

Module V: TEMPORAL DIFFERENCE LEARNING

By: Dimple Bohra

Contents

- Temporal Difference Learning
- What is Temporal Difference learning
- Advantages of Temporal Difference methods over Monte Carlo and Dynamic Programming methods
- TD(0)
- On-policy vs off-policy
- SARSA
- Q learning

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 in part 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 .

Temporal Difference Learning

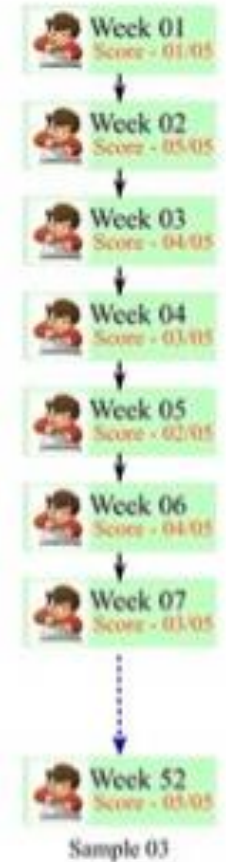
- For the control problem (finding an optimal policy), DP, TD, and Monte Carlo methods all use some variation of generalized policy iteration (GPI).
- The differences in the methods are primarily differences in their approaches to the prediction problem.
- Temporal difference learning in machine learning 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.

How TD works ?

- The trick is that rather than attempting to calculate the total future reward, temporal difference learning just attempts to predict the combination of immediate reward and its own reward prediction at the next moment in time.
- Now when the next moment comes and brings fresh information with it, the new prediction is compared with the expected prediction.
- If these two predictions are different from each other, the Temporal Difference Learning algorithm will calculate how different the predictions are from each other and make use of this temporal difference to adjust the old prediction toward the new prediction.

Example

- Consider a real-life analogy; if Monte Carlo learning is like an annual examination where student completes its episode at the end of the year. Similarly, we have TD learning, which can be thought like a weekly or monthly examination (student can adjust their performance based on this score (reward received) after every small interval and the final score is the accumulation of all weekly tests (total rewards)).



Temporal Difference ; TD(0)

TD algorithms and Notations

- There are three TD algorithms: TD(0), TD(1) and TD(λ). The notations used in these algorithms are:
- **Gamma (γ)**: the discount rate. A value between 0 and 1. The higher the value the less you are discounting.
- **Alpha (α)**: the learning rate. How much of the error should we accept and therefore adjust our estimates towards. A value between 0 and 1. A higher value adjusts aggressively, accepting more of the error while a smaller one adjusts conservatively but may make more conservative moves towards the actual values.
- **Delta (δ)**: a change or difference in value.

TD)(1)

- TD(1) makes an update to our values in the same manner as Monte Carlo, at the end of an episode.
- Consider random walk problem, going left or right randomly, until landing in 'A' or 'G'. Once the episode ends then the update is made to the prior states.
- TD(1) and MC only work in episodic environments meaning they need a 'finish line' to make an update.

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$

$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$



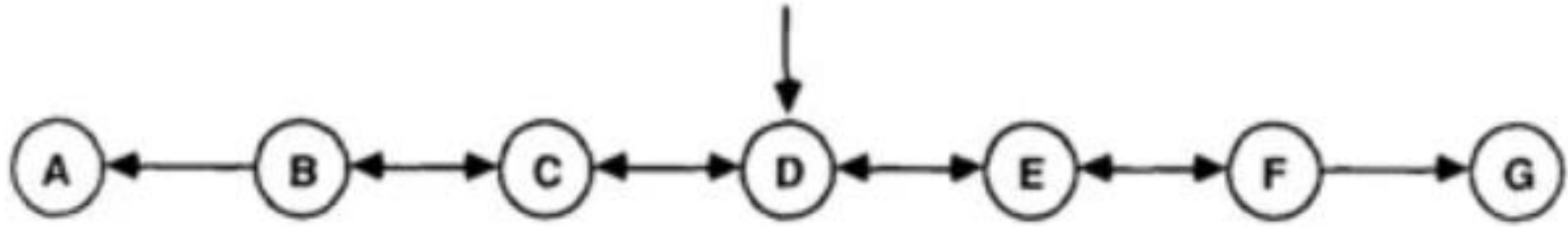


Figure 1: Random Walk Example [1]

Example : Random Walk

- The basic idea is that you always start in state 'D' and you move randomly, with a 50% probability, to either the left or right until you reach the terminal or ending states 'A' or 'G'. If you end in state 'A' you get a reward of 0, but if you end in state 'G' the reward is 1. There are no rewards for states 'B' through 'F'.

TD(1) Update

E.g.

$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$
 $B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$

- G_t is the discounted sum of all the rewards seen in our episode.
- So as we're traveling through our 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 (R_{t+2}) and so on.
- From equation we can see that we discount (γ) more heavily in the future with γ^{T-1} . So if $\gamma=0.2$ and you're discounting the reward at time step 6, your discount value γ become γ^{6-1} which equals 0.00032. Significantly smaller after just 6 time steps.

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

Figure 2: Sum of Discounted Rewards

TD(1) Update

- We'll use the sum of discounted rewards G_t , 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 multiple by an alpha (α) term to adjust how much of that error we want to update by.
- Lastly we make the update by simply adding our pervious estimate $V(S_t)$ 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 time step and compared it to our original estimate of values (all zeros).
- We weight the difference and adjust our prior estimate. Then start over again. This is TD(1) update

TD(0) Algorithm

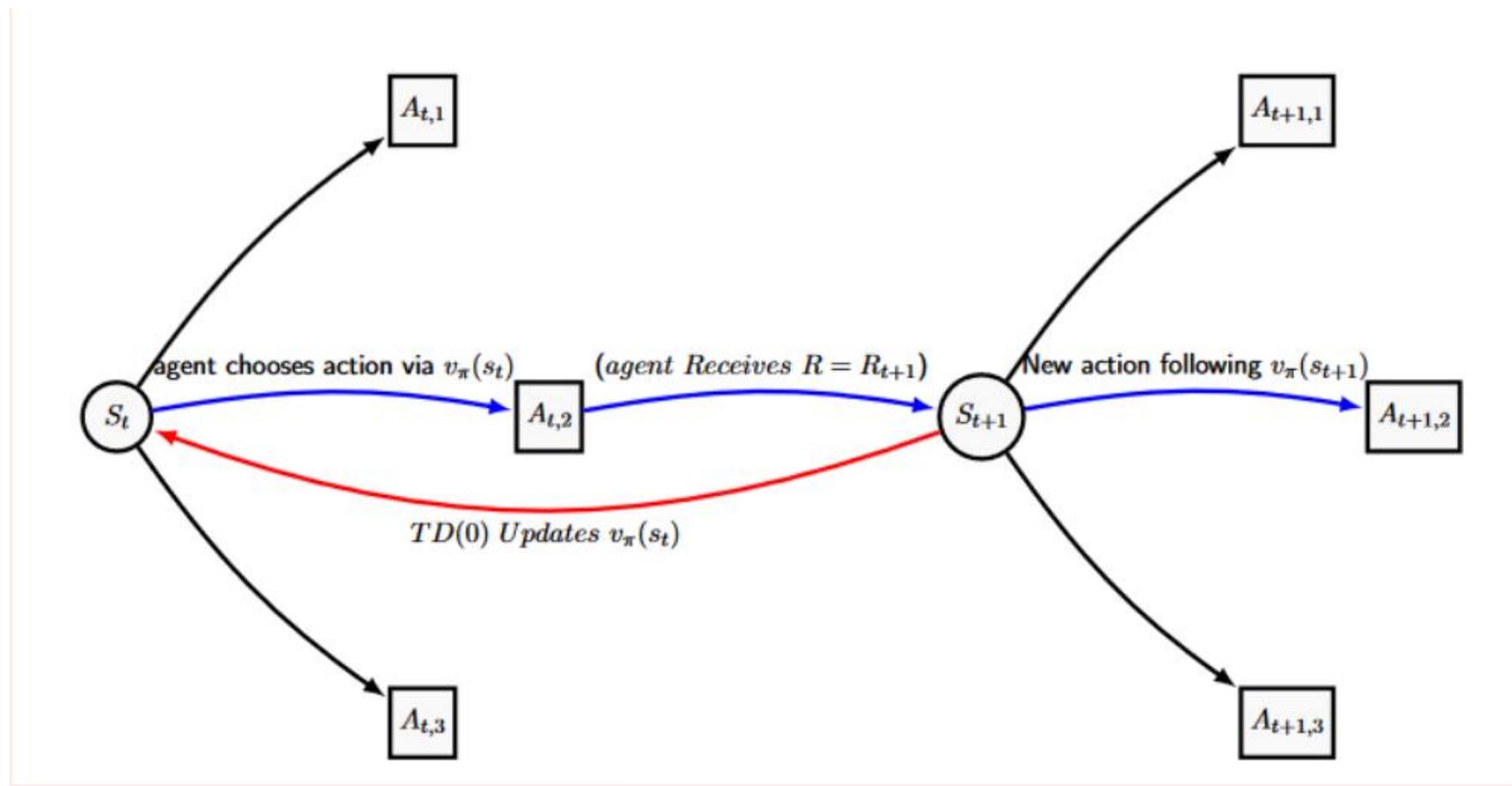
- Now that we've explained TD(1), TD(0) is easy to understand.

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)],$$

- Instead of using the accumulated sum of discounted rewards (G_t) we will only look at the immediate reward (R_{t+1}), plus the discount of the estimated value of **only 1 step ahead** ($V(S_{t+1})$) as:

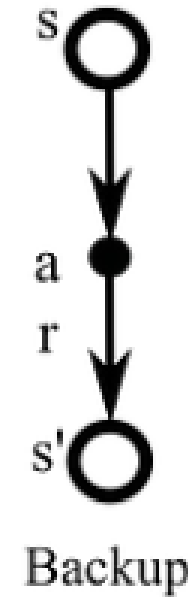
$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

TD(0) Update



TD(0)

- TD(0) is the simplest form of TD learning.
- In this form of TD learning, after every step value function is updated with the value of the next state and along the way reward obtained.
- This observed reward is the key factor that keeps the learning grounded and algorithm converges after a sufficient number of sampling (in the limit of infinity).
- The backup diagram of TD(0) and an example of TD(0) for examination example is given.



TD Error

- TD error arises in various forms throughout reinforcement learning and $\delta t = r_{t+1} + \gamma V(st+1) - V(st)$ value is commonly called the TD Error.
- Here the TD error is the difference between the current estimate for V_t , the discounted value estimate of V_{t+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.

TD(0) working

- TD(0) can be represented with the equation in the diagram.
- Equation 1 is generally shown in literature but find same equation written as per Equation 2 is more intuitive.
- We have α as a learning factor, γ as a discount factor.
- Here the value of a state S is getting updated in the next time step ($t+1$) based on the reward r_{t+1} observed after the time step t with the expected value of S in time step $t+1$.
- So its the bootstrap of S at time step t using the estimation from time step $t+1$ while r_{t+1} is the observed reward (real thing that makes the algorithm grounded)
- TD target and TD error as shown below are two important components of the equation which are used in many other areas of RL.

The diagram illustrates the TD(0) update equation and its components. It features two numbered equations, 1 and 2, each in a pink box. Equation 1 is $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$. Equation 2 is $V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}))$. A green arrow labeled 'Existing State Value' points from the $V(S_t)$ term in Equation 1 to the $V(S_t)$ term in Equation 2. A purple arrow labeled 'New State Value or the Target' points from the $R_{t+1} + \gamma V(S_{t+1})$ term in Equation 2 to the $R_{t+1} + \gamma V(S_{t+1})$ term in Equation 1. Below the equations, the TD target is defined as $TD\ target = R_{t+1} + \gamma V(S_{t+1})$, and the TD error is defined as $TD\ Error\ (\delta_t) = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$.

Existing State Value

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

2 $V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}))$

TD target = $R_{t+1} + \gamma V(S_{t+1})$

TD Error $(\delta_t) = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

New State Value or the Target

TD(0) Algorithm

```
Input: the policy  $\pi$  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  $\pi$  for  $S$ 
    Take action  $A$ ; observe reward,  $R$ , and next state,  $S'$ 
     $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
```

Example: Driving Home

Each day as you drive home from work, you try to predict how long it will take to get home.

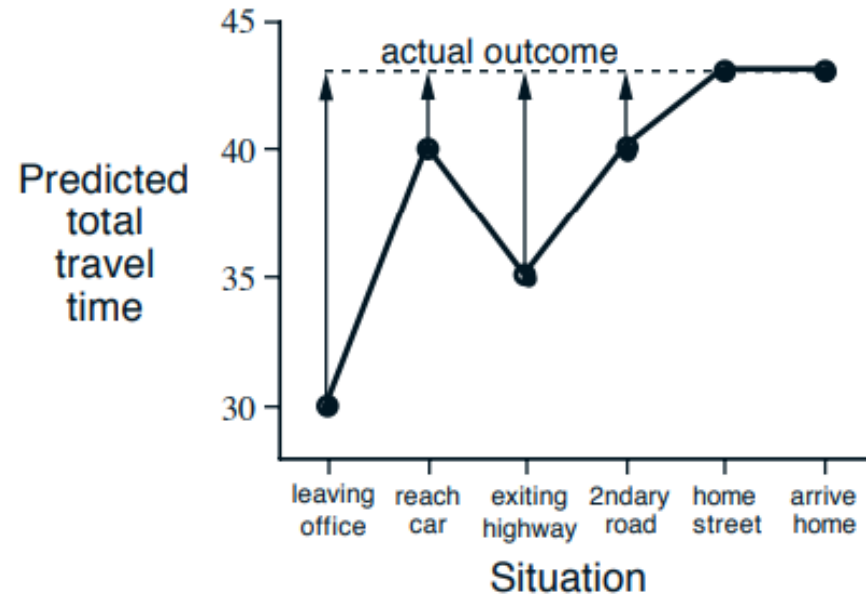
- When you leave your office, you note the time, the day of week, and anything else that might be relevant.
- Say on this Friday you are leaving at exactly 6 o'clock, and you estimate that it will take 30 minutes to get home.
- As you reach your car it is 6:05, and you notice it is starting to rain. Traffic is often slower in the rain, so you re-estimate that it will take 35 minutes from then, or a total of 40 minutes.
- Fifteen minutes later you have completed the highway portion of your journey in good time.
- As you exit onto a secondary road you cut your estimate of total travel time to 35 minutes.
- Unfortunately, at this point you get stuck behind a slow truck, and the road is too narrow to pass. You end up having to follow the truck until you turn onto the side street where you live at 6:40.
- Three minutes later you are home.

<i>State</i>	<i>Elapsed Time (minutes)</i>	<i>Predicted Time to Go</i>	<i>Predicted Total Time</i>
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

The sequence of states, times and predictions

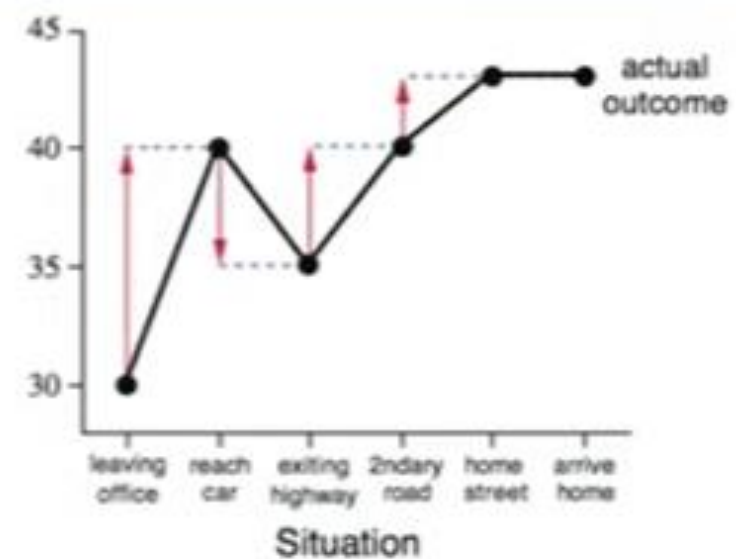
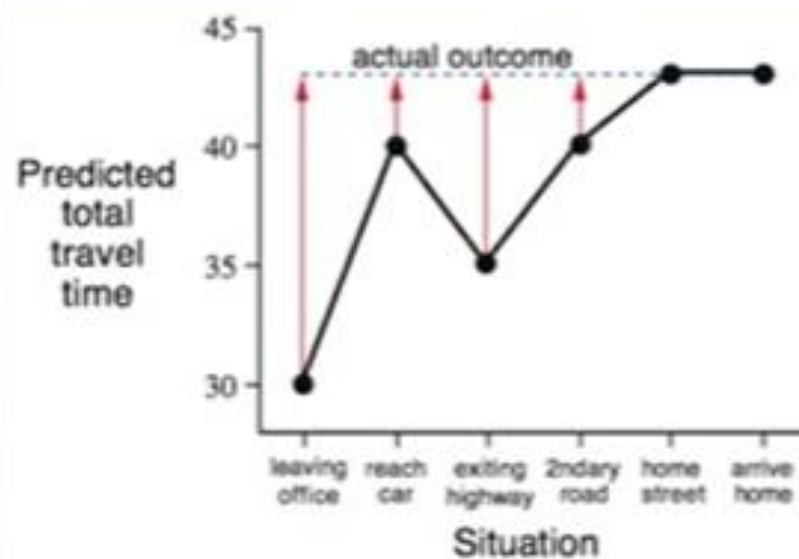
Example: Driving Home

<i>State</i>	<i>Elapsed Time (minutes)</i>	<i>Predicted Time to Go</i>	<i>Predicted Total Time</i>
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 $\alpha = 1$.
- 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.

<i>State</i>	<i>Elapsed Time (minutes)</i>	<i>Predicted Time to Go</i>	<i>Predicted Total Time</i>
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

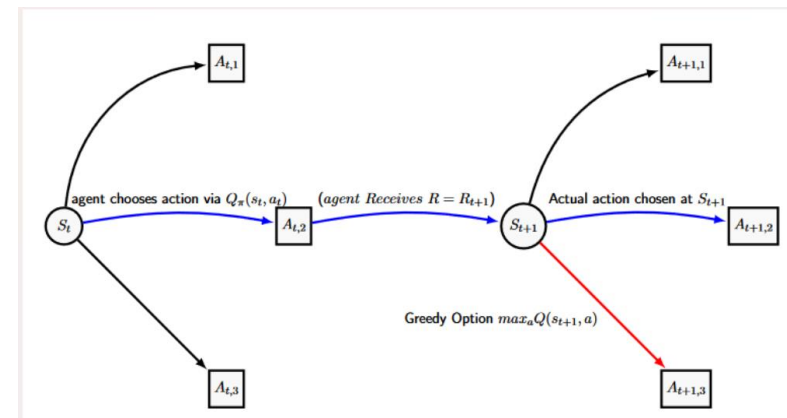
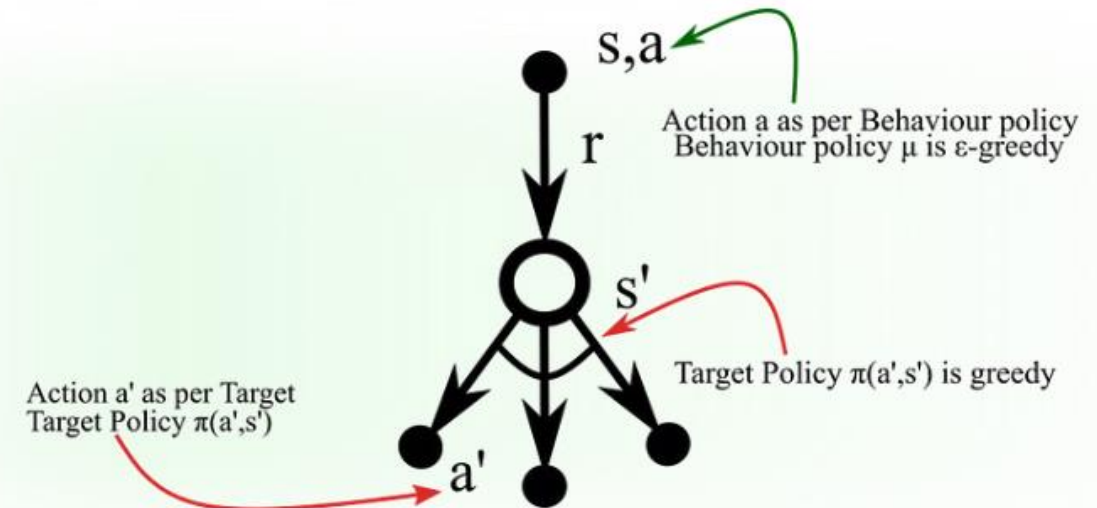


Advantages of TD Prediction methods

- 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

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

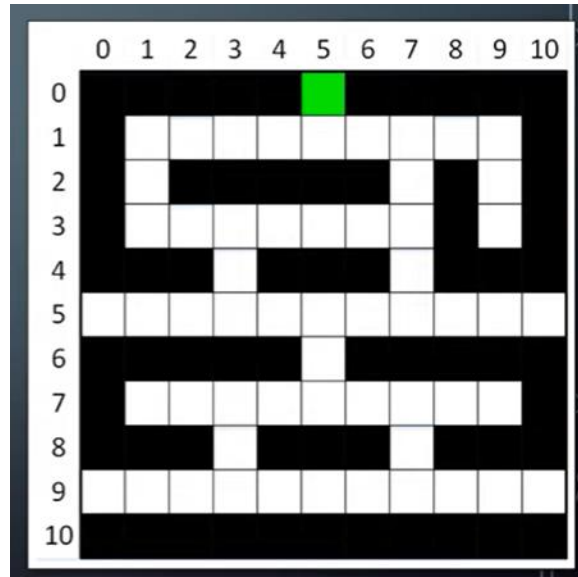
Current action as per Behavior Policy

Next Action As per Target policy

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

Q-values are stored in a *Q-table*, which has one row for each possible state, and one column for each possible action

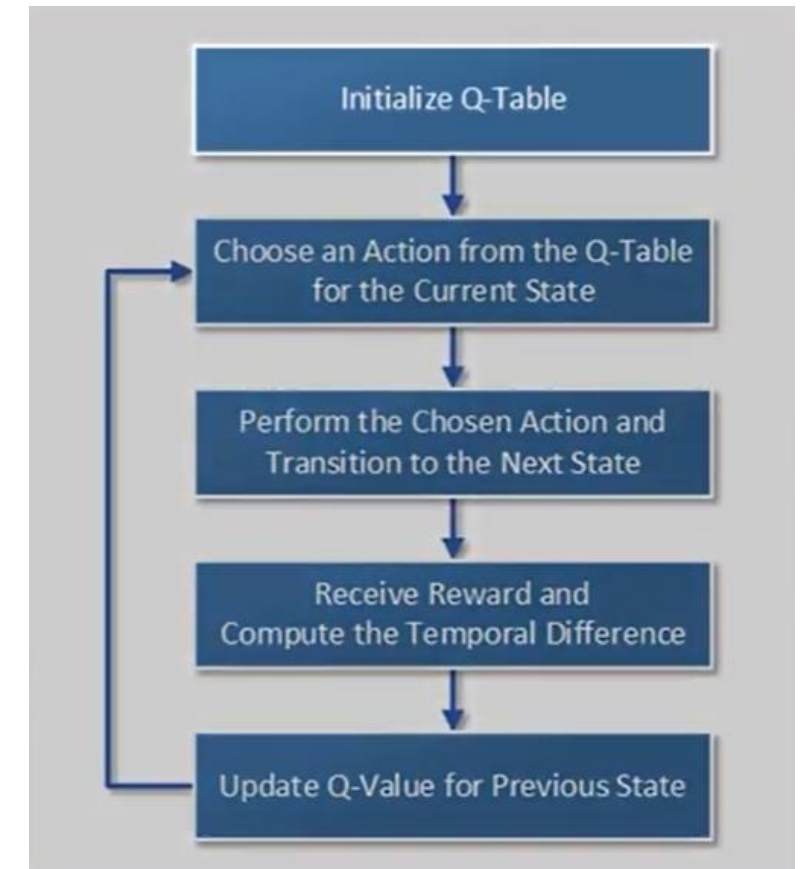
- An optimal Q-table contains values that allow the AI agent to take the best action in any possible state, thus providing the agent with the optimal path to the highest reward
- The Q-table therefore represents the AI agent's *policy* for acting in the current environment



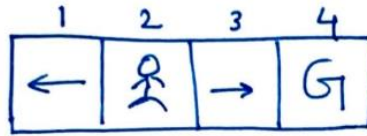
- Our e-commerce company's warehouse can be represented as a diagram:
 - Each black square is an item storage location (e.g., a shelf)
 - Each white square is part of an aisle where the robots can travel
 - The green square is the item packaging area

Q- Learning Algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
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'
 $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
 $S \leftarrow S'$;
 until S is terminal



Example: Grid world



Reward $\Rightarrow +1$ if reaches goal (G).

Actions \Rightarrow Left, Right

States $\Rightarrow 1, 2, 3, 4$

Q-table

		Actions \rightarrow	
		L	R
states \uparrow	1	$Q(1,L)$	$Q(1,R)$
	2	$Q(2,L)$	$Q(2,R)$
	3	$Q(3,L)$	$Q(3,R)$
	4	$Q(4,L)$	$Q(4,R)$

$\gamma = 1$

$\alpha = 0.1$

		L	R
1		0	0
2		0	0
3		0	0
4		0	0

$\rightarrow Q(3,R)$

$$Q(3,R) = 0 + 0.1(1 + 1 \times 0 - 0)$$

$$= 0 + 0.1$$

$$= 0.1$$

$$Q(2,R) = 0 + 0.1(0 + \gamma \times (0.1) - 0)$$

$$= 0 + 0.1 \times 0.1$$

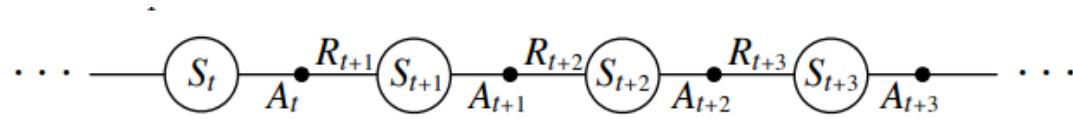
$$= 0.01$$

& so on...

Bootstrapping

- When you estimate something based on another estimation is called bootstrapping.
- In the case of Q-learning for example this is what is happening when you modify your current reward estimation r_t by adding the correction term $\max_{a'} Q(s', a')$ which is the maximum of the action value over all actions of the next state.
- Essentially we are estimating our current action value Q by using an estimation of the future Q .

SARSA Algorithm



- One of the TD algorithms for **control or improvement** is SARSA.
- SARSA name came from the fact that agent takes one step from one state-action value pair to another state-action value pair and along the way collect reward R (so its the $S_t, A_t, R_{t+1}, S_{t+1}$ & A_{t+1} tuple that creates the term **S,A,R,S,A**).
- SARSA is an **on-policy** method. SARSA use action-value function Q and follow the policy π .
- **GPI** is used to take action based on policy π (**ϵ -greedy** to ensure exploration as well as greedy to improve the policy).

SARSA Update

- we consider transitions from state–action pair to state–action pair, and learn the value of state–action pairs.

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

- This update is done after every transition from a nonterminal state S_t .
- If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is defined as zero. This rule uses every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state–action pair to the next.
- This quintuple gives rise to the name Sarsa for the algorithm.

SARSA Algorithm





Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
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)
 $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
 $S \leftarrow S'; A \leftarrow A';$
 until S is terminal

Frozen Lake Problem



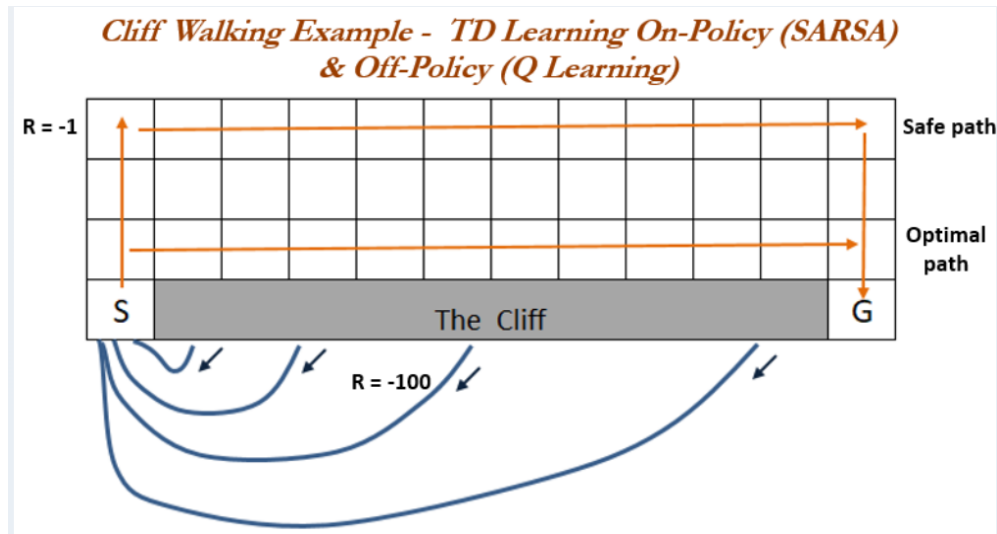
- **S**: The starting point. This is where the agent will start traversing from.
- **F**: Frozen. The lake is frozen at this point. It's safe to step here.
- **H**: Hole. There is a hole here, no go zone. This is one of the terminal states.
- **G**: Goal. This is where the frisbee is, the zone where the agent should be going. This is the desired terminal state

Frozen Lake Problem

State	 LEFT	 DOWN	 RIGHT	 UP
$S=0$	$Q(0, \text{left})$	$Q(0, \text{down})$	$Q(0, \text{right})$	$Q(0, \text{up})$
1	$Q(1, \text{left})$	$Q(1, \text{down})$	$Q(1, \text{right})$	$Q(1, \text{up})$
2	$Q(2, \text{left})$	$Q(2, \text{down})$	$Q(2, \text{right})$	$Q(2, \text{up})$
...
14	$Q(14, \text{left})$	$Q(14, \text{down})$	$Q(14, \text{right})$	$Q(14, \text{up})$
$G=15$	$Q(15, \text{left})$	$Q(15, \text{down})$	$Q(15, \text{right})$	$Q(15, \text{up})$

Q-Learning Vs SARSA

Cliff Walking Problem:

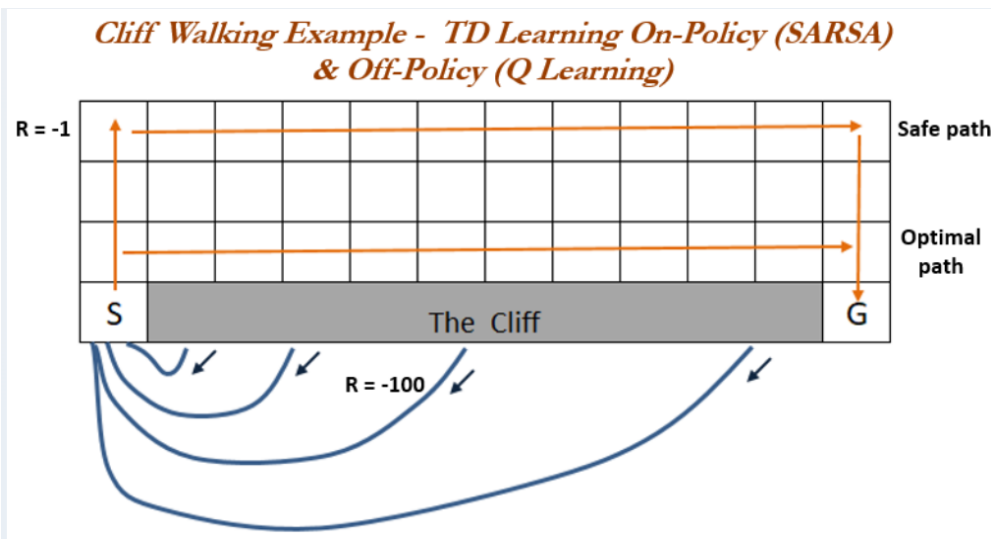


- This is a standard undiscounted, episodic task, with start and goal states, and the usual actions causing movement up, down, right, and left.
- Reward is -1 on all transitions except those into the region marked "The Cliff."
- Stepping into this region incurs a reward of -100 and sends the agent instantly 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, $\epsilon = 0.1$.
- 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



On policy and Off Policy

- The reason that Q-learning is off-policy is that it updates its Q-values using the Q-value of the next state s' and the *greedy action* a' . In other words, it estimates the *return* (total discounted future reward) for state-action pairs assuming a greedy policy were followed despite the fact that it's not following a greedy policy.
- The reason that SARSA is on-policy is that it updates its Q-values using the Q-value of the next state s' and the *current policy's* action a'' . It estimates the return for state-action pairs assuming the current policy continues to be followed.
- The distinction disappears if the current policy is a greedy policy. However, such an agent would not be good since it never explores.

References

- <https://www.lancaster.ac.uk/stor-i-student-sites/jordan-j-hood/2021/04/12/reinforcement-learning-temporal-difference-td-learning/>
- <https://web.stanford.edu/group/pdplab/pdphandbook/handbookch10.html>
- <https://towardsdatascience.com/temporal-difference-learning-47b4a7205ca8>
- <https://www.engati.com/glossary/temporal-difference-learning>
- <https://medium.com/@violante.andre/simple-reinforcement-learning-temporal-difference-learning-e883ea0d65b0> (td(0),td(1) and td(lambda))