1. What is temporal difference learning?

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based on other learned estimates, without waiting for a final outcome (they bootstrap). The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of reinforcement learning. As usual, we will start with the policy evaluation or prediction problem, that of estimating the value function v_π for a given policy π.

For the control problem (finding an optimal policy), DP, TD, and Monte Carlo methods all use some variation of generalized policy iteration (GPI). The differences in the methods are primarily differences in their approaches to the prediction problem. Temporal difference learning in machine learning got its name from the way it uses changes, or differences, in predictions over successive time steps for the purpose of driving the learning process.

2. Temporal difference learning is said to be a combination of the Monte Carlo (MC) method and the Dynamic Programming (DP) method. Justify

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based on other learned estimates, without waiting for a final outcome (they bootstrap). The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of reinforcement learning. As usual, we will start with the policy evaluation or prediction problem, that of estimating the value function v_π for a given policy π.

For the control problem (finding an optimal policy), DP, TD, and Monte Carlo methods all use some variation of generalized policy iteration (GPI). The differences in the methods are primarily differences in their approaches to the prediction problem. Temporal difference learning in machine learning got its name from the way it uses changes, or differences, in predictions over successive time steps for the purpose of driving the learning process.

3. State the advantages of Temporal Difference Learning.

- TD methods have an advantage over DP methods in that they do not require a model of the environment, of its reward and next-state probability distributions.
- With Monte Carlo methods one must wait until the end of an episode, because only then is the return known, whereas with TD methods one need to wait only one time step
- It is convenient to learn one guess from the next, without waiting for an actual outcome, but we can still guarantee convergence to the correct answer is the advantage of TD prediction over Dp and MC methods

4. What are the advantages of Temporal Difference Learning over Monte Carlo methods? Explain with suitable examples.

- TD learning is an online method, meaning it updates the value function after each time step, whereas MC methods require the episode to finish before updating the value function. This makes TD learning more efficient in terms of time and memory requirements.
- TD learning can learn from incomplete sequences of experience, whereas MC methods require a full episode to occur before any updates can be made. This makes TD learning more flexible and better suited for continuous reinforcement learning problems.
- TD learning can incorporate bootstrapping, which means using an estimate of the value function to update the current estimate. This enables TD learning to learn from smaller amounts of data and to generalize better to new situations.

5. Consider the real life example of examination patterns like annual examination and weekly examination to get the final score at the end of year. Identify the apprpriate RL approach for both patterns and justify the mapping.

One could use MC learning for the annual examination pattern by treating each exam as a separate episode and averaging the returns obtained from each episode. Similarly, one could use TD learning for the weekly examination pattern by updating the value function after each exam, even though the information is only available at the end of the week.

6. Give the update rule for TD(1) and TD(0) algorithm and explain each term in it.

TD(1) Update:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

Figure 2: Sum of Discounted Rewards

Here, Gt is the discounted sum of all the rewards seen in our episode.
As we are traveling through the environment, we keep track of all the rewards and sum them together with a discount (**γ**). The immediate reward (R) at a given point (time, t+1) plus the discount(**γ**) of a future reward (Rt+2) and so on.
From the above equation, we can see that we discount (**γ**) more heavily on the future with **γ**^T-1.
We'll use the sum of discounted rewards Gt, for our episode and we'll subtract that from the prior estimate. This is called the TD Error. Our updated estimate minus the previous estimate. Then we multiply by alpha to adjust how much of that error we want to update by.
Lastly, we make the update by simply adding our previous estimate V(St) to the adjusted TD Error as:

$$V(S_t) = V(S_t) + \alpha(G_t - V(S_t))$$

Figure 3: TD(1) Update Value toward Action Return

So that's 1 episode. We did 1 random walk and accumulated rewards. We then took those rewards at each timestep and compared them to our original estimate of values (all zeros)
We weigh the difference and adjust our prior estimate. Then start over again. This is TD(1) update.

TD(0) Update:
Instead of using the accumulated sum of rewards (Gt), we will only look at the immediate reward (Rt+1), plus the discount of the estimated value of only 1 step ahead (V(St+1)) as:

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

7. Define TD error.

- TD error arises in various forms throughout reinforcement learning and **δt = rt+1 + γV(st+1) − V(st)** value is commonly called the TD Error.
- Here the TD error is the difference between the current estimate for Vt, the discounted value estimate of Vt+1, and the actual reward gained from transitioning between st and st+1.
- The TD error at each time is the error in the calculation made at that time. Because the TD error at step t relies on the next state and next reward, it is not available until step t + 1.
- When we update the value function with the TD error, it is called a backup. The TD error is related to the Bellman equation.

8. Write pseudocode for TD(0) algorithm.

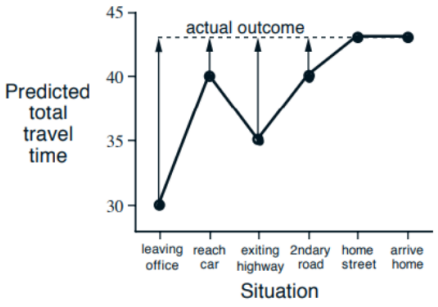
Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A; observe reward, R, and next state, S'
        $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
        $S \leftarrow S'$
    until S is terminal

9. For given data of Driving Home example, explain the corrections (changes in predictions) suggested by both the MC and TD approach with the help of graphs.

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

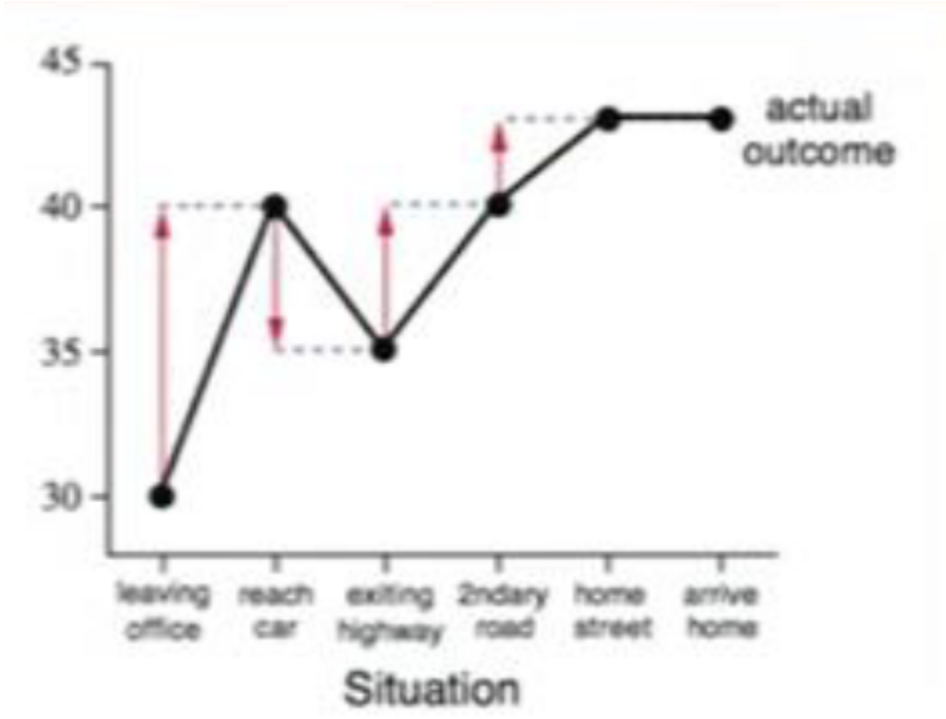
Example: Driving Home

State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

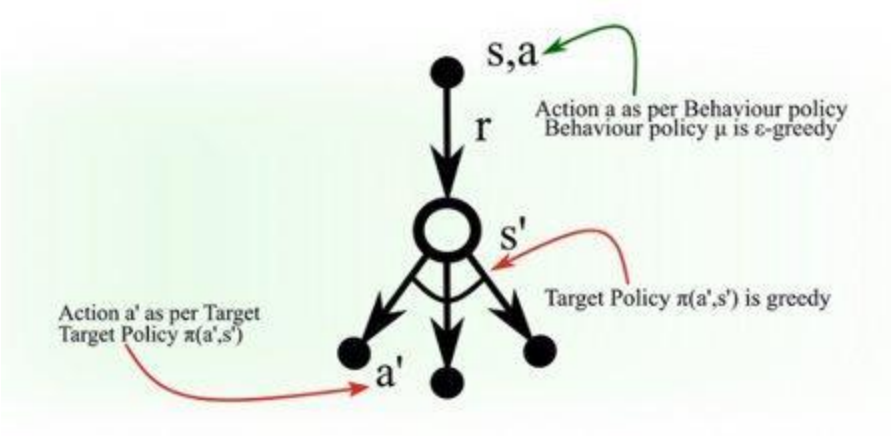


- A simple way to view the operation of Monte Carlo methods is to plot the predicted total time (the last column) over the sequence
- . The arrows show the changes in predictions recommended by the constant-α MC method, for α = 1.
- These are exactly the errors between the estimated value (predicted time to go) in each state and the actual return (actual time to go).
- For example, when you exited the highway you thought it would take only 15 minutes more to get home, but in fact it took 23 minutes.

In TD, the arrows are the errors between the estimated value (predicted time to go) in each state and the next estimated value.



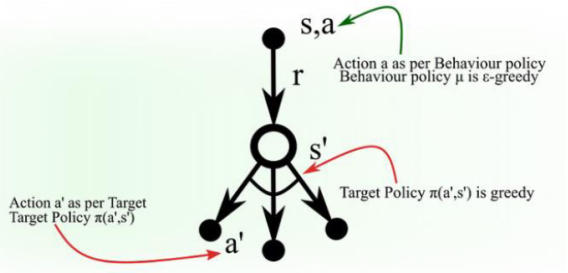
10. Identify the TD algorithm for given backup diagram and also state its update rule.

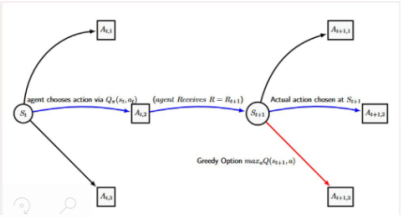


Q-Learning Algorithm

- Q-learning is an **off-policy** algorithm. In Off-policy learning, we evaluate target policy (π) while following another policy called **behavior policy** (μ) (this is like a robot following a video or agent learning based on experience gained by **another agent**).
- In the Q-learning, target policy is a **greedy policy** and behavior policy is the **ε-greedy policy** (this ensures exploration).

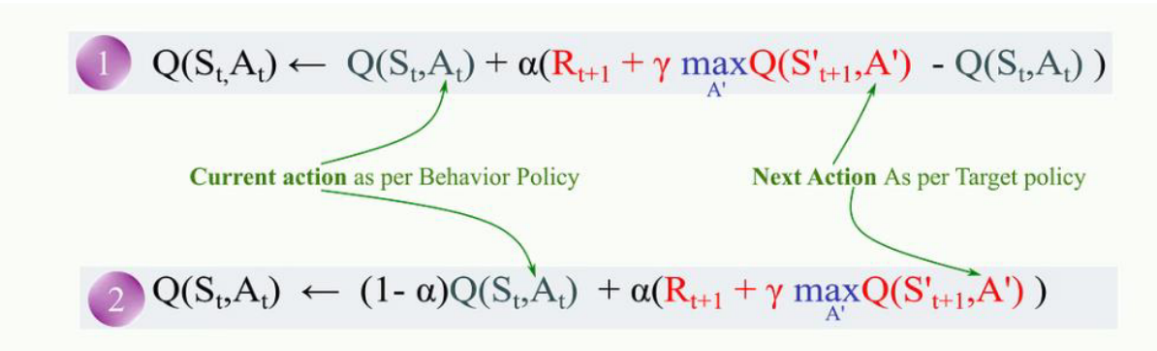
A Q-value indicates the *quality* of a particular action *a* in a given state *s*: *Q(s, a)*





Q- Learning Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$



11. Explain the difference between on-policy and off-policy TD methods.

12. Is SARSA an on-policy or off-policy method? Explain.

SARSA (State-Action-Reward-State-Action) is an on-policy reinforcement learning method.

In SARSA, the agent learns by updating its Q-value estimates based on the current policy, meaning that it uses the same policy for both action selection and value updates. Specifically, SARSA learns by estimating the value of taking an action in a given state and following the current policy thereafter. It updates the Q-value of the current state-action pair based on the current reward, the Q-value of the current state-action pair, and the estimated Q-value of that next state-action pair.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

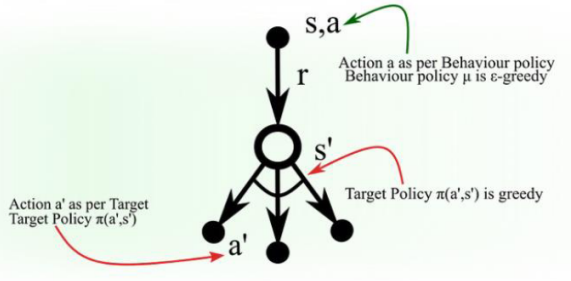
Because SARSA uses the same policy for both action selection and value updates, it is considered an on-policy method.

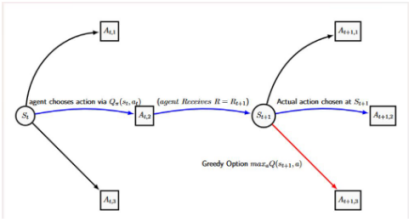
13. Is Q learning an on-policy or off-policy method? Explain.

Q-Learning Algorithm

- Q-learning is an **off-policy** algorithm. In Off-policy learning, we evaluate target policy (π) while following another policy called **behavior policy** (μ) (this is like a robot following a video or agent learning based on experience gained by **another agent**).
- In the Q-learning, target policy is a **greedy policy** and behavior policy is the **ε-greedy policy** (this ensures exploration).

A Q-value indicates the quality of a particular action *a* in a given state *s*: *Q(s, a)*





Q- Learning Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

1 $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha(R_{t+1} + \gamma \max_{A'} Q(S'_{t+1},A') - Q(S_t,A_t))$

Current action as per Behavior Policy

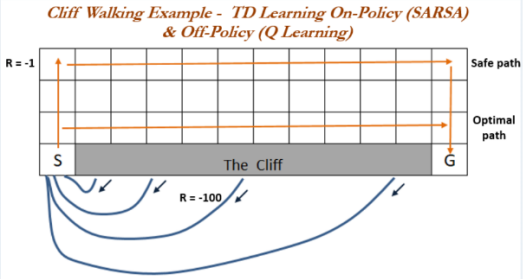
Next Action As per Target policy

2 $Q(S_t,A_t) \leftarrow (1-\alpha)Q(S_t,A_t) + \alpha(R_{t+1} + \gamma \max_{A'} Q(S'_{t+1},A'))$

14. Explain the difference between SARSA and Q learning with the help of cliff walking problem

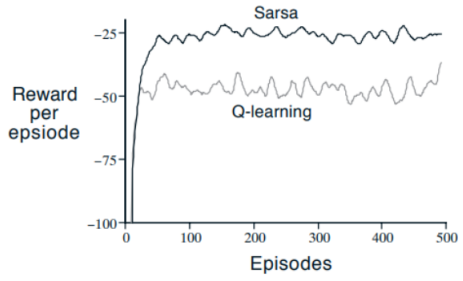
Q-Learning Vs SARSA

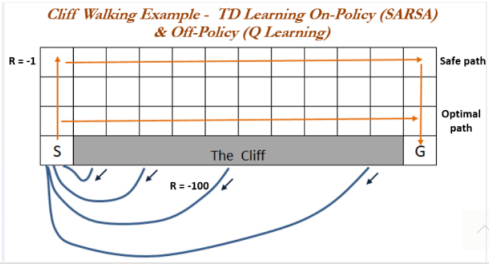
Cliff Walking Problem:



- This is a standard undiscounted, episodic task, with start and goal states, and the usual actions causing movement up, down, right, and left.
- Reward is −1 on all transitions except those into the region marked “The Cliff.”
- Stepping into this region incurs a reward of −100 and sends the agent instantly back to the start.

Q-Learning Vs SARSA





- The graph shows the performance of the Sarsa and Q-learning methods with ε-greedy action selection, ε = 0.1.
- After an initial transient, Q-learning learns values for the optimal policy, that which travels right along the edge of the cliff. Unfortunately, this results in its occasionally falling off the cliff because of the ε-greedy action selection.
- Sarsa, on the other hand, takes the action selection into account and learns the longer but safer path through the upper part of the grid.
- Although Q learning actually learns the values of the optimal policy, its on-line performance is worse than that of Sarsa, which learns the roundabout policy.
- Of course, if ε were gradually reduced, then both methods would asymptotically converge to the optimal policy

15. Write the pseudocode for SARSA.

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

16. Write the pseudocode for Q learning.

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

17. Consider the following simulation: A person walking near the cliff with the goal of reaching from one point to another. Would SARSA be the better option or Q learning? Justify your answer.

In the given simulation of a person walking near a cliff with the goal of reaching from one point to another, SARSA would be a better option compared to Q-learning. The primary reason is that SARSA takes into account the current state, the action taken, the resulting reward, and the next stage in its updates, while Q-learning only considers the maximum Q-value of the next state.

Here's the justification for using SARSA in this scenario:

Safety Concerns: Walking near a cliff introduces safety concerns. SARSA, being an on-policy algorithm, considers the action selection policy during the learning process. It can incorporate a policy that emphasizes safe actions and avoids risky actions near the cliff. This property is

crucial in this scenario to prevent the person from taking unsafe actions that might lead to falling off the cliff.

Exploration: Because SARSA is based on policy, it has a built-in way to explore. Following the current action selection strategy, it strikes a good balance between exploration and exploitation. This can be helpful near the cliff because it makes the person want to try other routes that might be safer or go faster. Q-learning, on the other hand, is an off-policy algorithm that tends to focus more on taking advantage of the best actions right now. This can lead to less-than-optimal decisions near the cliff.

Feedback on Risky Actions: SARSA tells you what rewards you get from the world based on the actions you choose. This instant feedback lets the person learn from things like taking a risk near the cliff and getting a bad result or falling off the cliff. SARSA can change its strategy so that this kind of thing doesn't happen again. Q-learning, which changes based on the highest Q-value of the next state, doesn't tell you right away what will happen if you do something risky.

Learning from Online Interaction: SARSA is an online learning algorithm, which means that it learns and changes its policy as it interacts with the world. This kind of learning in real-time can be helpful if the person is walking near a cliff because it lets them keep adapting to changing conditions or possible dangers.

18. Consider the following Q[S,A] table:

	Action 1	Action 2	Action 3
State 1	1.5	4	1
State 2	2.5	3	1.5
State 3	2	4	3

Assume that α=0.1, and γ=0.5. Update the Q table with the following (S, A, R, S', A') experiences using SARSA.

A. ⟨1, 1, 5, 2, 1⟩ // solution -> $Q(1,1) = Q[s,a] + \alpha(r + \gamma Q[s',a'] - Q[s,a]) = 1.5 + 0.1(5 + 0.5(2.5) - 1.5) = 1.975$

B. ⟨2, 1, 3, 3, 2⟩

C. ⟨3, 2, 6, 1, 3⟩

D. ⟨1, 3, 4, 2, 3⟩

At each step, state which value of the table gets updated and draw the final updated Q[S, A] table.

19. Consider the following Q[S,A] table:

	Action 1	Action 2	Action 3
State 1	1.5	4	1
State 2	2.5	3	1.5
State 3	2	4	3

Assume that α=0.1, and γ=0.5. Update the Q table with the following (S, A, R, S') experiences using Q learning.

A. ⟨1, 1, 5, 2⟩ // solution -> Q(1,1) = Q[s, a] + α(r + γmaxa' Q[s', a'] - Q[s, a]) = 1.5 + 0.1(5 + 0.5(3) - 1.5) = 2

B. ⟨2, 1, 3, 3⟩

C. ⟨3, 2, 6, 1⟩

D. ⟨1, 3, 4, 2⟩

At each step, state which value of the table gets updated and draw the final updated Q[S, A] table.

20. Explain Eligibility traces in Reinforcement Learning.

- Eligibility traces : it trace a shorter memory of all the states visited in near past and update the values of these already seen states only when a positive or negative event occurs

- Eligibility traces are one of the basic mechanisms of reinforcement learning.
- Almost any temporal-difference (TD) method, such as Q learning or Sarsa, can be combined with eligibility traces to obtain a more general method that may learn more efficiently.
- Mathematically a vector E called eligibility traces. It is a function of state E(s) or state action E(s,a)

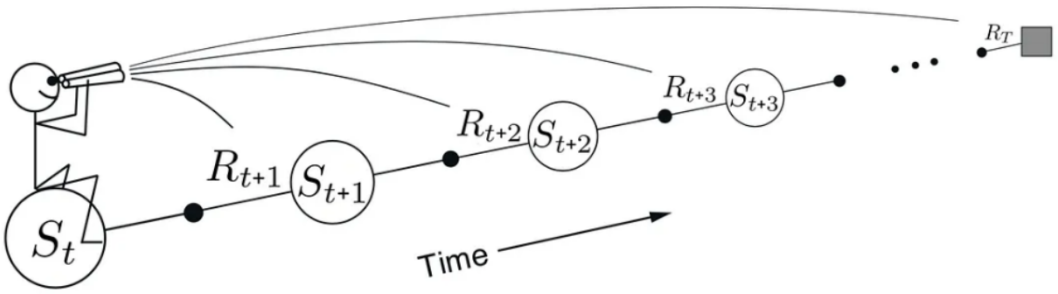
21. Explain the TD(λ) algorithm. What is the difference between TD(0), TD(1), and TD(λ)?

22. Explain the forward view of Eligibility traces

Forward View

- When TD methods are augmented with eligibility traces, they produce a family of methods spanning a spectrum that has Monte Carlo methods at one end and one-step TD methods at the other.
- In between are intermediate methods that are often better than either extreme method.
- In this sense eligibility traces unify TD and Monte Carlo methods in a valuable and revealing way.
- The forward view is most useful for understanding what is computed by methods using eligibility traces

Forward View



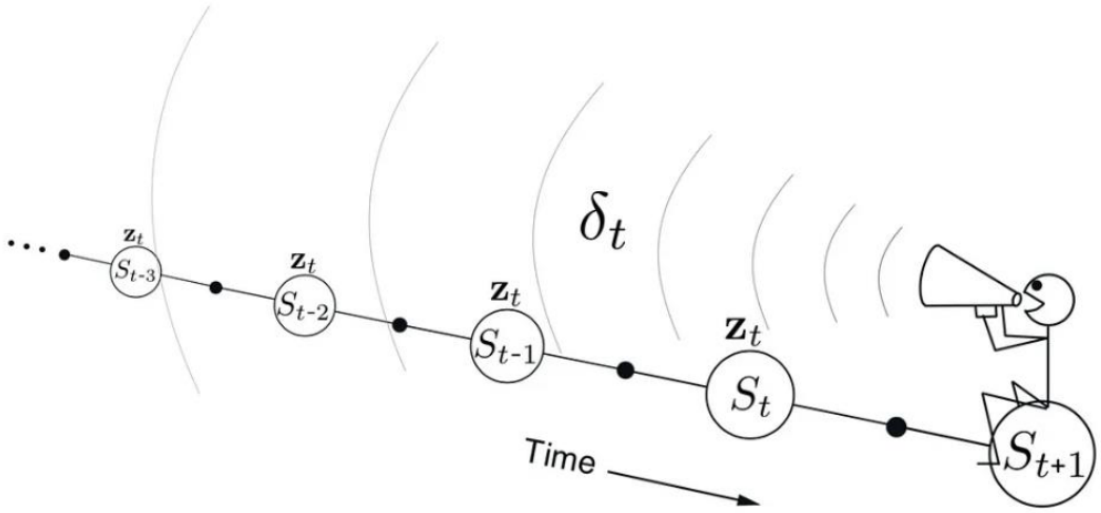
In Forward View we look ahead n steps for future rewards

In Temporal Difference and Monte Carlo methods we update a state based on future rewards. This is done either by looking directly one step ahead (TD(0)) or by waiting the episode to finish(MC).

23. Explain the backward view of Eligibility traces

Backward View

- When a TD error occurs, only the eligible states or actions are assigned credit or blame for the error.
- Thus, eligibility traces help bridge the gap between events and training information.
- Like TD methods themselves, eligibility traces are a basic mechanism for temporal credit assignment.
- The backward view is more appropriate for developing intuition about the algorithms themselves.



Backward View propagates the error δ to previous states

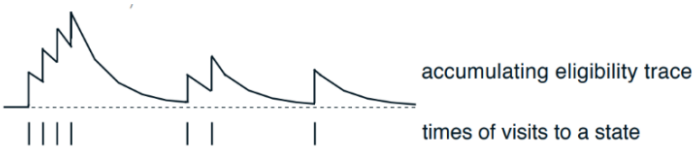
24. What are Accumulating trace, Dutch trace, and Replacing trace? Explain in brief. Compare three of them.

Accumulating Trace

$$E_t(S_t) = \gamma\lambda E_{t-1}(S_t) + 1.$$

- This kind of eligibility trace is called an **accumulating trace** because it accumulates each time the state is visited, then fades away gradually when the state is not visited, as illustrated as illustrated below.

$$E_t(s) = \begin{cases} \gamma\lambda E_{t-1}(s) & \text{if } s \neq S_t \\ \gamma\lambda E_{t-1}(s) + 1 & \text{if } s = S_t \end{cases}$$



Dutch Trace

- In the special case of λ = 1, TD(λ) with replacing traces is closely related to first-visit Monte Carlo methods.
- The second trace variation, called the **dutch trace**, is sort of intermediate between accumulating and replacing traces, depending on the step-size parameter α.
- Dutch traces are defined by

$$E_t(S_t) = (1 - \alpha)\gamma\lambda E_{t-1}(S_t) + 1$$

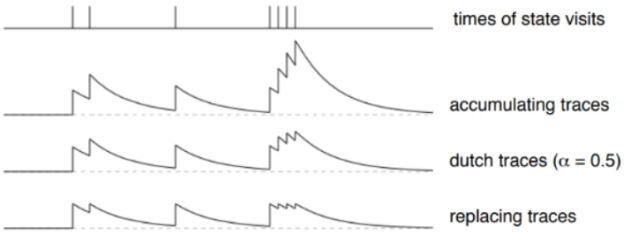
- Note that as α approaches zero, the dutch trace becomes the accumulating trace, and, if α = 1, the dutch trace becomes the replacing trace.

Replacing Trace

$$E_t(s) = \gamma\lambda E_{t-1}(s), \qquad \forall s \in \mathcal{S}, s \neq S_t$$

- Two alternative variations of eligibility traces have been proposed to address these limitations of the accumulating trace.
- On each step, all three trace types decay the traces of the non-visited states in the same way, that is as given above, but they differ in how the visited state is incremented.
- The first trace variation is the **replacing trace.**
- Suppose a state is visited and then revisited then with accumulating traces the revisit causes a further increment in the trace driving it greater than 1, whereas. with replacing traces, the trace is simply reset to 1: $E_t(S_t) = 1$

Accumulating, Dutch and Replacing Traces



- The figure above contrasts the three different kinds of traces.
- Accumulating traces add up each time a state is visited, whereas replacing traces are reset to one, and dutch traces do something in between, depending on α (here we show them for α = 0.5).
- In all cases the traces decay at a rate of γλ per step

25. Write the pseudocode for SARSA(λ)

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
      $e(s,a) \leftarrow \gamma \lambda e(s,a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

26. Define Eligibility traces in Reinforcement Learning. Explain Accumulating trace, Dutch trace, and Replacing trace with the help of graphs.

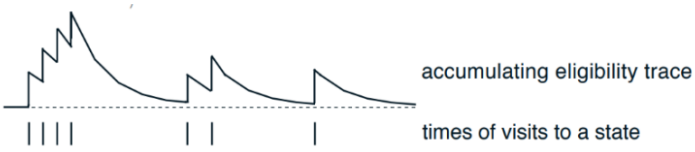
- Eligibility traces : it trace a shorter memory of all the states visited in near past and update the values of these already seen states only when a positive or negative event occurs

Accumulating Trace

$$E_t(S_t) = \gamma\lambda E_{t-1}(S_t) + 1.$$

- This kind of eligibility trace is called an **accumulating trace** because it accumulates each time the state is visited, then fades away gradually when the state is not visited, as illustrated as illustrated below.

$$E_t(s) = \begin{cases} \gamma\lambda E_{t-1}(s) & \text{if } s \neq S_t \\ \gamma\lambda E_{t-1}(s) + 1 & \text{if } s = S_t \end{cases}$$



Dutch Trace

- In the special case of λ = 1, TD(λ) with replacing traces is closely related to first-visit Monte Carlo methods.
- The second trace variation, called the **dutch trace**, is sort of intermediate between accumulating and replacing traces, depending on the step-size parameter α.
- Dutch traces are defined by

$$E_t(S_t) = (1 - \alpha)\gamma\lambda E_{t-1}(S_t) + 1$$

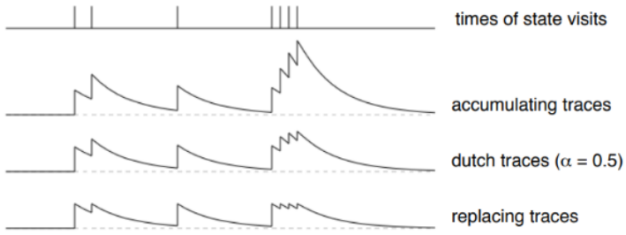
- Note that as α approaches zero, the dutch trace becomes the accumulating trace, and, if α = 1, the dutch trace becomes the replacing trace.

Replacing Trace

$$E_t(s) = \gamma\lambda E_{t-1}(s), \qquad \forall s \in \mathcal{S}, s \neq S_t$$

- Two alternative variations of eligibility traces have been proposed to address these limitations of the accumulating trace.
- On each step, all three trace types decay the traces of the non-visited states in the same way, that is as given above, but they differ in how the visited state is incremented.
- The first trace variation is the **replacing trace.**
- Suppose a state is visited and then revisited then with accumulating traces the revisit causes a further increment in the trace driving it greater than 1, whereas. with replacing traces, the trace is simply reset to 1: $E_t(S_t) = 1$

Accumulating, Dutch and Replacing Traces



- The figure above contrasts the three different kinds of traces.
- Accumulating traces add up each time a state is visited, whereas replacing traces are reset to one, and dutch traces do something in between, depending on α (here we show them for α = 0.5).
- In all cases the traces decay at a rate of γλ per step