## 1. What is temporal difference learning?

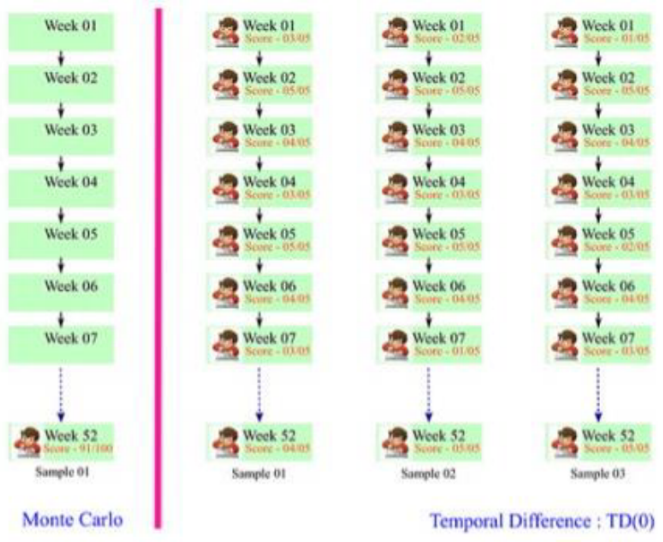
TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based on other learned estimates, without waiting for a 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 vл 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. What are the advantages of Temporal Difference Learning over Monte Carlo methods?

- 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.

## 3. 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.

## 4. 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:

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

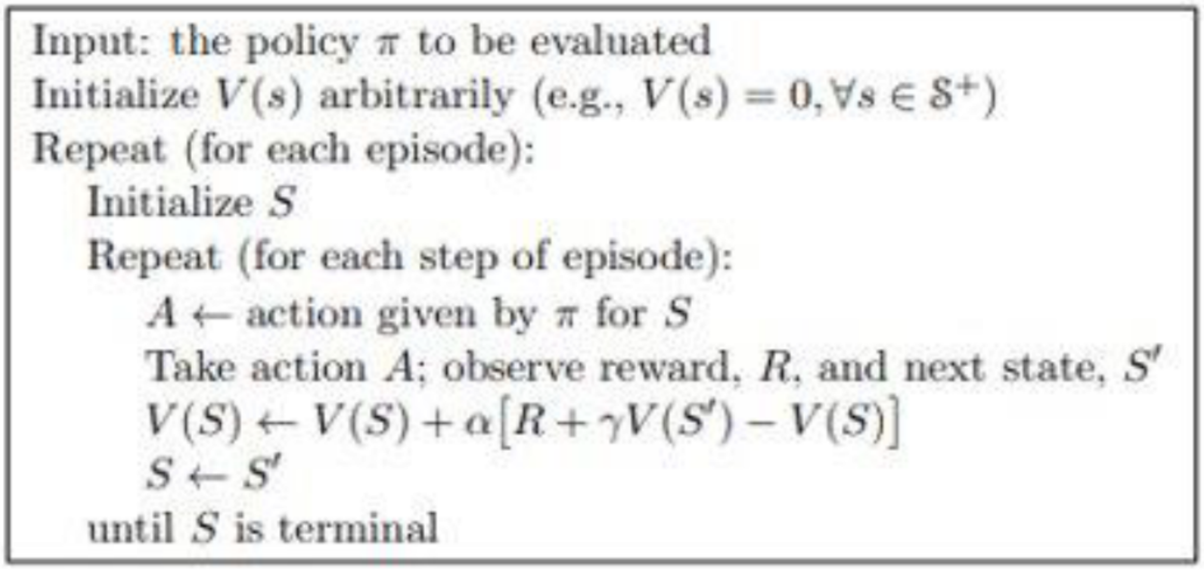
Figure 3: TD(1) Update Value toward Action Return

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:

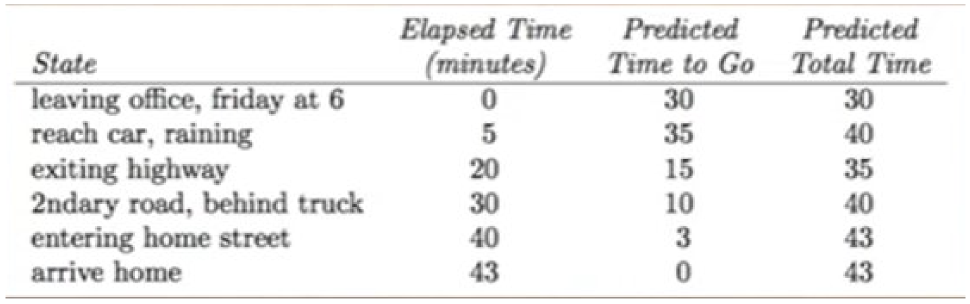
$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

5. 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.
6. Write pseudocode for TD(0) algorithm.

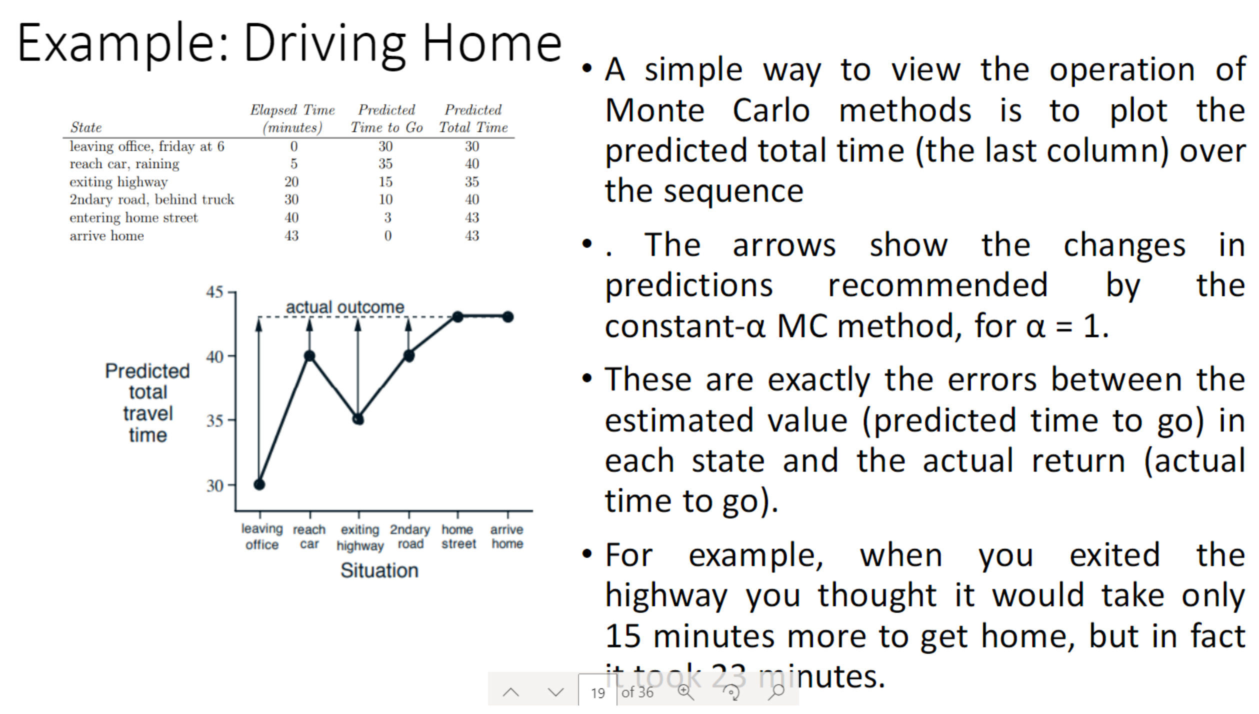


```
Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$; observe reward,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

7. 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 |

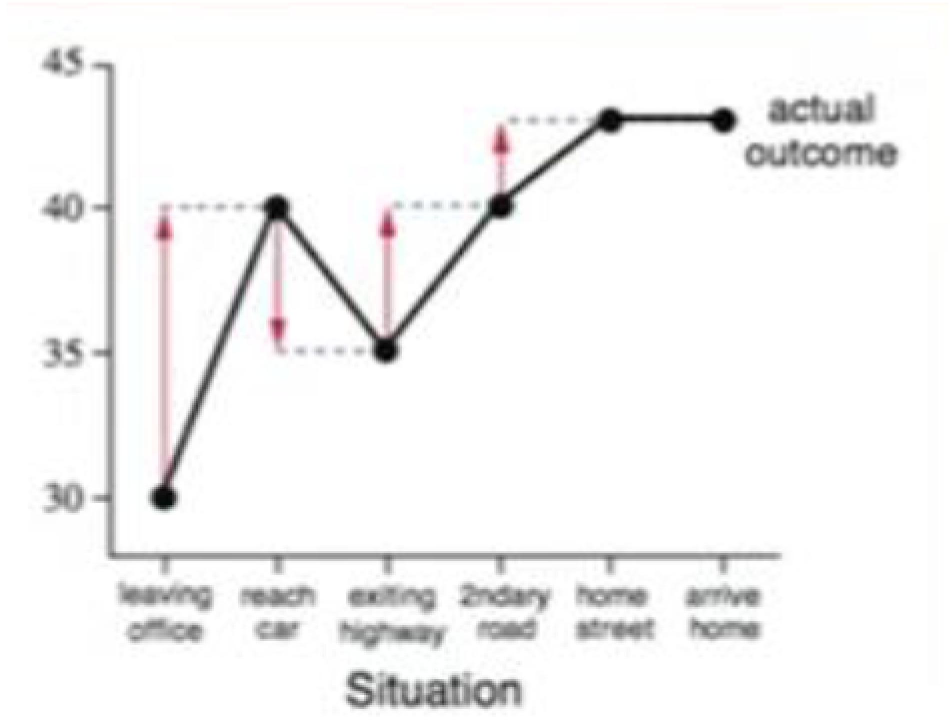


## Example: Driving Home

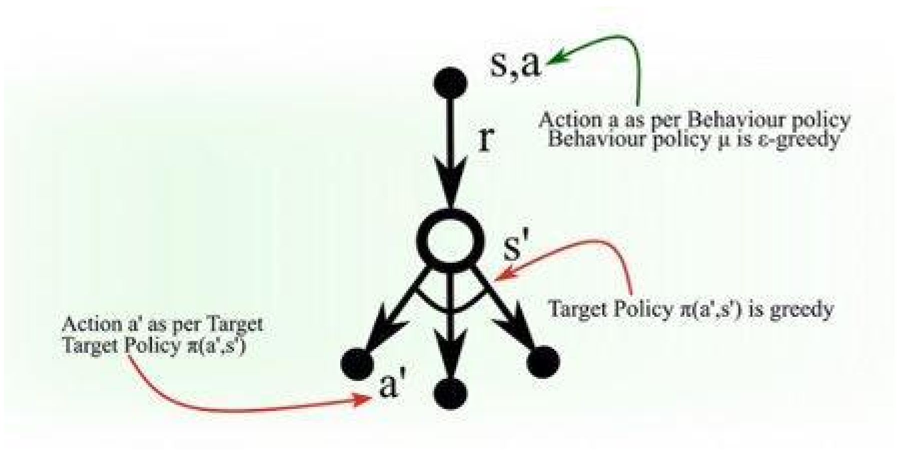
| State | Elapsed Time (minutes) | Predicted Time to Go | Predicted Total Time |
|---|---|---|---|
| leaving office, friday at 6 | 0 | 30 | 30 |
| reach car, raining | 5 | 35 | 40 |
| exiting highway | 20 | 15 | 35 |
| 2ndary road, behind truck | 30 | 10 | 40 |
| entering home street | 40 | 3 | 43 |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

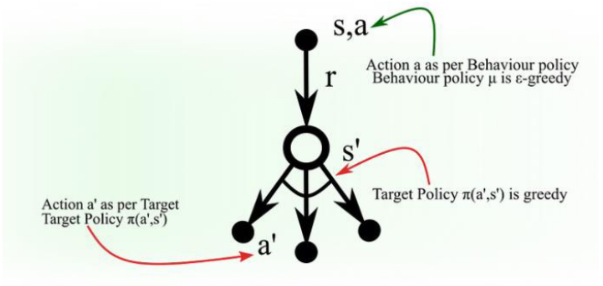


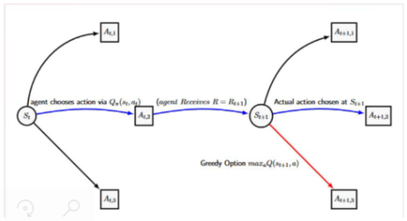
8. Identify the TD algorithm for given backup diagram and also state its update rule.


s,a
r
Action a as per Behaviour policy
Behaviour policy μ is ε-greedy
s'
Target Policy π(a',s') is greedy
Action a' as per Target
Target Policy π(a',s')
a'

## 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).

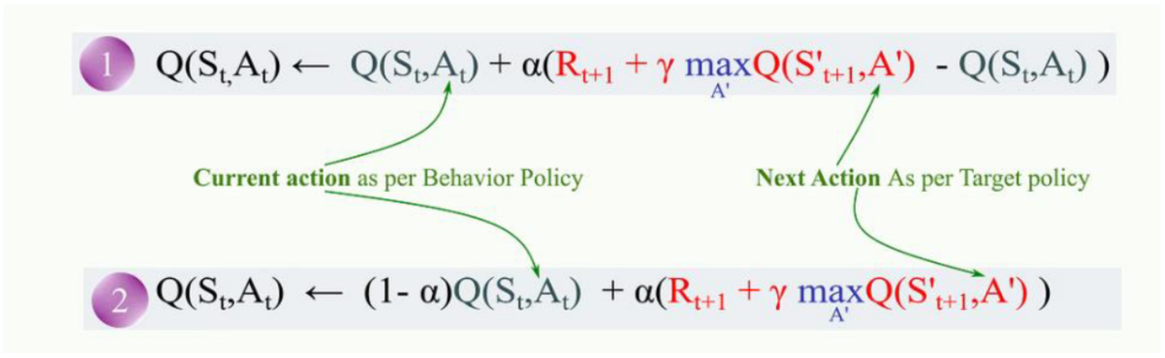

s,a
r
Action a as per Behaviour policy
Behaviour policy μ is ε-greedy
s'
Target Policy π(a',s') is greedy
Action a' as per Target
Target Policy π(a',s')
a'


$A_{t,1}$
$S_t$
agent chooses action via $Q_\pi(s_t, a_t)$
$A_{t,2}$
$(agent\ Receives\ R = R_{t+1})$
$S_{t+1}$
Actual action chosen at $S_{t+1}$
$A_{t+1,1}$
$A_{t+1,2}$
Greedy Option $max_a Q(s_{t+1}, a)$
$A_{t,3}$
$A_{t+1,3}$

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

## Q- Learning Update

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


Current action as per Behavior Policy
Next Action As per Target policy

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

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

9. 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.

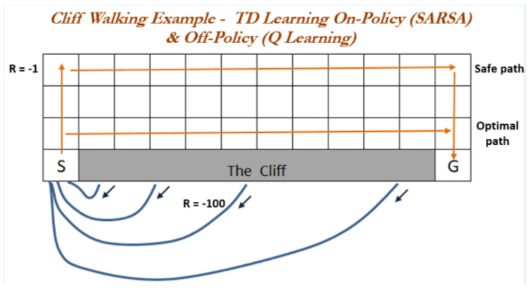
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

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

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

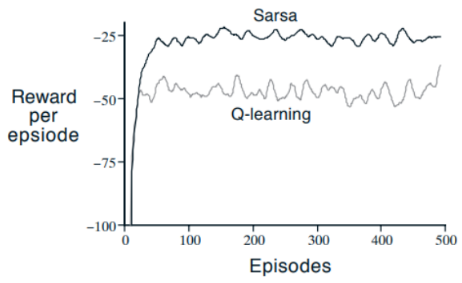
## Q-Learning Vs SARSA

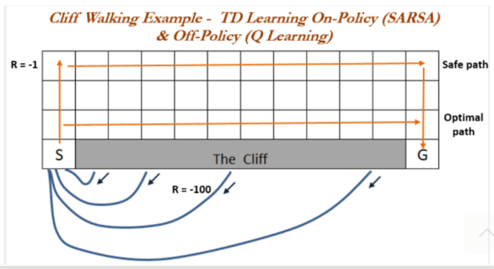
### Cliff Walking Problem:


Cliff Walking Example - TD Learning On-Policy (SARSA) & Off-Policy (Q Learning)
R = -1
Safe path
Optimal path
S
The Cliff
G
R = -100

- 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




Cliff Walking Example - TD Learning On-Policy (SARSA) & Off-Policy (Q Learning)
R = -1
Safe path
Optimal path
S
The Cliff
G
R = -100

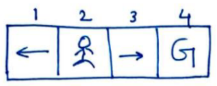
- The graph shows the performance of the Sarsa and Q-learning methods with ε-greedy action selection, ε = 0.1.
- After an initial transient, Q-learning learns values for the optimal policy, that which travels right along the edge of the cliff. Unfortunately, this results in its occasionally falling off 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

11. Write the pseudocode for Q learning.

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

12. Numericals based on Q learning and SARSA algorithm to update the Q table.

## Example: Grid world



Reward ⇒ +1 if reaches goal (G).

Actions ⇒ Left, Right

States ⇒ 1, 2, 3, 4

Q- table

Actions →

| states ↑ | L | R |
|---|---|---|
| 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 | $\bigcirc\!\!\!\!0 \rightarrow Q(3,R)$ |
| 4 | 0 | 0 |

$$Q(3,R) = 0 + 0.1\,(1 + 1 \times 0 - 0)$$
$$= 0 + 0.1$$
$$= 0.1$$
$$Q(2,R) = 0 + 0.1\,(0 + \gamma \times (0.1) - 0)$$
$$= 0 + 0.1 * 0.1$$
$$= 0.01$$

& so on.