Deep Reinforcement Learning

Shiv Kumar

02/01/2019

1 Introduction

We learn using trial and error based learning, by gaining understanding of the world through interaction. We understand the cause and effects of the environment using this interaction, i.e. we understand the response to our actions and accordingly remember which actions yielded better results and which yielded bad results. After establishing an understanding of the world, we try to accomplish specific goals.

Maximising reward signals

The scientific understanding on how this learning happens, i.e. the computational approach is RL. We study simpler environments at start with well defined rules and dynamics and these simpler algorithms will obviously have certain limitations as well. But their analysis gives us a good starting point and also a motivation to study more complex algorithms.

2 Applications of DRL

1. Self Driving Cars
2. Games :
   - TD Gammon : RL agent to play backgammon. There are 10^{20} states which need to be decided on. This is pretty complex. This agent helps us gain more knowledge of the game which was constrained due to human knowledge till now and helps us devise better strategies that had never been thought of before.
   - Alphago : More states or configs in this game than total atoms in the world. The agent beat one of the best players of go 4-1.
   - Atari games : Learning to play games just from pixels.
3. Robotics : A robot learning to walk (analogous to how humans learn to walk).
4. Finance
5. Biology
6. Inventory Management

3 Basic Intuition

In RL, there is an agent which is primarily the learner or the decision maker in certain given situation. For example, a puppy which has just entered the world. It has a very complex body structure which allows it to execute different possible tasks(like sitting, walking, jumping, etc.). Now when being trained, given commands by its owner, it gets to choose between a lot of possible actions. If the action performed is in sync with the owner's commands, then the pup is rewarded with some biscuits or some other kind of reward(pat on the back, or anything else), whereas if it isn't in sync, then it isn't rewarded. The

pup has a lot of possible actions but does not really have a sense of the cause and effect of these actions initially i.e. it does not know doing which action for which command is right and hence will have to try it out.

Selection of action by random initially

Given a certain command, it chooses an action by random (having full understanding that it has no idea what it is doing). For example, the owner asks the pup to sit down, and choosing a random action, it decides to sit down, and then waits for a response to its action and receives a single treat which is encouraging. Now it is obvious behaviour pattern that an agent would want to yield maximum possible reward in a situation.

It is again given an instruction to maybe walk and it decides to perform some other action and waits for response but does not get any return this time, which can be said to relatively discouraging and hence tries to understand whether the action is bad in general or just a wrong mapping. As time goes on, it will be able to perform a lot more interaction with its owner and gain a better understanding of what command maps to what action and accordingly gain better ability to maximize reward i.e. more interaction, more feedback, better ability to map commands and actions and hence better chance to maximize reward.

This process of systematically proposing and testing hypothesis (*proposition made on the basis of reasoning, without any assumption of its truth*) is the basic concept behind RL.

Note: The situation is not simple as this in general, there are some other problems as well which need to be kept in mind while understanding RL, i.e.

- Exploration-Exploitation Dilemma :
  - Exploration : Exploring potential hypothesis for now to choose actions. This is important to increase knowledge.
  - Exploitation : Exploiting the already available knowledge to make the best possible guess. This is important to increase reward signal.

  The problem comes in when we need to decide how to balance these two. Whether to be satisfied by what we already know or to experiment and see if there is a better strategy to this given that it may result in a worse result.
- Delayed rewards: Certain strategies might have less pay-offs in the short term but higher in the long term and vice-versa. Making the decision on which action to take then is also a dilemma.

The main takeaway is that learning from experience is the main approach to learning here.

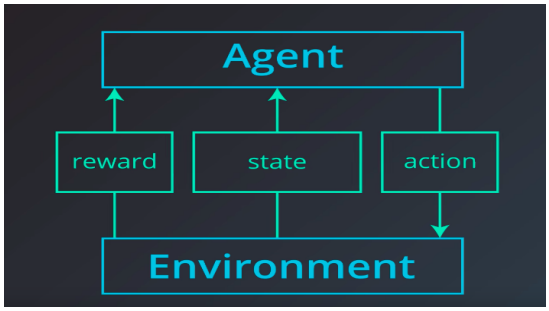
4 The RL framework : The problem

RL is basically an agent learning to interact with its environment to maximize some kind of reward signal or just reward. Here agent is the one who/which learns from trial and error(experience based learning) how to behave in an environment to maximize reward. Here we work with the assumption that time evolves in discrete time steps and at initial time-step, agent observes the environment. Also, the agent is able to fully observe what ever the state the environment is in at present.

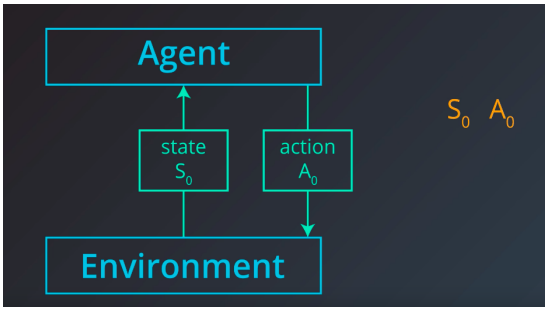
The agent receives an environment state at T_0, where it takes some action, based on which it receives an updated state and a corresponding reward as well, and this process goes on. This interaction of the agent is manifest as a sequence of states, actions and rewards.

$$\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4.., S_n, A_n, Rn + 1\}$$

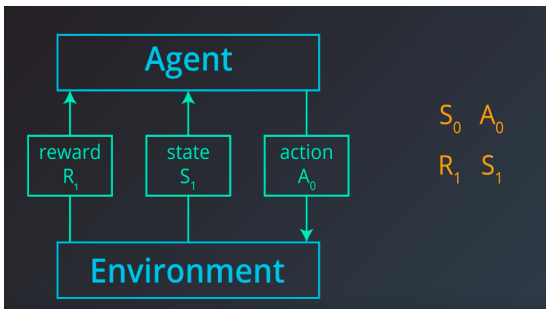
Now, the goal of any agent is to maximize the cumulative expected reward. Hence it should accordingly choose a strategy which results in such a maximized reward. This is done by interacting with the environment, i.e. the agent learns to play by the rules of the environment to complete a specific task or accomplish some goal.



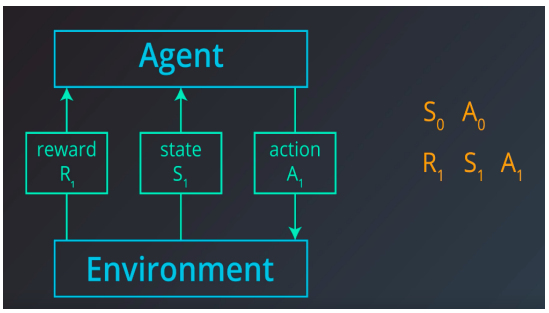
(a) The basic interaction



(b) Interaction at T_0



(c) Interaction post action at T_0



(d) Action for next time step

Figure 1: The basic RL framework

So we can define a mathematical model for a real world problem using the following:

- States
- Actions
- Rewards
- Rules of the environment

In RL there are majorly two types of tasks*(used to classify RL problems)*(instance of a RL problem, like if playing a game is a problem, a turn playing the game can be considered a task). They are episodic tasks and continuous tasks.

- Episodic tasks : Tasks(of problems) that have a defined terminal state or ending temp step such that games, car reaching its destination of crashing, walking robot agent falling. Here, when the episode ends, the agent looks at the total reward received to analyze performance. It then starts again with the gained knowledge, and over many episodes, agent starts making better decisions for itself, in order to choose a strategy to increase cumulative reward, which can be the game playing score in a game. The sequence of states, actions and rewards can be seen as follows :

$$\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4.., R_T, S_T\}$$

- Continuous tasks: These problems do not have an ending time step. Interaction with the environment continues without a pre-specified limit. For example, an algorithm that buys and sells stocks in response to the financial market. The sequence of states, actions and rewards can be seen as follows :

$$\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4.., R_T, S_T, ...\}$$

Considering the game of chess, where the opponent is a part of the environment, we are presented with a state of the environment(configuration of the state), and based on that, we need to select/or take action,

after which we get an updated configuration of the board, on the basis of which we are supposed to take another action and so on, till one doesn't win. This is a classic example of an episodic task. The reward here is received here only at the end of the game(delayed reward) i.e. +1, -1(sparse rewards). Since the reward is provided at the end, it would be unclear whether all moves were bad or whether the game was going well till one stage and one bad move was the mistake that ruined the whole game and same for positive reward as well.

4.1 Reward Hypothesis

In RL, all agents formalize their goals based on maximizing the expected cumulative reward, i.e. they specify a reward signal or function or formula, maximization of which results in the completion of some specified goal. This is the reward hypothesis. As in the case of the puppy, the reward was pretty easy to visualize, but how do we judge a robot which is learning to walk? It would be very subjective based on the viewer and trainer and hence we say that if the robot is able to maximize the reward function then it will achieve the goal we set for it that in this case, learning to walk with stability.

We use the the problem of teaching a robot to walk more deeply by trying to analyze the work done by Deepmind on the same, where the goal formally was for the robot to stay walking as long and quickly as possible while exerting minimum effort. First of all, to specify or frame this problem we need states, actions and rewards. This is considered an episodic tasks as it will end when the robot crashes to the ground and the reward will be considered till that time step.

- Actions : Decisions to be made in order for the robot to walk, as the humanoid has several joints, and we can specify actions as the amount of force to be applied to the joints in order to move.
- States : Context provided to the agent(robot) for choosing intelligent actions which include
  1. Current position and velocities of the joints
  2. Measurements of the surface i.e. information about the surface : Flat? Inclined? Large step?
  3. Contact sensor data : One use of which can be to see if the robot is standing or falling.
- Reward designed as a feedback mechanism that tells the agent, if it has chosen apt movements or not. This is done by specifying a reward function as follows:

$$r = min(v_x, v_{max}) - 0.005(v_y^2 + v_z^2) - 0.05y^2 - 0.02||u||^2 + 0.02$$

  Here the r gives the reward for a single time step, cumulative reward would be the summation of all. Here the last term of 0.02 is reward for not falling in a certain time step i.e. for not crashing. The first term is reward for walking forward. The faster the robot walks, the more reward it gets, but to a certain limit. In the second term it penalizes the moving of the agent left or right. In the third term, it penalizes deviation from the center of the track. The fourth term penalizes more force applied to the joints (since more force will results in big difference in the velocities and hence the positions of the joints, which is seen as erratic movement of the robot, which is highly undesirable, whereas, if force applied is less, then the movement will be much more stable for the robot). As per the reward hypothesis, here the reward signal is so designed that if the robot focuses on maximizing the reward, then it would accomplish its goal of being able to walk nicely. Here we need to balance these 4 interactive factors to maximize the cumulative reward, giving the robot a ability to walk.

4.2 Cumulative Reward

at each time step

Till now we got the reward for a single time step, but how do we go about maximizing the cumulative reward from this function? One way of thinking can be that we maximize the reward over all time steps. But this won't be the best way, since it can happen that in order to maximize the reward at a single time step, the agent gets so destabilized that it falls down, ending the episode, resulting in a low cumulative reward. It is clear we need to keep in mind future time steps in mind as well while deciding the action

in order to have higher cumulative reward, and this is true for reinforcement learning in general, since actions have both short and long term consequences, and the agent needs to gain an understanding of the complex effects, its actions have on the environment.

Since the agent(robot) has long term stability in mind, it might decide to walk a little slowly in order to avoid falling in the subsequent time steps, sacrificing some short term reward to improve long term reward.

The reward of the past have been decided already and the agent has no control over them, hence it should decide its actions based on the immediate time step and the future steps and that is exactly what it tries to do. The summation of the immediate return and the future time steps is known as the return or expected return for an arbitrary time step, and the goal is to maximize this return *(Return is also known as expected return because the agent can't really know the future rewards with complete certainty in some situations, for example the pup can't really be expected to know the reward 1000 steps in advance.)*. The return for any arbitrary time step is denoted by G_t. Here,

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ...$$

Since, in some cases, the agent can't be expected to know the rewards of the future with complete certainty, therefore it makes sense to assign weights to the rewards coming earlier to us which are comparatively easier to predict than later rewards. This motivates the concept of **discounted return and discounting**. Here we introduce a coefficient for all terms in the return called the **discount rate** which assigns more weight to earlier rewards as compared to later rewards as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$

Here G_t is the discounted return. Also, the discount rate is not learnt by the agent rather set by us. If we set γ=1, then all rewards will have an equal say, and if it is set to 0, then only immediate reward has a say.

Note: Discounted return is more relevant to continuous tasks.

4.3 Markov decision process

A Reinforcement learning problem can be defined rigorously using MDPs. To understand this we consider a recycling robot problem given in Sutton and Burto. The robot is designed for picking up empty soda cans, equipped with arms to grab the cans and runs on rechargeable battery. Here the robot needs to be taught when to search the room for cans and when to get its battery charged at the docking station. For this problem, we first define the actions:

- Search room for cans
- Stay put : Someone might bring a can to it
- Getting its battery recharged

These actions together are referred to as action space and denoted by A. Next we define the states which are the level of battery of the robot which can be high or low. This is referred to as the state space. State space includes all states except terminal states and denoted as S, whereas state space which includes terminal states is denoted as S^+. Now if the state is high, then the robot can search the room for cans, or wait, but it doesn't really need to charge its battery. Whereas when its battery is less, searching might be risky since it might get discharged and would be left stranded and would need human intervention which is not encouraged.

As explained above, there might be some actions in the action space that won't be possible for some states in the state space and hence they are denoted as $A(s)$

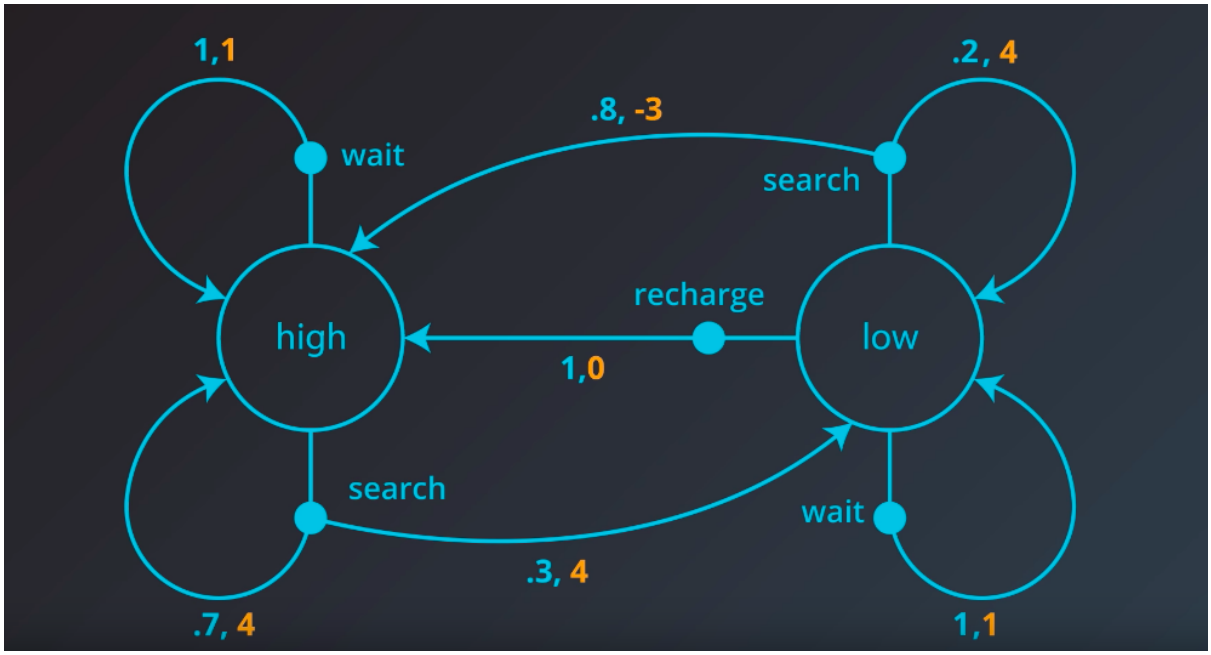


Figure 2: The recycling problem

4.3.1 One step dynamics

We have 2 possible states i.e. low and high based on the charging levels of the robot. We explain the problem of collecting empty cans with reference to these states and the permissible actions associated to them given in the action space, also associating certain probabilities with those actions with corresponding rewards. For example, if the robot is in high state and it chooses to wait, then there is 100% probability that it will remain in high state and will gain 1 reward point, or if it chooses to search when in high state, then there is 70% probability that it will retain to high state with 4 reward points given to. We also think that it does not make sense to recharge when in high state. If it is in low state and it chooses to search then it is with 20% probability that it will return to low state, but has 80% probability of being left stranded and being charged going to high state provided -3 reward points.

Note: When the environment responds to the agent at time step $t+1$, it considers only the state and action at the previous time step(S_t, A_t). It doesn't care what the sequence of steps or evolution procedure has been until now or the reward until now.

We can completely define how the environment decides the state and reward by specifying

$$p(s', r|s, a) = p(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$$

These conditional probabilities are said to specify the one step dynamics of an environment. For example:

$$p(high, 4|high, search) = p(S_{t+1} = high, R_{t+1} = 4|S_t = high, A_t = search) = 0.7$$

That is what is the probability that the next state is high given current state is high, reward for the performed action(which is search) is 4. Here it is 70%.

So any RL problem can be formally defined by defining a finite(both state and action space should be finite) MDP. It is defined by :

1. Set of states(finite)
2. Set of actions(finite)
3. Set of rewards*

4. One step dynamics of the environment*
5. Discount rate : In the previous example, we had a continuing task it is important to have a certain discount rate, else it would have to look into the limitless future. It should be close to 1 though, else it becomes quite short sighted.

*These are not known to the agent.

Now, using this whenever we have a real world, we can formally and fully define a RL problem using the above provided framework. Also, this works for both continuing and episodic task.

5 The RL framework : The solution

After formally defining the RL problem using finite MDP, we need to look for appropriate solutions for the same. The solution can be seen as a series of actions that need to be learned by the agent in pursuit of its goal of maximizing the reward signal. As the apt action changes as the state changes, it can be said that appropriate action response to any state that it can observe can be regarded as a solution to a certain RL problem. This motivates the idea of a policy which determines how an agent chooses an action in response to the current state or in other words gives a mapping between states and actions in order to maximize reward. There can be many such solutions or policies but not all will be optimal and finding the optimal one is the key objective in RL. There are majorly two kinds of policies i.e.

- Deterministic Policy : Simple state to action mapping.

$$\pi : S \rightarrow A$$

- Stochastic Policy : It gives the probability of choosing an action given a certain state.

$$\pi : S * A \rightarrow [0, 1]$$

$$\pi(a/s) = p(A_t = a|S_t = s)$$

Examples:

- Deterministic Policy :
  - $\pi(low) \rightarrow recharge$
  - $\pi(high) \rightarrow search$
- Stochastic Policy :
  - $\pi(recharge/low) = 0.5$
  - $\pi(wait/low) = 0.4$
  - $\pi(search/low) = 0.1$

To understand the selection of the best policy, we take the example of grid world(shown in Figure 3) where the agent needs to reach the target location starting at some location in the grid. The episode ends when it reaches the location. The grid cells are the state space whereas movements to different adjacent cells is what constitutes the action space. The cells can be grass patches or mountains where grass patches are easier to cross than mountains. The reward structure for this problem is as follows :

- -1 for each non terminal transition (barring the ones for mountains)
- -3 for mountain transition
- +5 for terminal transition

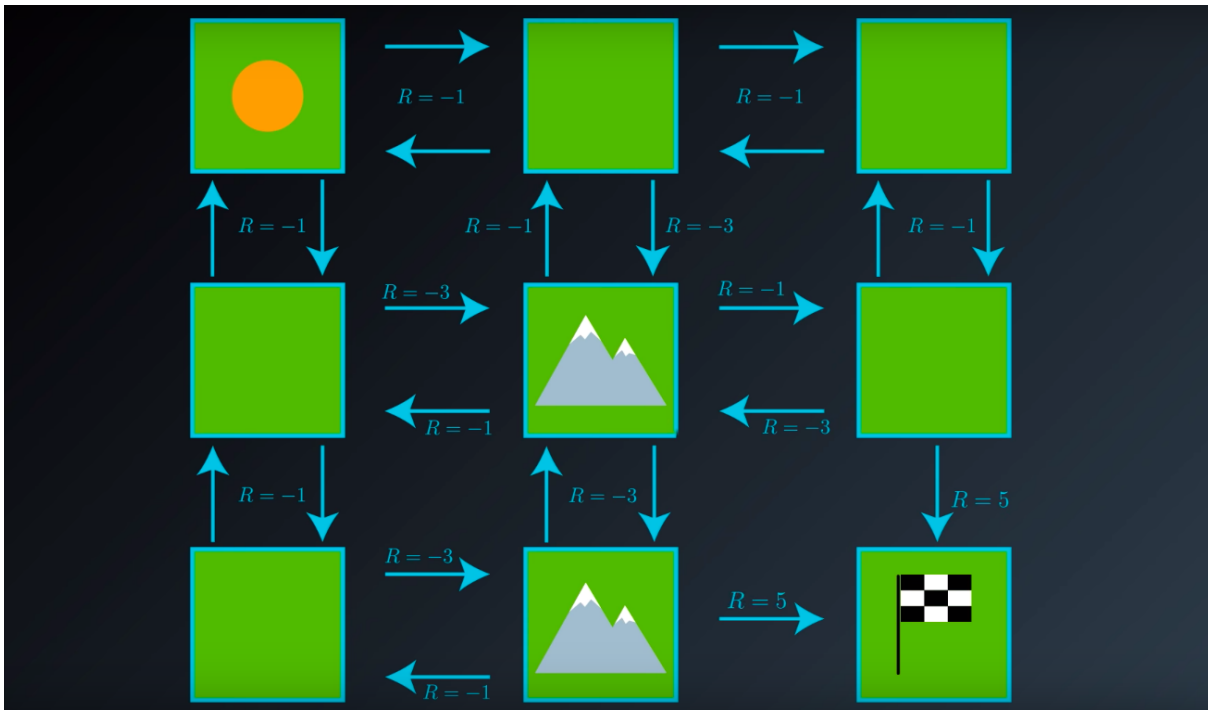


Figure 3: Gridworld

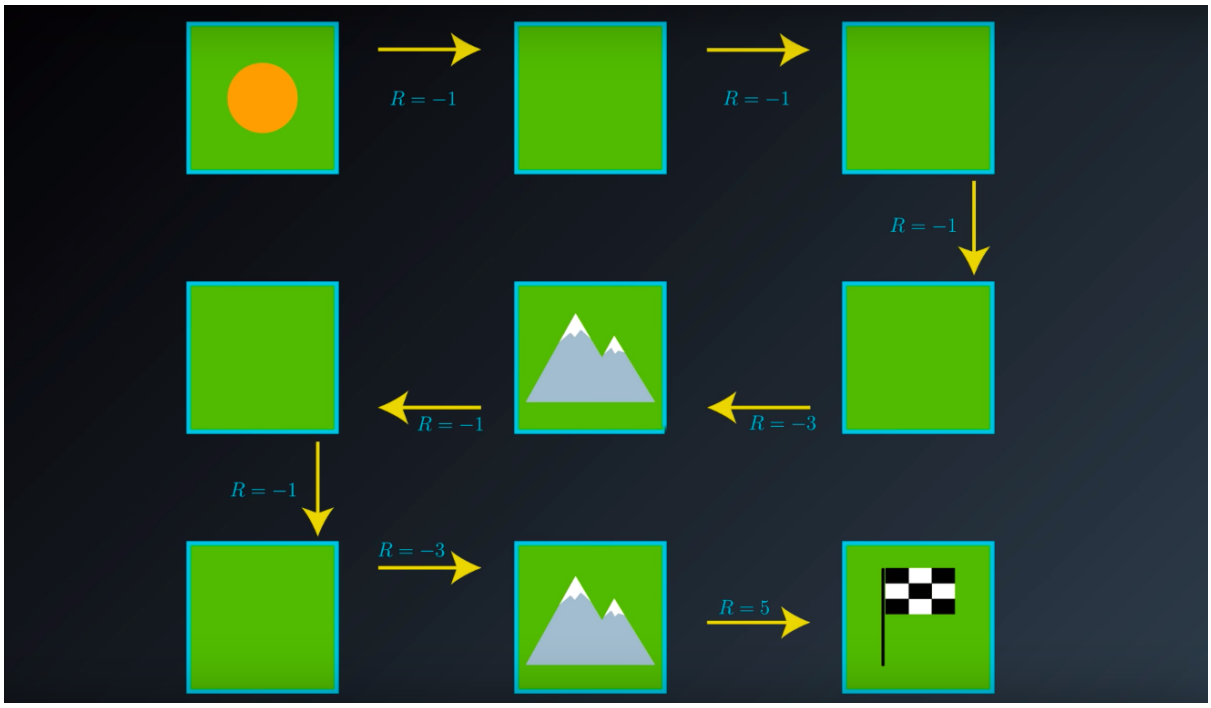


Figure 4: Gridworld with bad policy

5.1 State Value Functions

We start with a presumably bad policy(shown in Figure 4) and understand as to why it is bad, and then try to improve it. To understand why the policy is bad, we start from the initial state and find the cumulative reward if we move in the given sequence (given policy), which is equal to -6. Then we start from the cell right adjacent to it and again do the same, and get the cumulative reward equal to -5. The same can be done for each state. For the terminal state the reward would be 0 since if the agent starts there, then the episode ends straight away. Accordingly we have a number associated with each state, which is the output of a function of it, called the state value function. Formally, for each state it yields the result that is to follow, if the agent starts at that location and follows the policy for all time steps. Here the γ is taken to be 1, or we can say there is no discounting. The mathematical notation for the same is as follows:

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

For each state S, it yields the expected discounted return if the agent starts in state S and then uses the policy to choose its actions for all time steps

5.2 Bellman equations

The state value functions have a recursive property associated to it i.e. to find the value for any state we can do that by adding the immediate return and the discounted return of the next state (which is also calculated there forth, hence the recursion for which the terminal state is the base case with a value of 0). Also, here we take γ to be 1, which is not generally true for MDPs, so we need a framework which takes discounting into account and that is why we have taken discounted return of the next state. This can be explained further using Bellman equation (which gives the expected return for a state since we can't assume rewards for future states as well as future states with complete certainty) i.e.

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(s+1)|S_t = s]$$

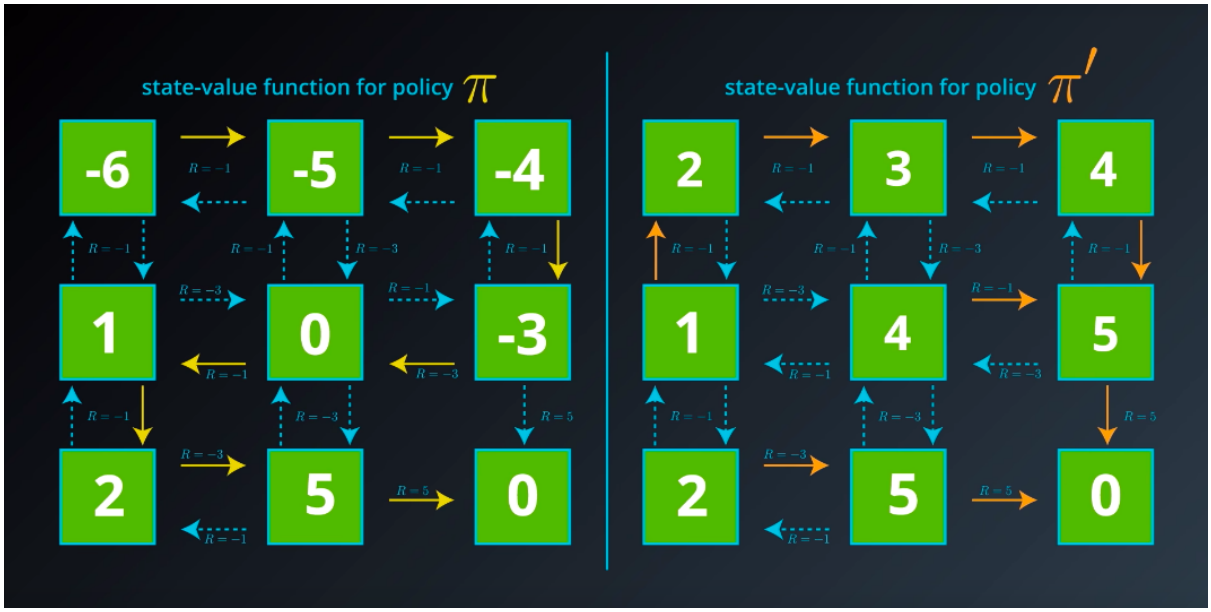


Figure 5: Policies compared

5.2.1 Calculating the reward

If the policy is deterministic, the state value function can be seen as

$$v_\pi(s) = \sum_{s' \in S^+, r \in R} p(s', r|s, \pi(s))(r + \gamma v_\pi(s'))$$

For a stochastic policy, the state value function would be

$$v_\pi(s) = \sum_{s' \in S^+, r \in R, a \in A(s)} \pi(a|s) p(s', r|s, \pi(s))(r + \gamma v_\pi(s'))$$

5.3 Comparing policies

Given two policies π_1 and π_2, we can say that policy π_1 is better than the other if $v_{\pi_1}(s) \geq v_{\pi_2}(s)$ for all $s \in S$. But it is possible that two policies might not be comparable in some situations. But it is also important to remember than an optimal policy will always exist even if it isn't unique. This strategy is the best strategy to accomplish some goal. Denoted by v_*.

5.4 Action value function

For finding optimal policies, it is important to calculate the action value function for the states. It is defined as the value of taking action a in state s under policy π i.e. *(If I take this action, then the reward would be this much.)*

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

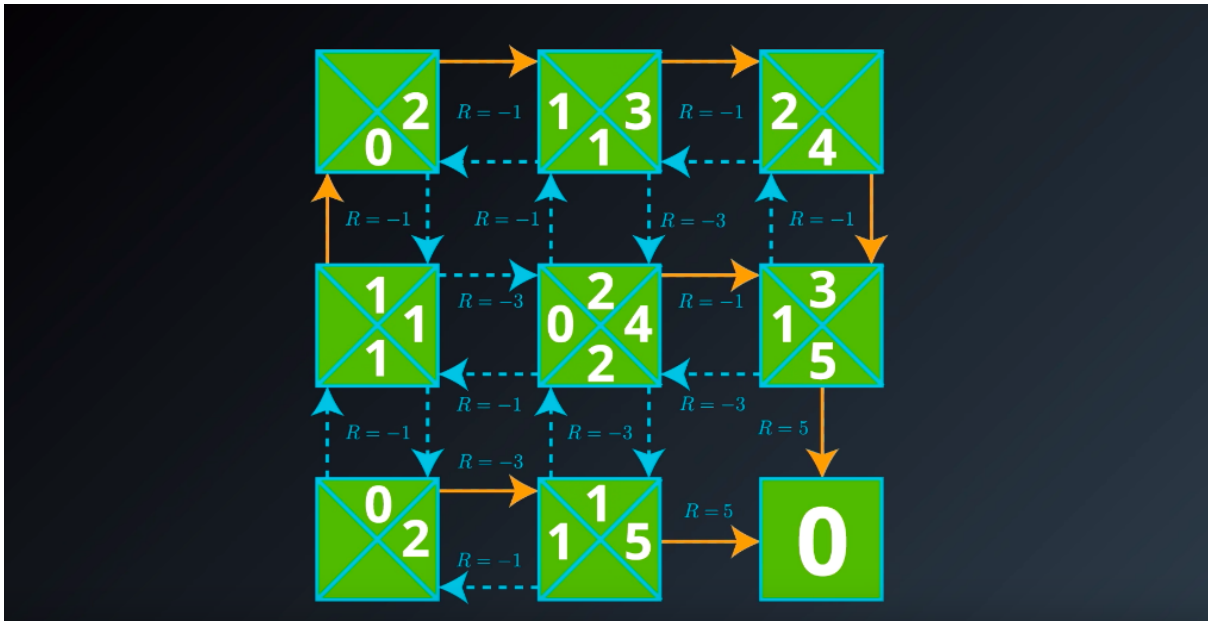


Figure 6: Action values

For each state s and action a, it yields the expected return if the agent starts in state s, then chooses action 'a' and then uses the policy to choose its actions for all time steps.

5.5 From optimal action values to optimal policy

By choosing the best action per state that yields the highest action value, we can move from q_* to π_*

6 Monte Carlo Methods

6.1 GridWorld Example

In this section, we undertake another grid based problem and using that we will go about understanding future concepts which will be applied to different complex problems ahead.

Here we have 4 cells, (there is a wall between 1 & 4) and we need to move from state 1 to state 4($|Statespace| = 4$), and actions can be any between up, down, left or right ($|Actionspace| = 4$). We define the discount rate here as 1. The rewards are -1 for all non terminal state resulting actions and +10 for terminal state resulting actions.

The one step dynamics is : If we are in state 1 and we decide to take action 'up', then there is 70% probability that the agent actually moves in that direction but there is also 30% probability (due to some factors which can be both agent or environment induced) that it won't be able to move in the selected direction (equal for each different direction other than the one selected). This is for all actions in the action space.

Remember this: In current state, what is the probability of certain action, with the probabilities of the state resulting from the action and also the reward associated to it keeping in mind the discount rate to find the discounted return.

Initially the agent knows nothing about the conditions and the environment; and the best way for the agent is to proceed with a random action and proceed ahead. Here the probability of each action being chosen in a state is the same i.e. it follows equi-probable random policy.

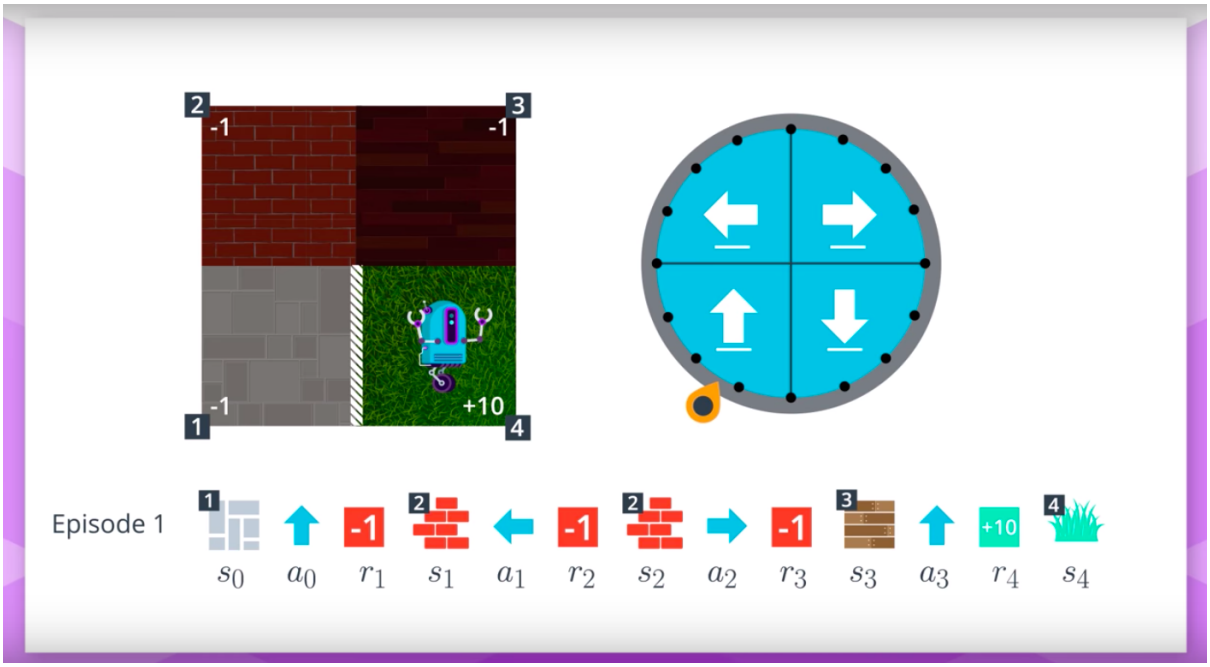


Figure 7: Grid world sample episode

For example in the first episode as shown in the figure 7(a) the agent is initially at state 0 and after choosing randomly we get action up (remember that the problem follows a stochastic pattern meaning

that the post state post an action is probabilistic, but the state should be reachable logically from that state using the action space), post the action the agent gets a reward of -1 and the next state is 2. Then after choosing randomly we get action left, then reward -1 and post state remains 1. Then action chosen is right, reward comes out to be -1 and next state is 3. Next action is up, but it ends up in the terminal state and gets a reward of +10. Similarly in episode 2 as shown in figure 7(b).

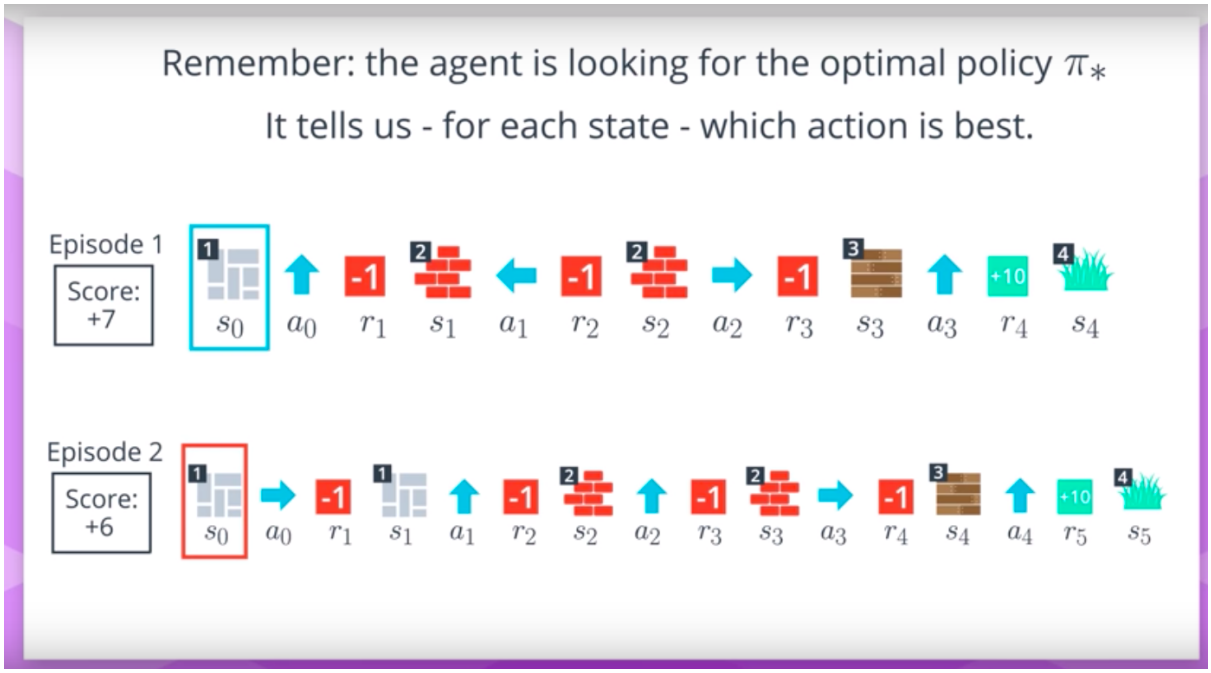


Figure 8: Grid world sample episode

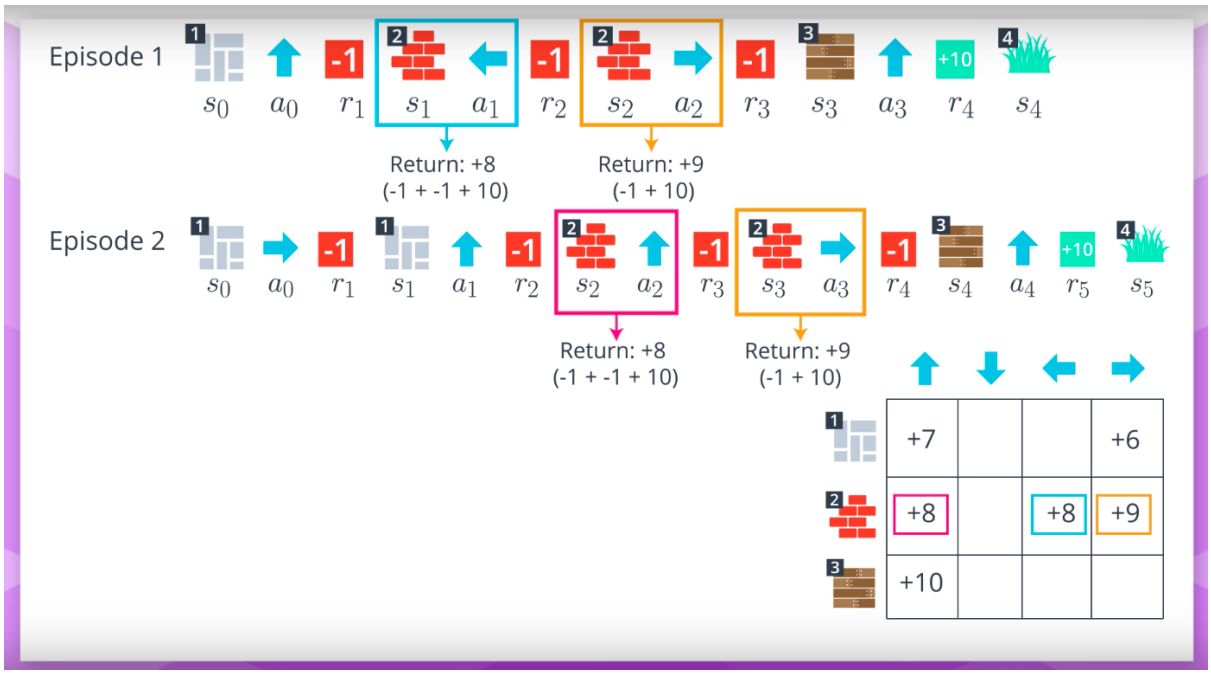
Now the question is how is this information valuable, or how can we develop an optimal policy from this information or how can we improve from the present policy. So here we note that we start from state 1 and here we choose the action that has yielded the highest reward in the past experience. Like in the short experience above, since up action yields higher reward, then it would be the preferred action.This is the basic idea behind the method of extracting the optimal policy by interacting with the environment.

But to truly understand the environment and to be able to devise an optimal policy, we need to conduct more episodes. This is because the agent needs to try out all permutations and combinations with the environment to be able to decide what action is better. Since the environment's dynamics are stochastic and that adds another layer of randomness which needs to be looked at properly. Performing more episodes brings under disposal so much information, that the agent gets enough knowledge to decide which action is statistically more rewarding.

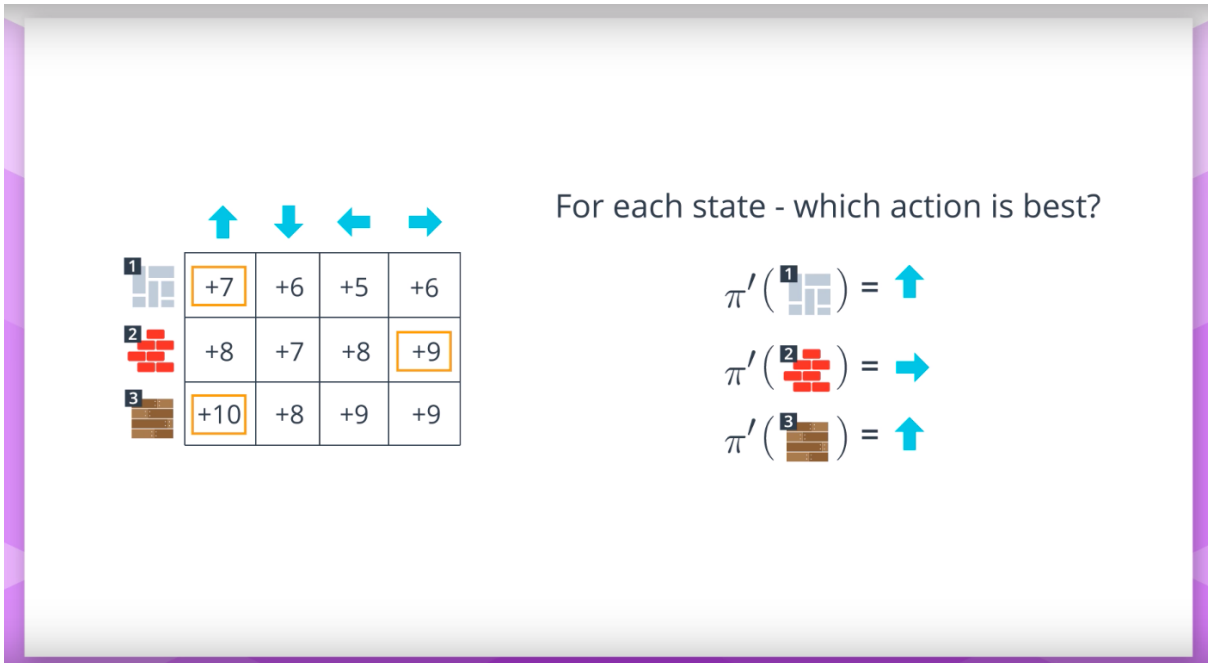
This is indeed the basic idea behind the Monte Carlo approach i.e. collect many episodes using a certain policy (equi-probable random policy here) and then for each state figure out which action yields the highest cumulative reward.

Now the issue is of how do we consolidate the data to choose the apt actions for a policy from the past experience. So here we maintain a table states vs actions, (as shown in figure 9) where the values are the rewards for the state if the particular action is taken. We take the summation of rewards upon the number of times the action has been taken for that state. Based on 100s or 1000s of episodes, a lot of information will be collected and we will be able to make a better guess of what is the better action to be taken in a certain state.

Once a lot of data is collected, we can choose the action with maximum reward for each state and that yields a better or equivalent policy than the one employed currently. As seen in Figure 10, this is not necessarily the optimal policy but is a small step in finding it, as we have a relatively better policy in hand.



(a) Grid world sample episode



(b) Grid world sample episode

Figure 9: Q table construction

Now this table allows us to estimate the action value function for the equi-probable random policy i.e. it allows us to estimate the expected return if the agent starts in a certain state and picks up an action and follows the policy for the future states. This table is known as the Q table and obviously the more data we collect the better our table would be, and that would allow us to estimate a better policy in a more efficient manner.

This problem of finding or estimating the value function given a policy is called the prediction problem. The Monte-Carlo approaches applied here are called Monte Carlo prediction methods. Now there are 2 versions to this based on how we treat the case where we have multiple times a state and action pair is encountered in an episode. The first option is of taking the average of all visits and the next is of taking into consideration only the first visit. Here visit is defined as an occurrence of a state action pair in an episode.