Chapter 14

Reinforcement Learning

"Properly used positive reinforcement is extremely powerful."
— **B.F. Skinner**

Reinforcement learning is a branch of machine learning. There are many applications of reinforcement learning such as games, robotics, and resource management. This chapter aims to provide the basics of reinforcement learning.

Learning Objectives

- Get an overview of Reinforcement Learning
- Provide a mathematical formulation of Reinforcement problem
- Learn the basics of Markov decision process
- Introduce the basics of Dynamic Programming
- Know about Model free methods
- Introduce *Q*-Learning Algorithm
- Gain knowledge about SARSA algorithm

14.1 OVERVIEW OF REINFORCEMENT LEARNING

Reinforcement learning (RL) mimics human beings and is a branch of machine learning. Humans observe the environment through senses such as eye and ear. The inputs are processed by brain. The brain then suggests the actions and acts voluntarily or involuntarily.

In humans, learning happens in a real world called environment. Take a kid for an example. How does the kid learn? The kid interacts with the environment and gains valuable experience. Experience leads to learning. For example, when a kid accidently burns his hand in fire, he learns to stay away from it in the future and be careful. Similarly, the kid learns to do good things repeatedly by the encouragement or gifts given by parents or teachers. Awards can be positive or negative. Awarding a kid is a positive reinforcement and scolding is a negative reward or deterrent as shown in Figure 14.1. Thus, kids gain experience and learn many things through rewards and punishments (negative rewards).

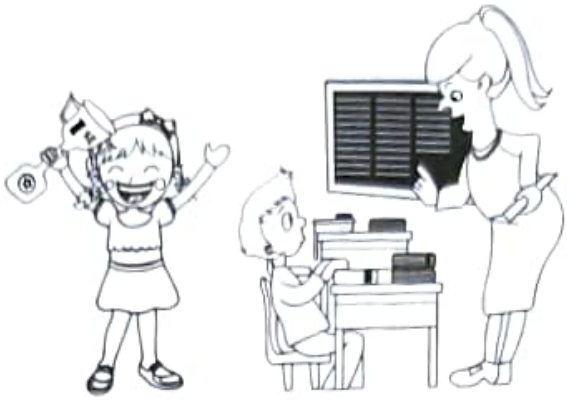

Figure 14.1: Examples of Positive and Negative Rewards

Scan for *'Additional Information'*

In computers, these scenarios can be simulated in many ways. As a kid executes many actions and receives a mix of positive and negative rewards that lead to gaining experience and learning, a computer program or robot can learn through experiences of simulated scenarios. Thus, reinforcement learning is a mathematical framework for learning. There are two types of reinforcement learning – positive and negative. Positive reinforcement learning is a recurrence of behaviour due to positive rewards. Rewards increase strength and the frequency of a specific behaviour. This encourages to execute similar actions that yield maximum reward. Similarly, in negative reinforcement learning, negative rewards are used as a deterrent to weaken the behaviour and to avoid it. In a maze game, there may a danger spot that may lead to loss. Negative rewards can be designed for such spots so that the agent does not visit that spot. Positive and negative rewards are simulated in reinforcement learning, say +10 for positive reward and –10 for some danger or negative reward.

Reinforcement learning is an example of semi-supervised learning technique and is used to model sequential decision-making process.

14.2 SCOPE OF REINFORCEMENT LEARNING

What are the situations where reinforcement learning can be used? Consider the following grid game shown in Figure 14.2, where a robot can move. Assume the starting node is *E* and goal node is *G*, the game is about finding the shortest path from starting to goal state.

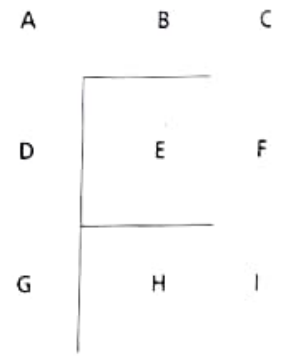

Figure 14.2: A Grid Game

Consider another grid game as shown in Figure 14.3.

In this grid game, the grey tile indicates the danger, black is a block and the tile with diagonal lines is the goal. The aim is to start, say from bottom-left grid, using the actions left, right, top and bottom to reach the goal state.

Reinforcement learning is highly suitable for solving problems like this, especially the ones with uncertainty. Reinforcement is not

suitable for environments where complete information is available. For example, the problems like object detection can be better solved using a classifier than by reinforcement learning.

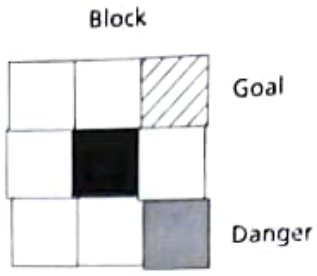

Figure 14.3: A Grid Game

Characteristics of Reinforcement Learning

1. Sequential decision making – Consider the Figure 14.3. It can be seen the path from start to goal is not done in one step. It is a sequence of decisions that leads to the goal. One wrong move may result in a failure. This is the main characteristic of reinforcement learning.
2. Delayed feedback – Often, rewards are not immediate. One must spend many moves to get final success or failure. Feedback in terms of reward is often delayed.
3. The agent actions are interdependent as any action affects the subsequent actions. For example, one wrong move of an agent may lead to failure.
4. Time related – All actions are associated with time stamps inherently as all actions are ordered as per the timeline inherently.

Challenges of Reinforcement Learning

1. Reward design is a big challenge as in many games, as determining the rewards and its value is a challenge.
2. Absence of a model is a challenge – Games like chess have fixed board and rules. But, many games do not have any fixed environment or rules. There is no underlying model as well. So, simulation must be done to gather experience.
3. Partial observability of states – Many states are fully observable. Imagine a scenario in a weather forecasting where the uncertainty or partial observability exists as complete information about the state is simply not available.
4. Time consuming operations – More state spaces and possible actions may complicate the scenarios, resulting in more time consumption.
5. Complexity – Many games like GO are complicated with much larger board configuration and many possibilities of actions. So, labelled data is simply not available. This adds more complexity to the design of reinforcement algorithms.

Applications of Reinforcement Learning

There are many applications of RL. Some of the application domains where reinforcement learning is used are listed below:

- Industrial automation
- Resource management applications to allocate resource
- Traffic light controller to reduce congestion of traffic
- Personalized recommendation systems like news
- Bidding for advertisement

- Customized applications
- Driverless cars
- Along with deep learning games like Chess and GO
- Deep mind applications like to generate programs and images

Scan for information on *'Context of Reinforcement Learning' and 'SoftMax Method'*

14.3 REINFORCEMENT LEARNING AS MACHINE LEARNING

In supervised learning, there is a supervisor but in reinforcement learning the supervisor is absent. Supervised learning learns using labelled data. It maps the input to fixed, pre-defined output. Supervised learning has a luxury of labelled data. In reinforcement learning, there is no labelled data. Instead, the labels are generated by the interaction of the agent with the environment. The agent may be a human or a computer program such as a chatbot. It can be obvious that games like Chess and GO cannot be solved using supervised learning as training dataset of these games cannot be generated. In the absence of the training set, the only option left is reinforced learning where the agent can interact with the environment and learns by trial and error.

Thus, one can observe the difficulties faced by reinforcement learning when compared with other machine learning algorithms. In supervised classifiers, the choice is one-shot, say cancer or not cancer. But in reinforcement learning, it is a sequential decision making, say finding the path from start to goal state. One must wait for more moves to see success or failure. The reason behind success and failure is found by observing the patterns over a long time and these decisions have longer consequences. For example, in a game, one wrong move may result in failure.

Reinforcement learning is different from supervised learning. The differences are listed in the following Table 14.1.

Table 14.1: Differences between Reinforcement Learning and Supervised Learning

Reinforcement Learning	Supervised Learning
No supervisor and labelled dataset initially	Presence of supervisor and labelled data
Decisions are dependent and are made sequentially	Decisions are independent of each other and based on input given in the training phase
Feedback is not instantaneous and delayed by time	Usually, the feedback is instantaneous once the model is created
Agent action affects the next input data	Dependent of initial input or the input given at start
No target values, only goal oriented	Target class is predefined by the problem
Example: Chess, GO, Atari Games	Example: Classifiers

The differences between reinforcement learning and unsupervised learning are listed in Table 14.2.

Table 14.2: Differences between Reinforcement Learning and Unsupervised Learning

Reinforcement Learning	Unsupervised Learning
Mapping from input to output is present	Not present
Get constant feedback from environment	No feedback from environment

14.4 COMPONENTS OF REINFORCEMENT LEARNING

The components of reinforcement learning are shown in Figure 14.4. These are environment, agent, actions and rewards.

Figure 14.4: Basic Components of RL

Reinforcement Problems

There are two types of problems in reinforcement learning – Learning and Planning.

In learning problems, the environment is unknown and the agent learns by trial and error. The agent interacts with the environment to improve policy.

Planning is another problem where the environment is known and the agent computes with the model and improves policy.

Environment and Agent

Environment is the world where all actions take place. It is the framework, where the input, output and reward are specified. The environment describes the state or state variables or simply as state. Initially, the environment is in a state called initial state.

For example, in a car system, the maps, game rules and obstructions in the road are described in the environment.

An agent is an autonomous body that looks at the environment and takes an action. It can be any human or another computer program such as a robot or chatbot.

States and Actions

The input for reinforced learning is called a state and the output is action.

Consider the following graph in Figure 14.5 generated by two actions, up and down, by an agent.

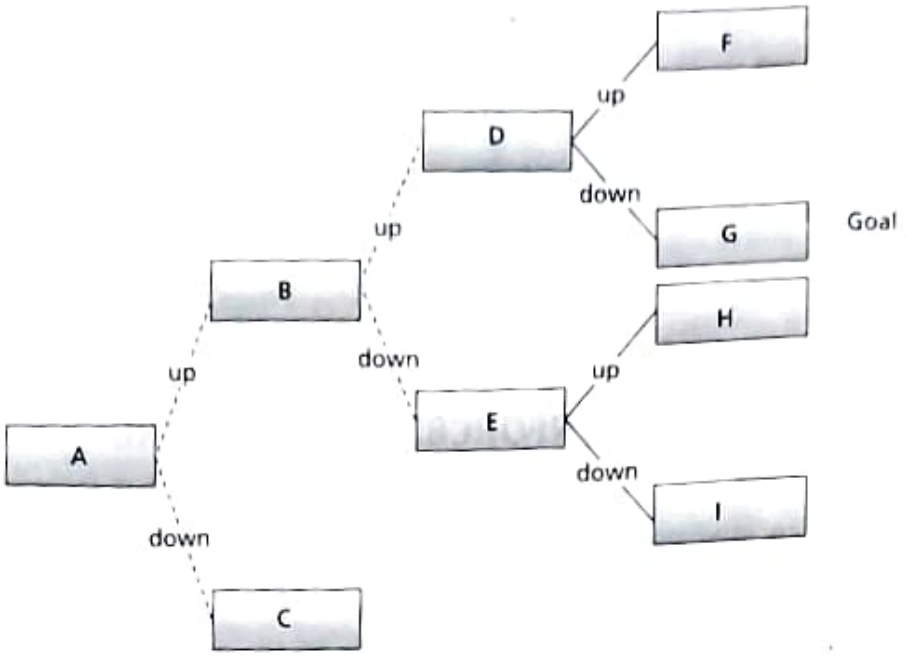

Figure 14.5: A Game: UP Action Results in Darker Line and Down Operation Results in Dotted Lines

In Figure 14.5, the states are A, B, C, D, E, F, G, H and I. These states can be cities of a shortest path problem or any other state encountered by a robot navigation. Let A be the starting state and G the goal or target state to be reached. The symbol '*S*' is used to denote the general state and '*s*' the specific state. 's_t' is used to denote a state at time '*t*'.

Other important nodes are:

1. GOAL node – This is also known as terminal node and absorbing state. It is the goal and the agent aims to get as it is the highest discounted cumative node.
2. Non-terminal node – All other nodes are called non-terminal nodes.
3. Start – The initial state.

Actions are transitions between states, say the path between A and B is caused by UP action. Then, the agent collects the rewards. In the game shown in Figure 14.5, the actions are UP and DOWN only. Symbol A is used to denote the general actions – UP and DOWN. The symbol '*a*' is used to specify the specific action and 'a_t' is used to denote the action at time '*t*'.

'Episodes' is the number of steps necessary to reach the goal state from start state. For example, in Figure 14.5, the aim is to find optimal path between A and G. There are two types of episodes.

One is called episodic and another is called continuous. In episodic tasks, there is a well-defined starting state and end state. End state is called as terminal state or goal state. An episode is one which consists of states, actions, and rewards. For example, all the paths from A to G, such as

$$A \rightarrow B \rightarrow D \rightarrow F \rightarrow G$$

$$A \rightarrow B \rightarrow E \rightarrow G$$

are all examples of episodes. An episode is one that starts from a starting state and reaches the goal state. So, a successful episode would be E-F-C-B-A-D-G.

Some times, there may not be any goal node. A continuous task has no terminal state, that is, no designated goal states.

G_t is called return or utility or total reward. It is a value that indicates the closeness of the goal state.

Its value ranges from 0 to 1. It gives more value for the immediate return. The unpredictability can be tackled by discount factor, which is the sum of the immediate reward and all other subsequent rewards multiplied by a factor of γ.

When γ is 1, the discount factor is close to the total reward. When γ is zero, only immediate reward matters and virtually all future actions are not taken into account. Often, the value of γ is chosen as 0.8 or 0.9.

Example 14.1: Let us assume that at the fifth step, the reward is +1. If the discounted factor γ is 0.7, what is the discounted reward or utility? Assume that the previous steps' rewards till the fifth step are zero.

Solution: The reward at 5th step is 1. Therefore, the discounted reward using the Eq. (14.4) is given as:

$$(0.7)^5 \times 1 = 0.16807$$

Broadly stated, reinforcement algorithms are classified as model-based and model-free methods based on the availability of models for the given problem. Certain real-word problems can be solved by constructing models using Markov decision process. Let us discuss about it now.

14.5 MARKOV DECISION PROCESS

Markov Decision Process (MDP) is an extension of Markov Chain. What is a Markov chain? A Markov chain is a stochastic process satisfying Markov property. It is a sequence of random variables $X_0, X_1, \ldots, X_n$ satisfying conditional probability.

A Markov chain built on two states is shown in Figure 14.6. Let us assume that there are two universities. Eighty percent students of university A leave for university B for getting a masters' degree and only 20% remain in the same university to study further. Similarly, 60% of students of university B move to university A and 40% remain to study in the same university. The Markov chain for this is given as follows:

It can be observed that there are two states. The transition between state *A* and state *B* is an edge with probability. For example, the transition between states *A* and *B* is 0.8. A transition matrix *P* of a Markov chain at time t is a matrix that has probability of transition that exists between two states. The transition matrix at time *t* is given as follows:

$$(P_t)_{i,j} = Pr(X_{t+1=j} \mid X_{t=i})$$

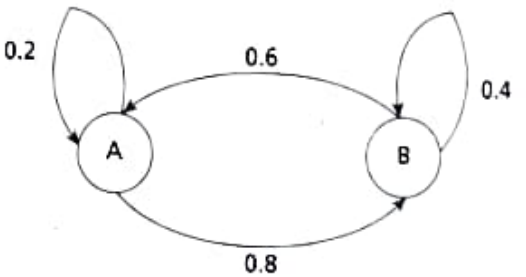

Figure 14.6: Markov Chain

The transition matrix for the Markov chain is given as follows:

$$P = \begin{pmatrix} 0.2 & 0.8 \\ 0.4 & 0.6 \end{pmatrix}$$

The sum of rows of *P* is 1. In other words, a row of the matrix is a probability vector.

If u be the probability vector that represents the initial state of the Markov chain, the probability of the chain in state 's' after 'n' steps is the i^{th} entry of the vector:

$$u_{n+1} = u_n \times P \tag{14.5}$$

The next state is computed by multiplying u and P.

Example 14.2: Assume the universities' hold on a town is initially 60% and 40%. So, what would be the prediction after one month, that is, the hold of university after an year. After two years, will the hold remain same or change? Assume the rest of the probability transition information from Figure 14.6.

Solution: The starting distribution is $u_0 = (60\%, 40\%) = (0.6, 0.4)$. The Markov chain is shown in Figure 14.6 and the transition matrix is given as:

$$P = \begin{pmatrix} 0.2 & 0.8 \\ 0.4 & 0.6 \end{pmatrix}$$

The prediction after a year as in Eq. (14.5), $u_1 = u_0 \times P$, will be:

$$= (0.6 \;\; 0.4)\begin{pmatrix} 0.2 & 0.8 \\ 0.4 & 0.6 \end{pmatrix}$$

$$= \begin{pmatrix} 0.28 \\ 0.72 \end{pmatrix}$$

The hold of the universities will be 28% and 72%. This is obvious because 80% of the students shift from the university A to B.

The second-year prediction $u_2 = u_1 \times P$ would be:

$$= (0.28 \;\; 0.72)\begin{pmatrix} 0.2 & 0.8 \\ 0.4 & 0.6 \end{pmatrix}$$

$$= \begin{pmatrix} 0.344 \\ 0.656 \end{pmatrix}$$

It can be observed that the hold on the student by the universities changes.

If rewards are added to markov chain, the markov decision process is obtained where the edges are associated with probability as well as reward.

As discussed earlier, MDP is an extension of Markov Chain. The components of MDP are:

1. A set of states
2. A set of actions
3. Reward Function R
4. Policy π
5. Value V

The simplest structure of MDP is Markov chain where the environment is modelled as a graph. The nodes of the graph represent states and edges represent transitions. Each edge is associated with a static probability.

Markov assumption holds good for reinforcement learning. Markovian property states that the probability of reaching a state s_t and getting the reward r_t depends solely on the preceding states s_{t-1} and r_{t-1}. All other previous states are not important. This is called Markov assumption.

A state is Markov if and only if:

$$p[s_{t+1} / s_t] = p[s_{t+1} / s_1, s_2, ..., s_t] \tag{14.6}$$

Here, s' and s are new and old states and 'a' is the action, and 'r' is the reward.

MDP performs the following actions:

- Observes the current state s_t
- Performs the action a_t
- Gets the reward $r(s_t, a_t)$
- Enters a new state s_{t+1}

The aim of the agent is always to maximize the total rewards accumulated over time. The movement of one state to another is called a transition. The agent manipulates the environment from one state to another.

The state transition probability is given as follows:

$$P^a_{ss'} = \{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{14.7}$$

It indicates the moving from states to s' after taking an action 'a'. In other words, given a state 's', it is the probability of the occurrence of next state s'. One can also represent it as a matrix called state transition matrix where each row indicates the transition probability from one state to another.

The expected value of the reward is given as:

$$R^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a\} \tag{14.8}$$

Training and Testing of RL Systems

Once the MDP is modelled, the next stage is called training and inference. The agent tries to accomplish a goal repeatedly and modify its parameter over time. This is called learning. Once the trained model is available, it is put to use and this process is called inference. At deployment level, training and inference go together and when environment is changed, training is done again. This is to improve the system.

14.6 MULTI-ARM BANDIT PROBLEM AND REINFORCEMENT PROBLEM TYPES

Reinforcement learning uses trial and error to learn a series of actions that maximize the total reward. There are two sub-problems in the reinforcement problem.

1. The main problem is the prediction of total reward or return. This is called prediction, policy evaluation or value estimation. It requires a formulation of a function called state-value function. The problem of estimating the state value function can be done using temporal differencing or TD-Learning.

14.9 Q-LEARNING

Q-Learning is an off-policy method because Q is updated based on policies that are implicit to Q and is better guaranteed for maximum returns. Q indicates Quality. What is Q-value? $Q(s, a)$ is a numerical value assigned to a state-action pair. It means a value of the action that is performed in state 's'.

Here, the main objective is to find Q-Value? Q-value is nothing but the immediate reward and other rewards that are yet to come, known as total return reward. To compute the total return reward, these information are necessary.

1. Starting state
2. Action
3. Reward
4. New state

Initially, a table called Q-table is constructed and filled with initial random values. Q-learning involves two policies - learning policy and update policy. Q-learning is done by methods like greedy, softmax or softmax plus.

The agent's next move will be the action where rewards are high. Alternatively, it would be the cell that has the highest Q-value. As the moves are determined by Q-values, the computation and updation of Q-values are important for Q-learning. Then comes the Q-learning update policy. The updation of Q-values is done using Temporal-Differencing method that is mentioned in section 14.8.2. This updation is made by blending the new and old values. Blending is done by α. When α is zero, the value does not change at all. When α is one, the next value replaces the old value. Here, α is known as the learning rate. The discount factor (γ) is also used for update. Blending is done using temporal difference learning.

The updation procedure of Q-value is given as follows:

1. Perform any random action on state s_t
2. Get a new state, s_{t+1}
3. Get the award $r(s_t, a_t)$

The temporal difference at time 't' is done using the update:

$$TD_t(s_t, a_t) = r(s_t, a_t) + \gamma \max_a (Q(s_{t+1}, a)) - Q(s_t, a_t) \tag{14.23}$$

Here, $r(s_t, a_t)$ is the reward obtained by performing action a_t and $Q(s_{t+1}, a)$ is the estimate of the best action at state s_{t+1} i.e., s' and $Q(s_t, a_t)$ is the Q-value of the action a_t at state s_t. Here, the best action performed at future states discounted by discount factor (γ) is $\gamma \max_a (Q(s_{t+1}, a)) - Q(s_t, a_t)$.

If TD is high, then it is a 'surprise factor' and denotes the highest reward. If TD is less, it represents the 'frustration' factor and denotes the lesser reward. In other words, TD is a sort of 'reward'. It is initially high and slowly gets minimized as it reaches the end of the training, that is, when it reaches the final goal.

The update is carried out using the Temporal difference learning (TD) and Bellman equation. The Bellman equation that is used to update the value is given as follows:

$$Q_t(s_t, a_t) = Q_t(s_t, a_t) + \alpha TD_t(s_t, a_t) \tag{14.24}$$

The Q-Learning algorithm is given as follows:

Algorithm 14.7: Q-Learning

1. Set $Q(s_t, a) = 0$ where s_t is the terminal node in an episode.
2. For all states s and action a, set Q Value = 0.
3. Repeat.
4. Select s_t randomly.
5. Choose action a_t from Q using ε- greedy.
6. Perform action a_t such that $r(s_t, a_t) > 0$ and reach the next state.
7. Update the action-value function using TD using Eq. (14.24) and Bellman equation:

$$Q_t(s_t, a_t) = Q_{t-1}(s_t, a_t) + \alpha \times TD_t(s_t, a_t) \tag{14.25}$$

Until the terminal state s is reached.

Here, α is the learning rate. It is between 0 and 1. If α is zero, then nothing is updated and there is no learning at all. When setting it to a higher value such as 0.9, the learning happens fast. $\gamma$is the discount factor. It is in the range 0 – 1. It should be less so that the algorithm can converge.

The inference is simple, the best action is the maximum of Q function as follows:

$$a_t = \arg\max_a (Q(s_t, a))$$

14.10 SARSA Learning

SARSA is based on episodic formulation and discounted factor. Its name is derived from the function $Q(s, a, r, s', a')$, that is, 'r' is the reward and $s'(s_{t+1})$ and $a'(a_{t+1})$ are the new states and actions, respectively. It selects a state 's' and actions 'a' are chosen based on the ε-greedy approach.

SARSA is an on-policy method because each action a_t is based on Q and Q is updated based on the actions taken. SARSA converges to optimal policy and with the action-value function provided all state-action pairs are visited infinitely often.

The major difference between SARSA and Q-learning is that not necessarily the maximum reward is used to update Q value. Instead, it is based on estimates. Thus, SARSA is an online method.

SARSA is based on estimated optimal action:

$$a^*_{t+1} = \arg\max_a Q(s_{t+1}, a) \tag{14.26}$$

The ideal rule for updating Q:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t) \tag{14.27}$$

This can be given in terms of reward as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t) \tag{14.28}$$

Where, α and γ are learning rate and discount factor, respectively.

The SARSA algorithm is given below.

Algorithm 14.8: SARSA

1. For all states and actions, set $Q(s_t, a) = 0$.
2. Initialize α, ε and γ.
   Initialize the state s_t.
   Choose action a_t based on selection policy ε-greedy strategy and execute it.
   Get a new state s_{t+1}, observe the reward r_{t+1}.
   Choose action a_{t+1} from Q based on ε-Greedy procedure and execute it.
   Update Action-value function of Eq. (14.28).
3. Update the values $s_t = s_{t+1}$, $a_t = a_{t+1}$.
4. If s' is terminal node, then start a new episode else repeat above steps.

The parameters of the SARSA algorithm as same as the Q-learning algorithm, that is, learning rate and discount factor.

Summary

1. Reinforcement learning is a branch of machine learning. In reinforcement learning, the agent learns by getting a reward from the environment. It aims to maximize the reward by taking suitable action.
2. Environment is the world where all actions take place. The environment is the framework, where the input, output and rewards are specified.
3. The input for reinforced learning is called a state. The output of reinforced learning is action. The environment is described as the state or state variables or simply as state.
4. Policy takes the input states and processes it, and then suggests an action.
5. Reward is a measure of agent performance.
6. When the environment immediately delivers reward after the execution of action, it is called an immediate reward. When the environment gives the reward after the end like the objective of the game, it is called a long-time reward.
7. The accumulation of all rewards collected over an action is called total reward.
8. Discount factor is the sum of the immediate reward and all other subsequent rewards multiplied by a factor of γ.
9. The interactions of an agent with the environment are modelled as Markov Decision Process (MDP).
10. Markov assumption holds good for reinforcement learning. The probability of reaching a state s_t and getting the reward r_t depends solely on the preceding states s_{t-1} and r_{t-1}. All other previous states are not important. This is called Markovian assumption.
11. Dynamic programming is used to solve subproblems. It is an off line algorithm as it needs to wait till the end of episode to update values and actions.
12. Model-based environment means that for any state, the next state and action, the probability distribution is known.