CS698R: Deep Reinforcement Learning

Assignment #1

Name: Akshat Sharma
Roll NO.: 190090

Solution to Problem 1: Multi-armed Bandits

1. The bernoulli bandit environment is implemented in the Bernoulli_Bandit class. To check if the environment is working correctly we ran 1000 episodes with the parameter settings highlighted in part 1. To check if the reward function returns the correct value we assert that reward is 1 only when the agent takes the left action and lands in state 1 or takes the right action and lands in state 2. Any deviation from this behaviour would raise an assertion error.
   To check if the transitions are according to the parameters α and β we estimate the fraction of times the agent lands in state 1, given it takes the action left. Under the limit of infinite transitions this value will converge to α. The results are visible in the code.

2. For the gaussian bandit environment we create an environment with different values of σ to confirm that the variance of the q values is indeed σ. We also estimate the reward of transition for every action and verify that it has mean and variance close to $q(a)$ and σ. The results are visible in the code. Set mean indicates the true value of $q(a)$ and estimated mean indicates the average over the samples obtained by interacting with the environment.

3. (a) Note that the agent gets the first positive reward at episode = 4. Hence the table contains the value [0.25, 0]. For the 5th episode the agent receives 0 reward, hence the estimate for $q(0)$ falls to $\frac{1}{5}$. Note that prior to the first positive reward the agent greedily takes the left action as np.argmax returns the index of the leftmost maxima. Once the agent receives a positive reward for the left action (in an expected $1/\alpha$ steps) the agent will forever select the greedy left action and $q(1)$ will remain 0. This is reflected in the experiments.

   (b) For the pure exploration strategy the agent samples an action randomly. The results are self-explanatory.

   (c) For the ϵ-greedy strategy the agent decides to explore with probability ϵ. Note that with $\epsilon = 0$ the strategy is identical to the pure exploitation strategy and the valuation of one of the states is 0 (as highlighted in part(a)). For the other runs notice that the Q values are close to the theoretical values (i.e. $[\alpha, \beta]$).

   (d) For episode number t and decay rate λ, the epsilon value for the episode is $(1 - \lambda t)$ for linear decay and $(e^{-\lambda t})$ for exponential decay. Epsilon is capped below at 0.

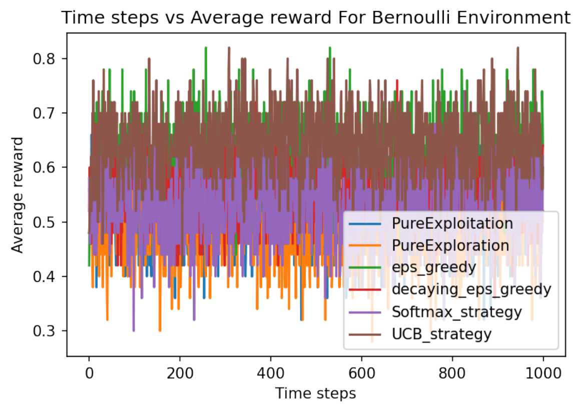
   (e) The softmax strategy samples actions from the distribution $softmax(\frac{q}{t})$. We normalize the logits(across mean only) before exponentiation to prevent numerical overflow.

   (f) The UCB strategy picks the action with the best $q + U$ value, where U is the upper confidence bound. Actions that have been unexplored for a long time have a higher U value and hence will be explored after a sufficient number of episodes. This is how UCB balances the exploration-exploitation trade-off.

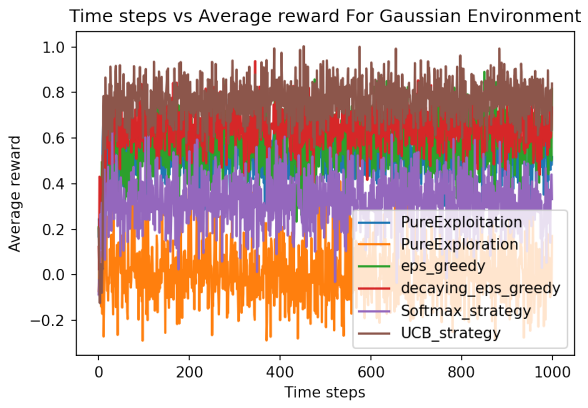
   All strategies also include the ret_reward_list and ret_action_list paramaters which, when true, make the function return the list of rewards(required for part 4-7) and actions taken(required for part 8-9) by the agent.

4. Note : We run this experiment with the default values of the hyperparameters. The default values and the plots showing the performance of strategies with different hyperparameters can be found in appendix 1.
   To average out the results for 50 instances of the environment we maintain a running average of the reward received by the agent using a strategy over all 50 instances of the environment. For clarity let the run k

on env_k generate a reward list r_k of length 1000, then the running mean maintains the mean of all the r_k lists seen so far. Plot 1 shows the results.



(a) Time steps vs Average reward For Bernoulli Environment



(b) Time steps vs Average reward For Gaussian Environment

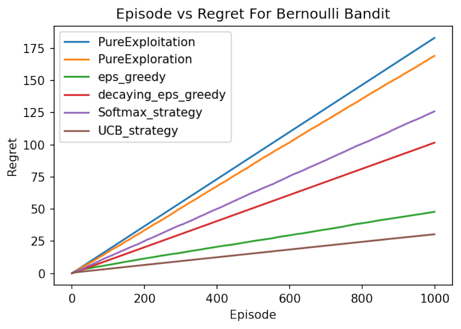
Figure 1: Time step vs average reward

Upon inspection we observe that the UCB strategy works the best on both the environments (for the default hyperparameter values).

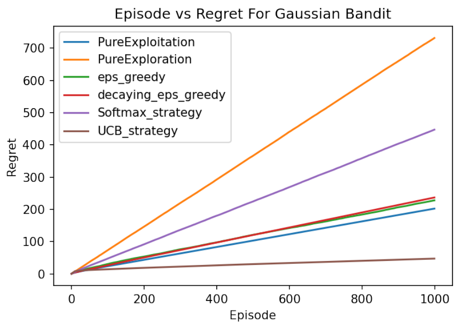
5. Note : We run this experiment with the default values of the hyperparameters. The default values and the plots showing the performance of strategies with different hyperparameters can be found in appendix 1.
   The gaussian bandit environment has a larger action space than the bernoulli environment, hence performance of the strategies differs more significantly in the gaussian environment (as is evident from the plot). Strategies that balance the exploration-exploitation trade-off methodically (e.g. UCB strategy) rank above other naive strategies like pure exploration and pure exploitation.

6. The agent regret is defined as $\sum_{e=1}^{E} \mathbb{E}[v_* - q_*(a)]$. The figure 2a plots the regret as a function of time steps for various strategies on the Bernoulli bandit environment. All strategies are run with default hyperparameter values.

7. The figure 2b plots the regret as a function of time steps for various strategies on the Gaussian bandit environment. All strategies are run with default hyperparameter values.



(a) Episode vs Regret for Bernoulli Environment

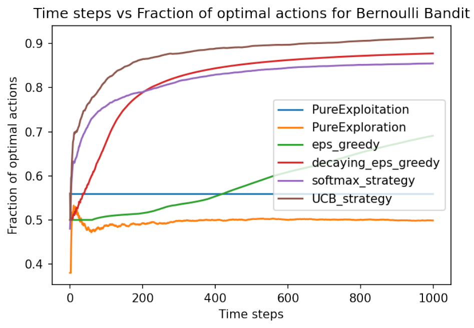


(b) Episode vs Regret For Gaussian Environment

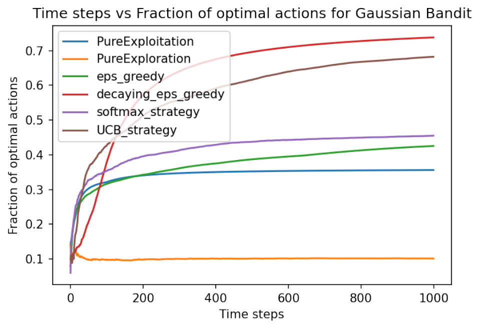
Figure 2: Episode vs Regret

8. The figure 3a plots the fraction of optimal actions taken by the agent for the bernoulli bandit environment. The strategies are run using the optimal value of the hyperparameters found in appendix 2.

9. The figure 3b plots the fraction of optimal actions taken by the agent for the gaussian bandit environment. The strategies are run using the optimal value of the hyperparameters found in appendix 2. Note that in both the environements, UCB and decaying ϵ strategies take the optimal action most frequently, followed by the softmax strategy. Naive strategies (pure exploitation/exploration) rank the lowest. Also the action space for the gaussian environment is larger, hence a strategy takes more time to learn the optimal action. This is evident from the scale of the y-axis in 3 (upto 0.9 in 3a and 0.7 in 3b).



(a) Time steps vs fraction of optimal actions for Bernoulli Environment



(b) Time steps vs fraction of optimal actions for Gaussian Environment

Figure 3: Time steps vs fraction of optimal actions

Solution to Problem 2: MC Estimates and TD Learning

1. To test the generate trajectory function we run it on the 'left' policy(always takes the left action) and maxSteps = 10. The results are visible in the code. Note that the trajectory either ends in state 0 or 6. Some trajectories are empty, denoting that the episode did not terminate in 10 steps. The experience tuples returned by the function consist of the current state, action taken, reward and the next state.

2. The decay alpha first uses the decay type, initial and final values to calculate the decay rate. Using the decay rate the function returns the list of step size parameter values.

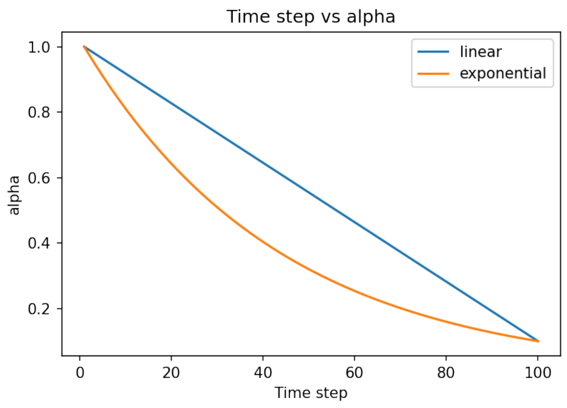


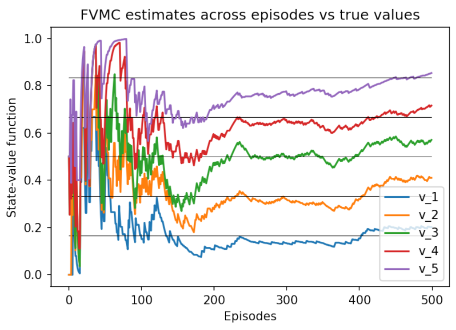
Figure 4: Time step vs alpha

3. The MonteCarloPrediction function includes two additional parameters, namely, print_trajectory and return_targets. The print_trajectory parameter is used in part 3 to verify that the function is working correctly. The return_targets is used to plot the returns as a function of episodes in part $12-13$. To test the function we run it on a single episode with the print_trajectory flag. The step_size parameter α decays as described in part 5 and has a value of 0.5 at the first episode. The γ value of the MDP is set to 1. The output trajectory is $s_3 \to s_4 \to s_5 \to s_6$. The final transition generates a reward of 1 hence the MC update for all the states in the trajectory looks like $v(s) = 0 + 0.5 \cdot (1 - 0) = 0.5$.

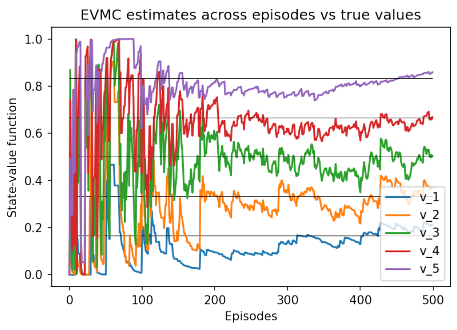
4. The TemporalDifferencePrediction function includes two additional parameters, namely, print_trajectory and return_targets. They behave as described above. To test the function we compare the state value estimates of returned by the function with the true state value estimates(see appendix 3 on how to find the true state value estimates). Note that the returned values are close to the true values, so the function is working as desired.

5. Please see figure 5a for details.

6. Please see figure 5b for details. Note that for both FVMC and EVMC, the state value estimates appear to converge to their true estimates(marked by the black lines). The plots start off with high variance but as the number of episodes increases α decays off and the targets from more recent trajectories have less effect on the estimate. Hence the variance of the plots decreases with the number of episodes. EVMC has higher variance than FVMC in the earlier phase of training. This is due to the fact that if a certain state appears multiple times in one trajectory then EVMC would push the value of the state to the target multiple times, hence increasing the variance in the estimate. This is equivalent to performing a single update using a higher effective α' (see appendix 4 for a justification of the statement).



(a) FVMC estimates across episodes vs true values



(b) EVMC estimates across episodes vs true values

Figure 5: MC estimates across episodes vs true values

7. TD estimates show significantly lower variance than MC estimates. This is because TD bootstraps the estimates of next state to obtain the target for the current state. In this sense TD is the true online version of the MC algorithm.

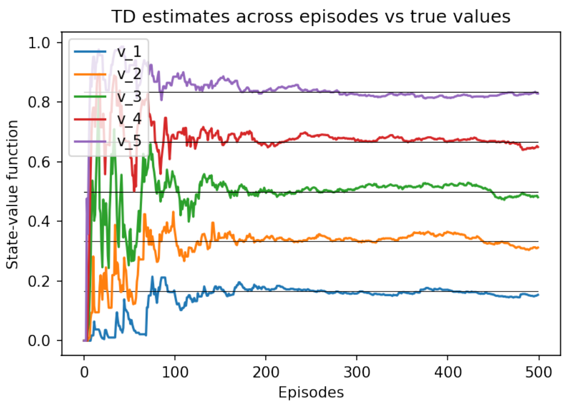
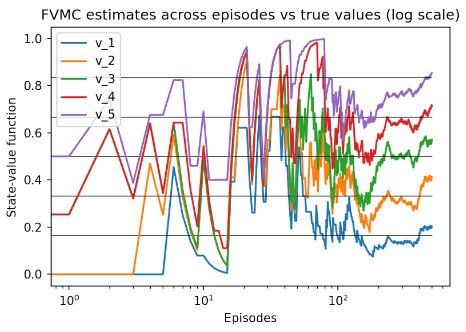


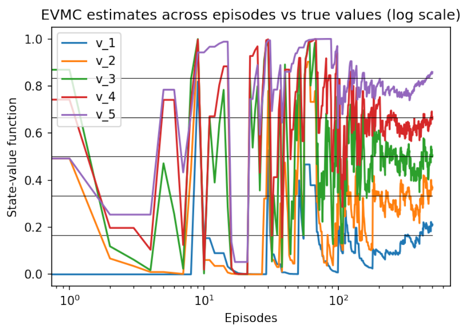
Figure 6: TD estimates across episodes vs true values

8. Please see figure 7a for details.

9. Please see figure 7b for details. The log scale plots highlight the difference in variance in the earlier phase of the training. The reason is as highlighted in point 6.



(a) FVMC estimates across episodes vs true values (log scale)



(b) EVMC estimates across episodes vs true values (log scale)

Figure 7: MC estimates across episodes vs true values (log scale)

10. Please see figure 8 for details. The TD estimates are steadier than MC estimates throughout the course of the training.

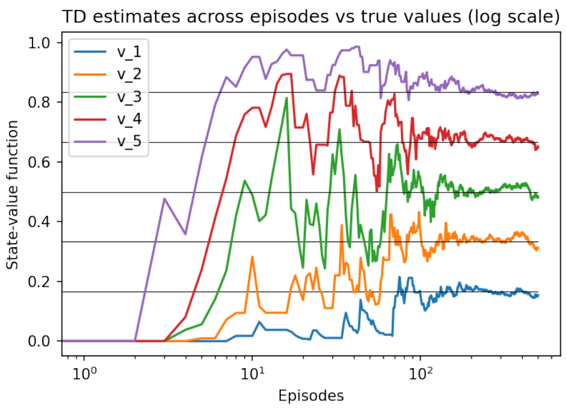
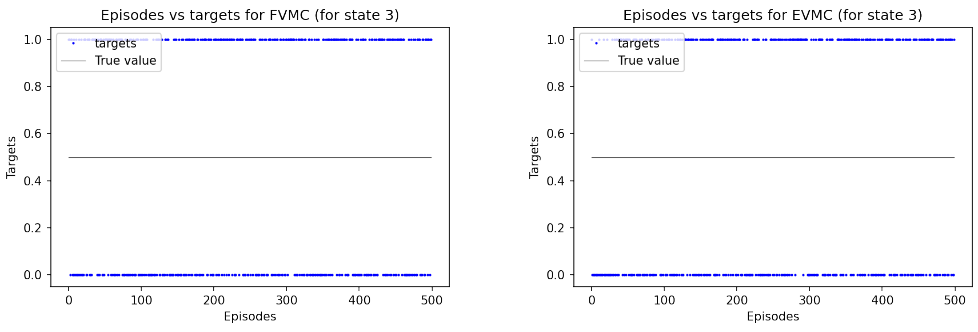


Figure 8: TD estimates across episodes vs true values (log scale)

11. The TD estimates have lower variance than MC estimates. The bias in TD methods does not seem to affect the values to which the estimates converge over the span of 500 episodes. Please see point 6 and 7 for elaborations.
12. Please see figure 9a for details.
13. Please see figure 9b for details. This is because the MC target is the total return from the state to a terminal state. As all non-terminal transitions return 0 reward, the reward of the final transition is the MC target. Hence, the targets for the MC updates are either 0 or 1, depending upon whether the trajectory terminates at state 0 or state 6.



(a) Episodes vs targets for FVMC (for state 3) (b) Episodes vs targets for EVMC (for state 3)

Figure 9: Episodes vs targets for MC (for state 3)

14. Please see figure 10 for details. Any action from state 3 will transition to state 2 or state 4. Hence the TD target is related to the current estimate of state 2 or state 4. This is evident from the plot. A majority of the points above the true value are from the transition state 3 to state 4 (as state 4 has a higher true state value than state 3) and the points below are from the transition state 3 to state 2.

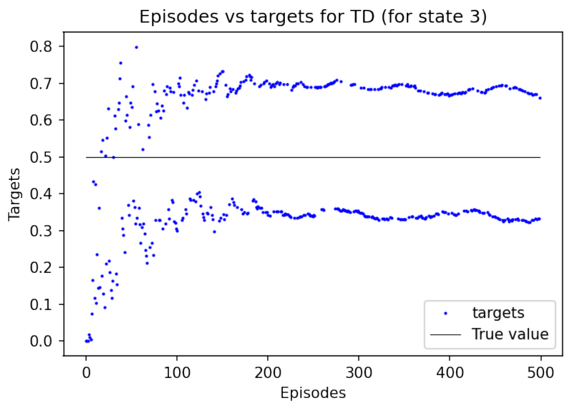


Figure 10: Episodes vs targets for TD (for state 3)

15. The MC targets are either 0 or 1 depending on the terminal state of the trajectory. The TD targets are bootstrapped from the neighbouring states. Please see points 13 and 14 for elaborations.

Appendix

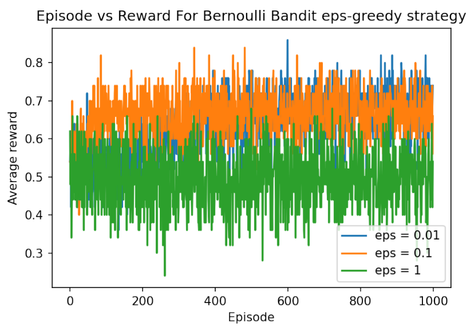
1 Default hyperparameter values for question 1

No.	Strategy	Default values
1	ϵ greedy	$\epsilon = 0.3$
2	Decaying ϵ greedy	decay rate = 1, decay type = exponential
3	Softmax strategy	temperature = 1.0
4	UCB strategy	c = 0.1

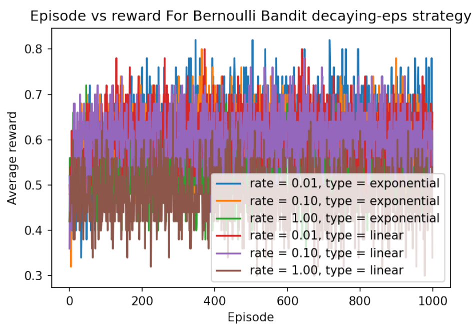
Table 1: Default parameters for various strategies

2 Hyperparameter tuning

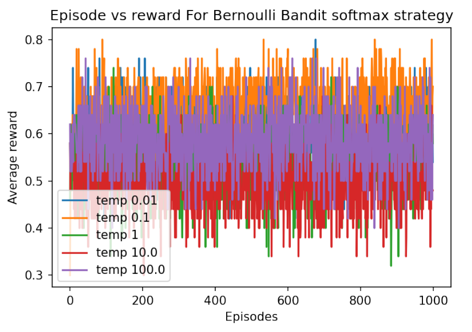
We tried a couple of parameter settings for different strategies. For the ϵ greedy strategy we tried out three values of ϵ i.e. $[0.01, 0.1, 1]$. For the decaying ϵ strategy we tested out two types of decay ([linear, exponential]) and three decay rates ($[0.01, 0.1, 1]$). For the softmax strategy we tested out five possible temperature values, ranging from 0.01 to 100. For the UCB strategy we ran the experiments for five c values ($[0.0, 0.25, 0.5, 0.75, 1.0]$). The results are shown below. For part 8-9 of Q1 we use the optimal hyperparameters.



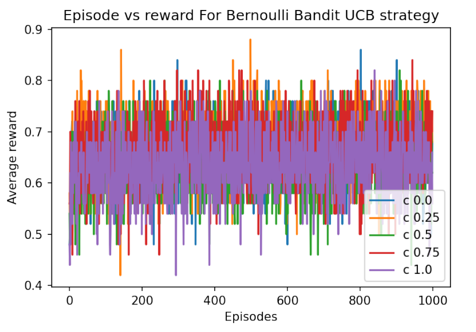
(a) ϵ greedy strategy



(b) decaying ϵ strategy

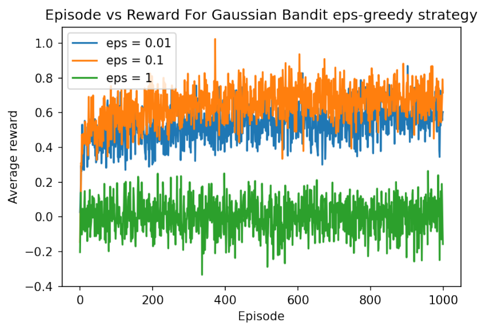


(c) Softmax strategy

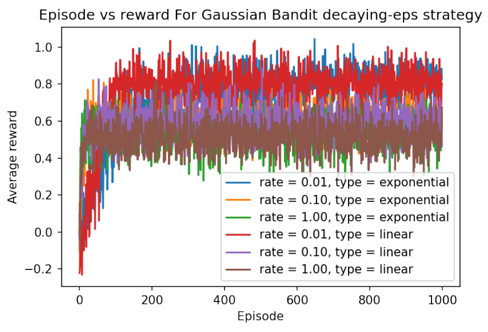


(d) UCB strategy

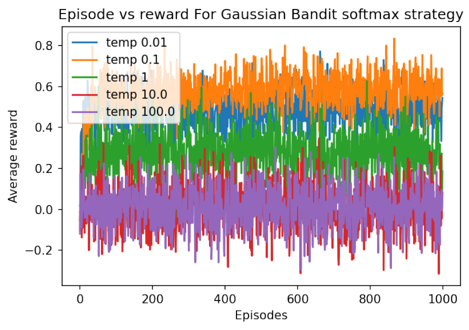
Figure 11: Hyperparameter tuning on Bernoulli environment



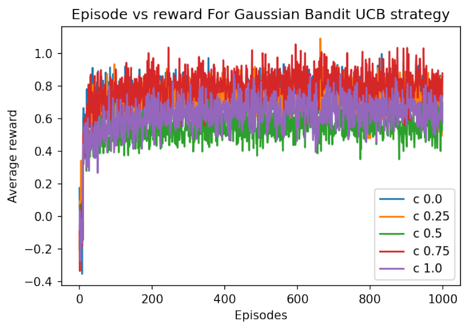
(a) ϵ greedy strategy



(b) decaying ϵ strategy



(c) Softmax strategy



(d) UCB strategy

Figure 12: Hyperparameter tuning on Gaussian environment

The optimal hyperparameters found are as below.

No.	Strategy	Optimal parameter values
1	ϵ greedy	$\epsilon = 0.01$
2	Decaying ϵ greedy	decay rate = 0.01, decay type = linear
3	Softmax strategy	temperature = 0.1
4	UCB strategy	c = 0.75

Table 2: Optimal parameters for strategies on the Bernoulli environment

No.	Strategy	Optimal parameter values
1	ϵ greedy	$\epsilon = 0.1$
2	Decaying ϵ greedy	decay rate = 0.01, decay type = exponential
3	Softmax strategy	temperature = 0.1
4	UCB strategy	c = 0.75

Table 3: Optimal parameters for strategies on the Gaussian environment

All experiments in question 1 are conducted via the starting seed = 131. All the involved random-number generators are first seeded and from here on the seed is passed down to any new random-number generators. See the seed method of the environments for more details.

3 Bellman equations for true state value estimates

The bellman equation is

$$v_\pi(s,a) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s')) \tag{1}$$

Writing this for all the states under the 'left' policy gives the following equations. We set γ to be 1.

$$\begin{aligned}
v(6) &= 0 \\
v(5) &= \frac{1+v(6)}{2} + \frac{v(4)}{2} \\
v(4) &= \frac{v(5)}{2} + \frac{v(3)}{2} \\
v(3) &= \frac{v(4)}{2} + \frac{v(2)}{2} \\
v(2) &= \frac{v(3)}{2} + \frac{v(1)}{2} \\
v(1) &= \frac{v(2)}{2} + \frac{v(0)}{2} \\
v(0) &= 0
\end{aligned}$$

Upon solving these equations we obtain the true valuations of the states to be

$$v(s) = \frac{s}{6} \tag{2}$$

for all non-terminal states s.

4 Difference between EVMC and FVMC updates

Consider a trajectory $s_0, s_1, \cdots s_T$ wherein some state s_* occurs n times ($n > 1$). Let the starting estimate of the state be $v_0 = 0$ and the subsequent estimates be $v_2, v_3, \cdots v_n$. Let the target for MC update be t ($t \in \{0, 1\}$). Hence the MC updates on state s_* will be

$$v_i = v_{i-1} + \alpha \cdot (t - v_{i-1}) \tag{3}$$

$$v_i = (1-\alpha)v_{i-1} + \alpha t \tag{4}$$

for $i \in \{2, 3, \cdots n\}$. This gives us the valuation of s_* at the end of the trajectory to be

$$v_n = (1-\alpha)^n v_0 + n\alpha t = n\alpha t \tag{5}$$

This is an update with an effective step_size parameter of $\alpha' = n\alpha$.