, 0, 5], [5, 0, 0, 4], [4, 0, 0, 5], [5, 0, 0, 4], [4, 0, 0, 3], [3, 0, 0, 2], [2, 0, 0, 1], [1, 0, 0, 0]]

   With a policy of only right (action=1), the agent reaches state 6 and the function terminates, returning the following trajectory:

   [[3, 1, 0, 2], [2, 1, 0, 3], [3, 1, 0, 4], [4, 1, 0, 5], [5, 1, 1, 6]]

   Since there is an equal probability of going either left or right, the policy is irrelevant in this environment.

2. In both MC and TD Learning algorithms, the learning rate(alpha) plays an important role as it decides the rate at which the values for the different states are updated. We decay the learning rate from a higher to a lower value, such that there is faster learning in the earlier episodes and slower learning in the later episodes. This reduces the variance and helps the values for the states converge to the true values. Both linear and exponential decay of the learning rate were implemented in the function.

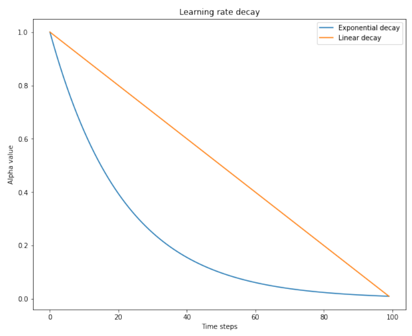


Figure 22: Exponential and linear alpha decay.

3. The Monte Carlo algorithm is used in situations where we do not have prior knowledge of the underlying MDP dynamics. The value of a state, V(s) is the expected return from a state (s), where the return is the sum of reward received. The algorithm approximates the value of a state (s) by drawing a large number of sample trajectories from the state (s) and taking the average of all the returns.

$$V_{e+1}(s) = \frac{G_1 + G_2 + G_3 .... + G_n}{N_s(e)}$$

The implemented Monte Carlo algorithm incrementally changes the value for a state (s) per episode. It does this by calculating the MC error, or difference between the received return in the episode and current value of the state. The MC error is then multiplied by the learning rate and added to the current value for the state.

$$V_{e+1}(s) = V_e(s) + \alpha(e)[G_e - V_e(s)]$$

There are two variants of the Monte Carlo algorithm, First Visit Monte Carlo (FVMC), where only the first visit to a state (s) in an episode is considered and Every Visit Monte Carlo (EVMC), where every visit to a state (s) in an episode is considered.

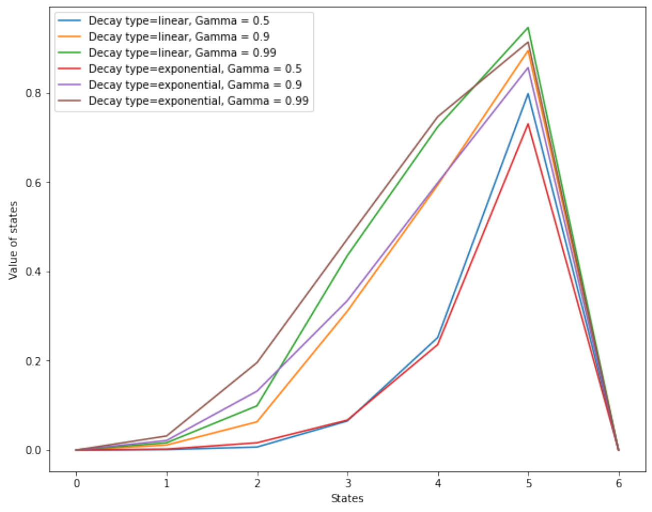


Figure 23: Test cases for FVMC algorithm

As we can see, the values for the states 0 and 6 are 0. This is because they are terminal states and the expected return in these states is 0. The decay type and gamma values were manipulated here. As gamma increases, the valuation of the states also increase. We also notice that the values for exponential decay learning rates are slightly lower compared to linear decay. This is because, as seen above, the exponential decay rates drop much faster than linear decay rates and therefore the value updation becomes smaller much quicker. The seed used here is 39 and maximum number of episodes is 500.

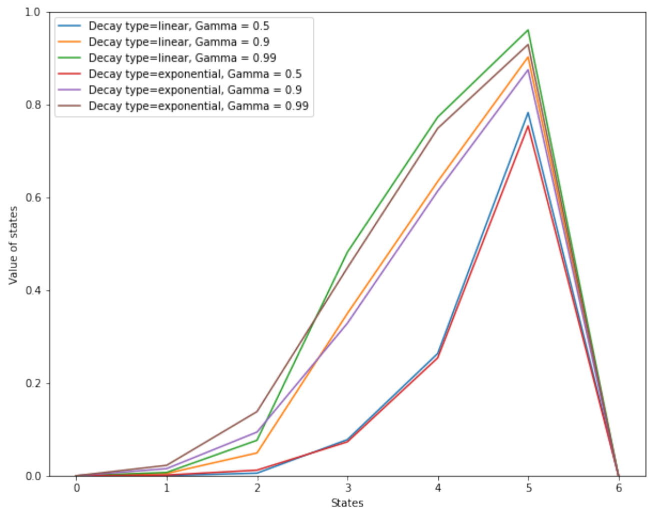


Figure 24: Test cases for EVMC algorithm

A similar plot is seen in the EVMC condition using the same hyper-parameter and seed values.

4. The temporal difference algorithm functions differently such that it approximates the value of a state, V(s) by using the reward received at the next state and the approximated value of the new state V(s') at every time step(t). The difference between the target value and current value of the state is the TD error. The TD error is multiplied by the learning rate and added to the current value of the state at every time step, to get the new value, V t+1(s).

$$V_{t+1}(s) = V_t(s) + \alpha(e)[R_{t+1} + \gamma V_t(s+1) - V_t(s)]$$

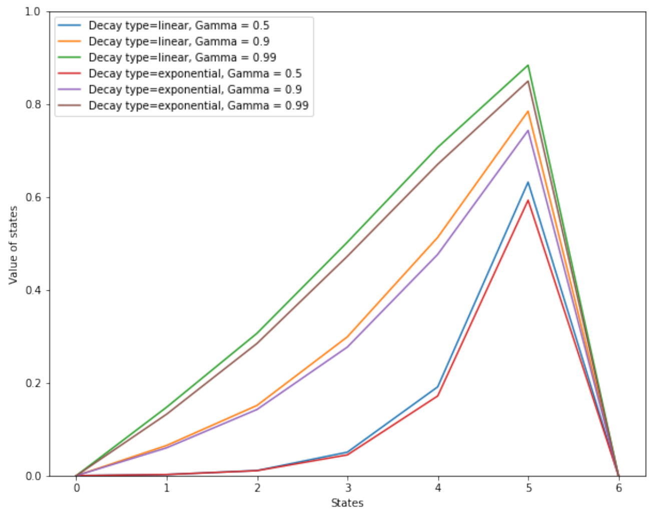


Figure 25: Test cases for TD algorithm

A similar trend is seen for the value approximations in the TD algorithm compared to the MC algorithms. The ascent towards the higher values is much smoother in the TD algorithm. The approximate values for smaller states (1,2) are higher and for bigger states(4,5) are lower in comparison to the MC algorithms. The maximum number of episodes used are 500 and the seed used is 30.

5. Since the dynamics of the environment is known to the designer, the true value of the states can be calculated. A value iteration process was used here to calculate the true values for the states (gamma=0.99). The values for the 5 non-terminal states at each time step (0-500) obtained by the FVMC algorithm is plotted along with the true values for the states.

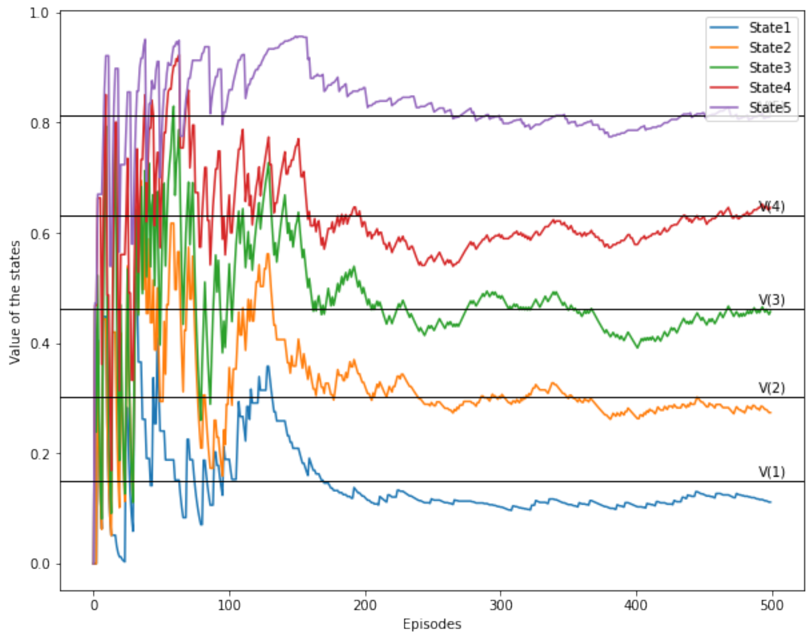


Figure 26: Values of states over time (FVMC)

The algorithm seems to have a large variance initially but eventually converges to the true values for the states. Since the algorithm directly uses the rewards received in the trajectories to update the values, and due to the stochasticity in the environment, a large variance is expected. α was decayed exponentially from 0.5 to 0.01 till 250 episodes and kept constant after. The seed used here was 50.

6. A similar result was seen using the EVMC algorithm. However, there seems to be more variance in the EVMC algorithm's approximation of the true values of the states. This is probably because all visits to a state are considered in the EVMC approach which would naturally increase the variance in the value estimates. The seed used here was 50.

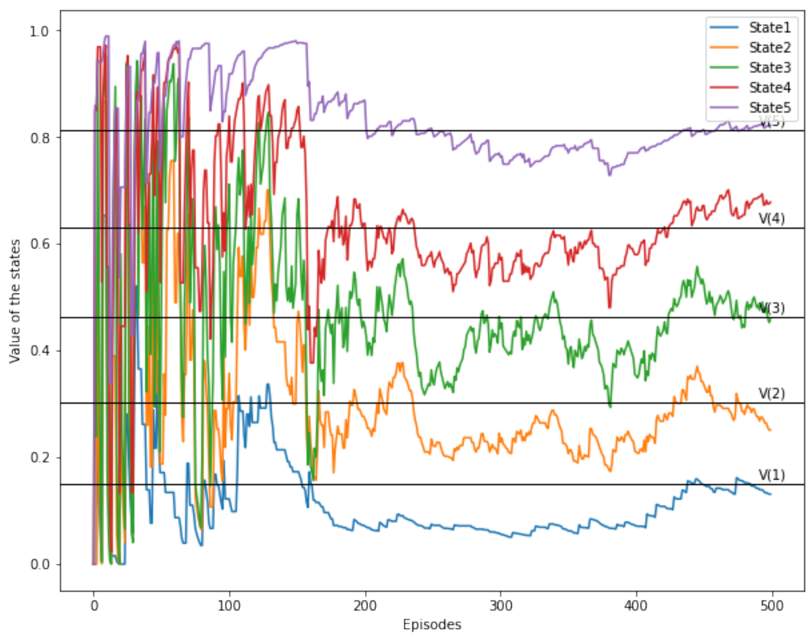


Figure 27: Values of states over time (EVMC)

7. The Temporal Difference algorithm seems to be doing slightly better than the MC algorithms in approximating the true values by arriving at the true values at a faster pace. There is also lower variance in the value approximations with the value approximations staying around the true values after convergence.

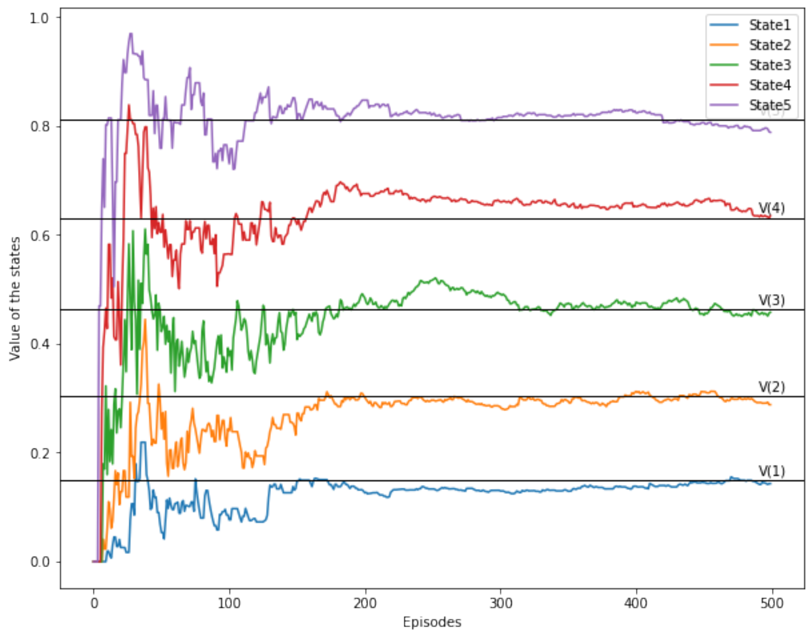


Figure 28: Values of states over time (TD)

8. The x-axis of the plots can be converted into the log scale as this allows a better estimation of the change in the value approximations over time.

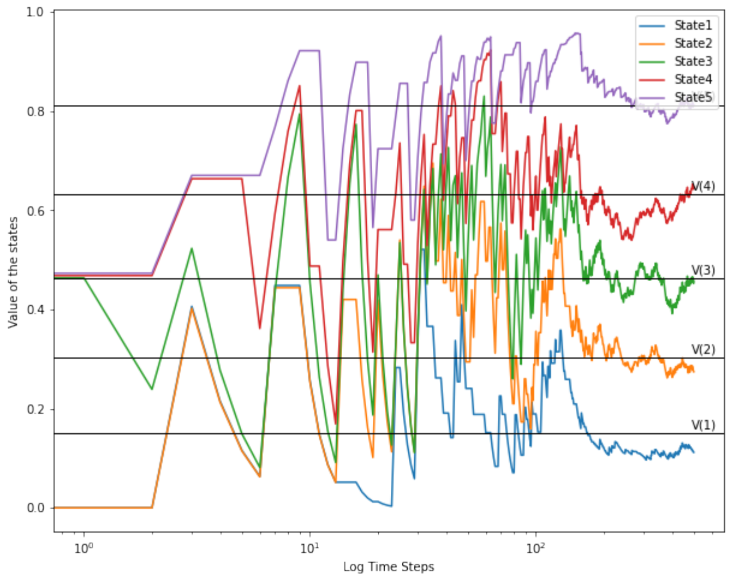


Figure 29: Values of states over log time (FVMC)

Looking at the values for the FVMC plot, there is a large amount of variance in the value approximations for the first 100 time steps and converges eventually. The algorithm is unbiased and is evidenced as the values hover both above and below the true values of the states before converging. The values only start to converge towards the later time steps with the help of the constant, small alpha values. The seed used here was 50.

9. Looking at the plot for the EVMC value approximations, there seems to be a much larger amount of variance in the values compared to the FVMC estimates, which is probably due to the reason stated above. The values for the states can also start off at much larger values compared to the FVMC algorithm. Since all the visits to the state are being considered, the discounting of the reward values will be lower, due to the lower decay of γ, and higher approximations can be made. However, the larger variance in target values G_e also result in larger variance in estimated values. The seed used here was 50.

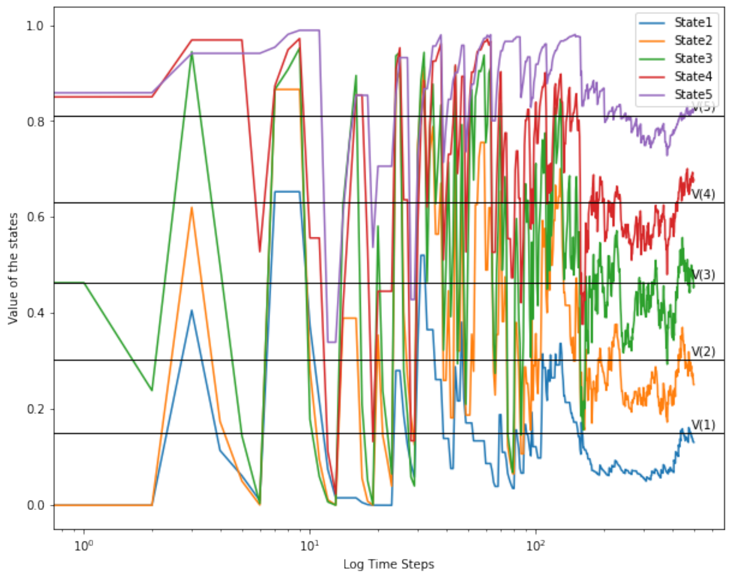


Figure 30: Values of states over log time (EVMC)

10. In the TD algorithm, the incremental update of the values are related to each other as the update of the value for a state is dependent on the value of the next state reached by the agent. The variance of the states is also much lesser and there is a smooth convergence towards the true values. There is an under-estimation of the values for states 1 and 2 because of the time it takes for the values to be 'back-propagated' towards the previous states. There is also an over-estimation of states 4 and 5 initially because of the high difference between target and estimated values for the states. However, the value estimates for the states eventually converge to the true values.

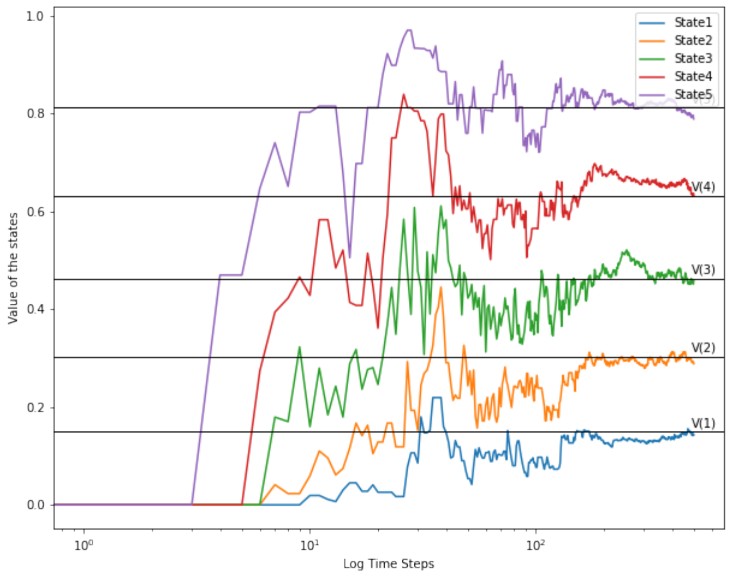


Figure 31: Values of states over log time (TD)

11. Eventually the three algorithms seem to converge to the true values. However the MC algorithms seem to have a higher variance for the value estimates, especially the EVMC algorithm. Therefore, they may not be appropriate when smaller episodes are run due to the wider possible range of estimates that can be received and the time steps taken for the algorithms to converge. The TD algorithm's value estimations however are biased with under-approximations of the lower states and over-approximations of the higher states for smaller episodes. Knowledge of the bias could help adjust the value approximations accordingly when the episodes run are not sufficient. For a large number of episodes, it seems that there would not be a big difference in performance of the algorithms in the current environment. However, environments with more complicated reward patterns and states could change the variance and bias of the algorithms accordingly.

12. The target for the Monte Carlo algorithm is the sum of rewards received in an episode G_e.

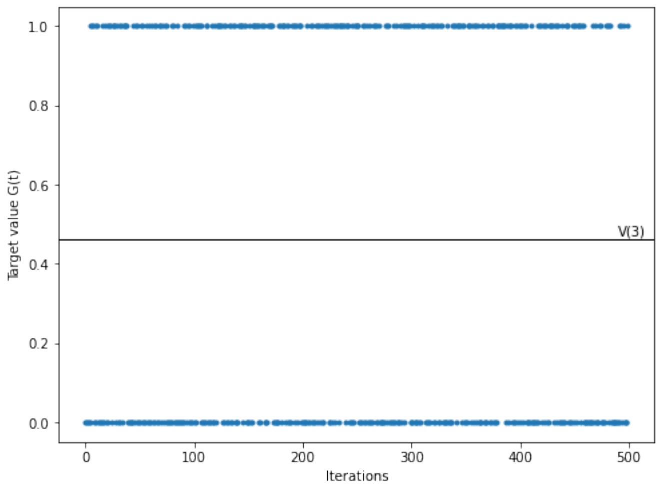


Figure 32: FVMC target estimate over time (state 3)

Since the rewards are received only when the episode is terminated and can only be either 0 or 1, if the rewards are not discounted ($\gamma = 1$) then the sum of rewards in an episode will be either 0 or 1. Therefore the target values for the different episodes fluctuate between 0 and 1. Since the environment is initialised at state 3, there is also an equal probability of reaching a positive reward(1) or no reward and there is an equal distribution of rewards above and below the true value.

13. The same logic stated above holds true for the EVMC algorithm as well and the target value takes the values of 0 and 1 with an equal probability.

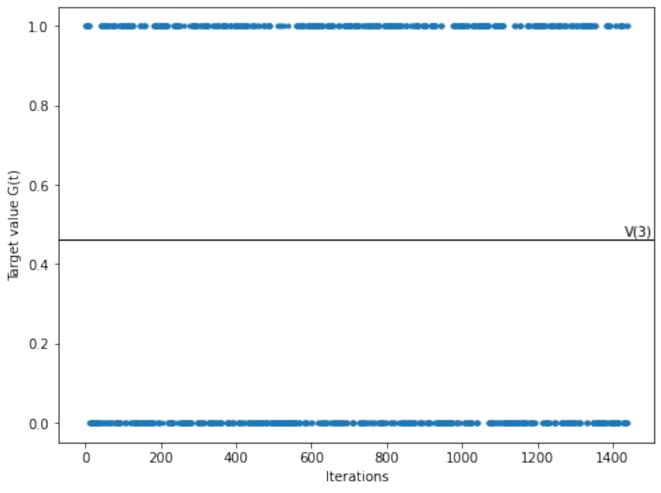


Figure 33: EVMC target estimate over time (state 3)

14. For the TD algorithm, the target is defined as the reward received in the next state plus the discounted value of the next state. Therefore the target value need not be only 0 and 1 and is distributed between the two values. Initially, the target values are far away from the true values for the respective states. Since the agent can reach state 2 (further away from reward) or state 4 (closer to reward) from state 3, the target values are distributed both above and below the true values. Eventually, the target values converges to the true values as seen in the plot.

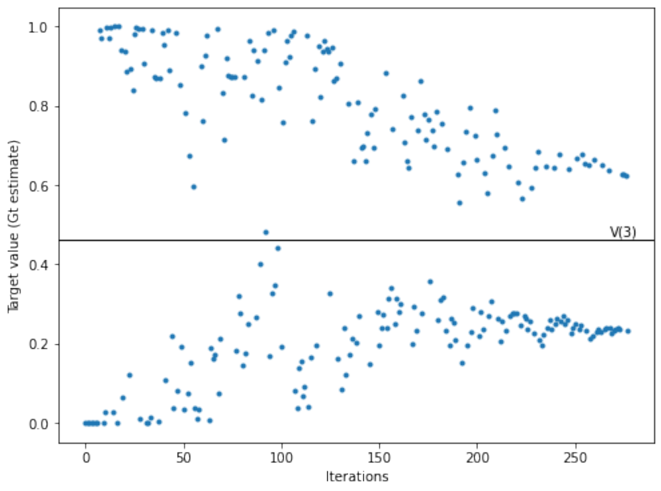


Figure 34: TD target estimate over time (state 3)

15. The TD target values are not only the rewards received but the value of the neighbouring states as well. In the current environment, reward is only directly received in state five and the rest of the states depend on the neighbouring states for their value approximations. Therefore, if the neighbouring states converge to the true values, so will the current state and if the neighbouring states diverge, so will the current state. The lower variance in the TD estimates can also be explained through these plots as the target values mostly lie significantly lower than 1 and higher than 0. Therefore, there is a smoother convergence towards the true values. The MC algorithms' targets however fluctuate between 0 and 1 (when$\gamma\bar{1}$). Therefore, a few deviant trajectories can create strong variance in the value estimates and can cause a larger amount of time to converge. However, as the number of samples increase, it is inevitable that the algorithms will converge to the true values.

    Based on the above observations, there seems to be a bias-variance trade off in choosing between the algorithms. The TD algorithm seems to have a higher bias and the MC algorithms seem to have a higher variance. Keeping in mind the requirements of the designer, the respective algorithm can be chosen. The MC algorithms may need a large number of samples through many iterations in order to converge. The speed of the TD algorithm and predictability of the bias puts it above the MC algorithms for the current environment.