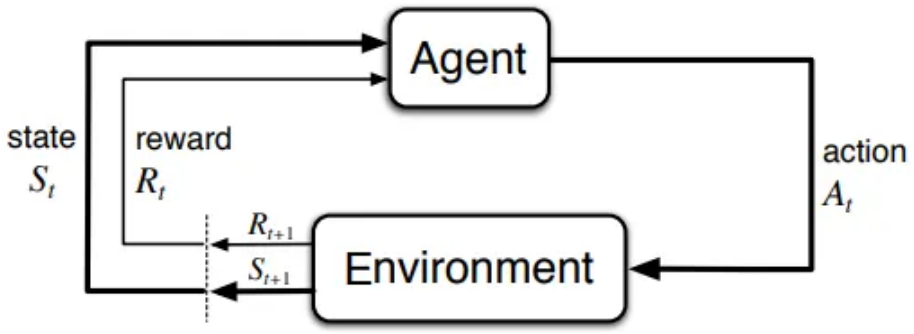
#INSIDERL

Reinforcement Learning : Markov-Decision Process (Part 1)

blackburn · Follow
Published in Towards Data Science
14 min read · Jul 18, 2019

In a typical Reinforcement Learning (RL) problem, there is a learner and a decision maker called ***agent*** *and the surrounding with which it interacts is called* ***environment****. The environment, in return, provides* ***rewards*** *and a* ***new state*** *based on the* ***actions*** *of the agent. So, in reinforcement learning, we do not teach an agent how it should do something but presents it with rewards whether positive or negative based on its actions.* ***So our root question for this blog is how we formulate any problem in RL mathematically****. This is where the* ***Markov Decision Process****(MDP) comes in.*



Typical Reinforcement Learning cycle

Before we answer our root question i.e. How we formulate RL problems mathematically (using MDP), we need to develop our intuition about :

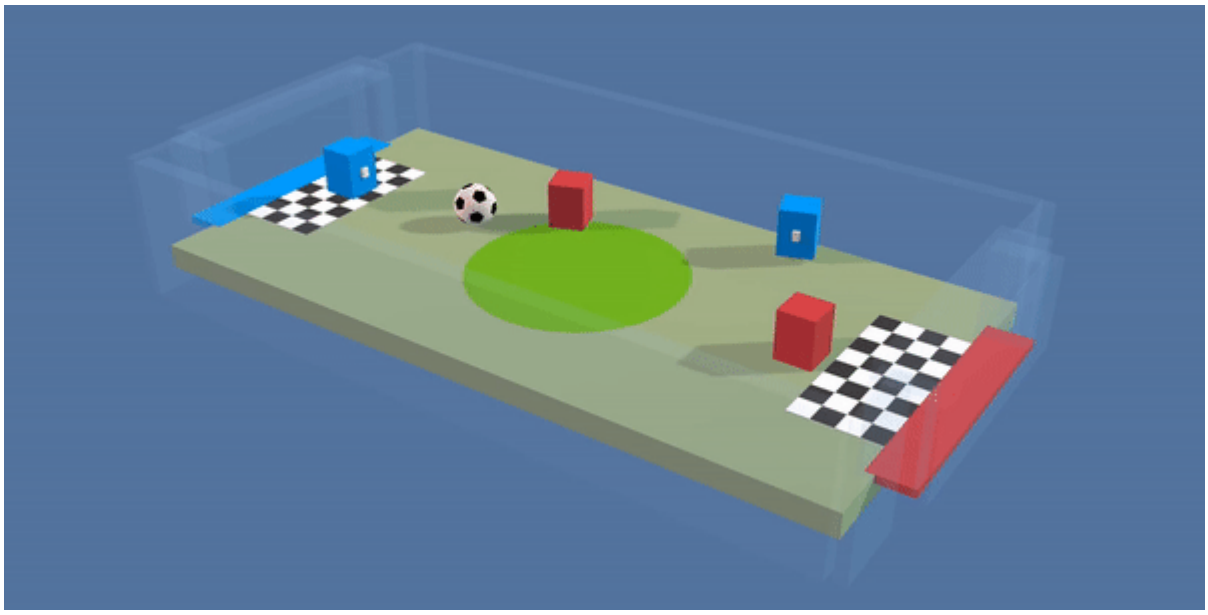
- The Agent-Environment relationship
- Markov Property
- Markov Process and Markov chains
- Markov Reward Process (MRP)
- Bellman Equation
- Markov Reward Process

Grab your coffee and don't stop until you are proud! 🧐

The Agent-Environment Relationship

First let's look at some formal definitions :

> ***Agent*** *: Software programs that make intelligent decisions and they are the learners in RL. These agents interact with the environment by actions and receive rewards based on there actions.*
>
> ***Environment*** *:It is the demonstration of the problem to be solved.Now, we can have a real-world environment or a simulated environment with which our agent will interact.*



Demonstrating an environment with which agents are interacting.

> ***State*** *: This is the position of the agents at a specific time-step in the environment.So,whenever an agent performs a action the environment gives the agent reward and a new state where the agent reached by performing the action.*

Anything that the agent **cannot change arbitrarily** is considered to be part of the environment. In simple terms, actions can be **any decision we want the agent to learn** and **state can be anything which can be useful in choosing actions.** We do not assume that everything in the environment is unknown to the agent, for example, reward calculation is considered to be the part of the environment even though the agent knows a bit on how it's reward is calculated as a function of its actions and states in which they are taken. This is because **rewards cannot be arbitrarily** changed by the agent. Sometimes, the agent might be fully aware of its environment but still finds it difficult to maximize the reward as like we might know how to play Rubik's cube but still cannot solve it. So, we can safely say that the agent-environment relationship represents the limit of the agent control and not it's knowledge.

The Markov Property

> ***Transition*** *: Moving from one state to another is called Transition.*
>
> ***Transition Probability****: The probability that the agent will move from one state to another is called transition probability.*

The ***Markov Property*** *state that :*

"Future is Independent of the past given the present"

Mathematically we can express this statement as :

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots, S_t]$$

Markov Property

S[t] denotes the current state of the agent and s[t+1] denotes the next state. What this equation means is that the transition from state S[t] to S[t+1] is entirely independent of the past. So, the **RHS** of the Equation means the same as **LHS** if the system has a Markov Property. Intuitively meaning that our current state already captures the information of the past states.

State Transition Probability :

As we now know about transition probability we can define state Transition Probability as follows :

For Markov State from S[t] to S[t+1] i.e. any other successor state , the state transition probability is given by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State Transition Probability

We can formulate the State Transition probability into a State Transition probability matrix by :

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ & \cdots & \cdots & \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

State Transition Probability Matrix

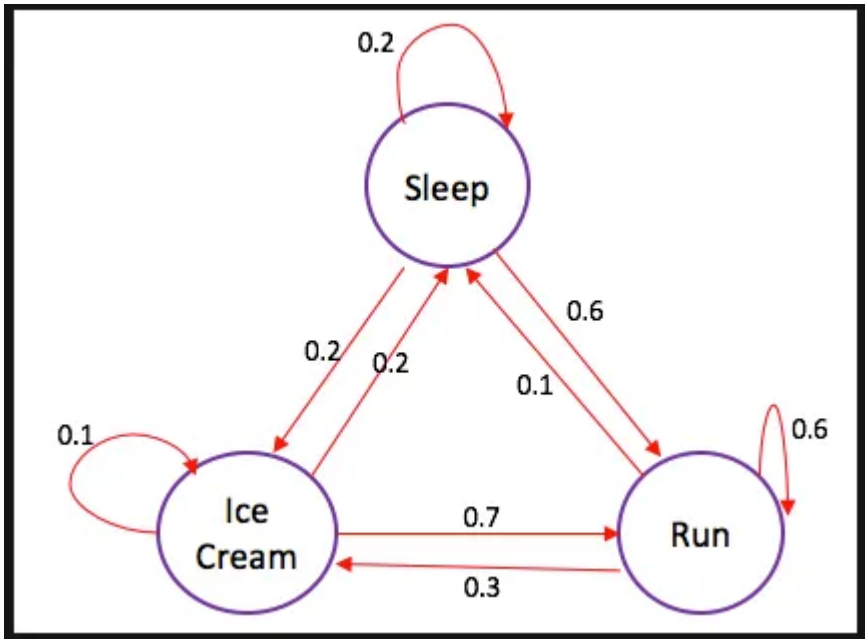
Each row in the matrix represents the probability from moving from our original or starting state to any successor state.Sum of each row is equal to 1.

Markov Process or Markov Chains

Markov Process is the memory less ***random process*** *i.e. a sequence of a random state S[1],S[2],….S[n] with a Markov Property*.So, it's basically a sequence of states with the Markov Property.It can be defined using a set of states(S) and transition probability matrix (P).The dynamics of the environment can be fully defined using the States(S) and Transition Probability matrix(P).

But what ***random process*** means ?

To answer this question let's look at a example:



Markov chain

The edges of the tree denote **transition probability**. From this chain let's take some sample. Now, suppose that we were sleeping and the according to the probability distribution there is a **0.6** chance that we will **Run** and **0.2** chance we **sleep more** and again **0.2** that we will eat **ice-cream**. Similarly, we can think of other sequences that we can sample from this chain.

Some samples from the chain :

- Sleep — Run — Ice-cream — Sleep
- Sleep — Ice-cream — Ice-cream — Run

In the above two sequences what we see is we get random set of States(S) (i.e. Sleep,Ice-cream,Sleep) every time we run the chain.Hope, it's now clear why Markov process is called random set of sequences.

Before going to Markov Reward process let's look at some important concepts that will help us in understand MRPs.

Reward and Returns

> ***Rewards*** *are the* ***numerical values*** *that the agent receives on performing some action at some* ***state(s)*** *in the environment. The numerical value can be positive or negative based on the actions of the agent.*
>
> *In* ***Reinforcement learning****, we care about* ***maximizing*** *the cumulative reward (all the rewards agent receives from the environment) instead of, the reward agent receives from the current state(also called immediate reward). This* ***total sum of reward*** *the agent receives from the environment is called* ***returns****.*

We can define Returns as :

$$G_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

Returns (Total rewards from the environment)

r[t+1] is the reward received by the agent at time step t[0] while performing an action(a) to move from one state to another. Similarly, r[t+2] is the reward received by the agent at time step t[1] by performing an action to move to another state. And, r[T] is the reward received by the agent by at the final time step by performing an action to move to another state.

Episodic and Continuous Tasks

> ***Episodic Tasks****: These are the tasks that have a* ***terminal state*** *(end state).We can say they have finite states. For example, in racing games, we start the game (start the race) and play it*

> *until the game is over (race ends!). This is called an episode. Once we restart the game it will start from an initial state and hence, every* ***episode*** *is independent.*
>
> ***Continuous Tasks*** *: These are the tasks that have no ends i.e. they* ***don't have any terminal state****.These types of tasks will* ***never end****.For example, Learning how to code!*

Now, it's easy to calculate the returns from the episodic tasks as they will eventually end but what about continuous tasks, as it will go on and on forever. The returns from sum up to infinity! So, how we define returns for continuous tasks?

This is where we need **Discount factor(ϒ).**

> ***Discount Factor (ϒ)****: It determines how much* ***importance*** *is to be given to the* ***immediate reward and future rewards****. This basically helps us to avoid* ***infinity*** *as a reward in continuous tasks. It has a value between 0 and 1. A value of* ***0*** *means that more importance is given to the* ***immediate reward*** *and a value of* ***1*** *means that more importance is given to* ***future rewards. In practice****, a discount factor of 0 will never learn as it only considers immediate reward and a discount factor of 1 will go on for future rewards which may lead to infinity. Therefore,* ***the optimal value for the discount factor lies between 0.2 to 0.8****.*

So, we can define returns using discount factor as follows :**(Let's say this is equation 1 ,as we are going to use this equation in later for deriving Bellman Equation)**

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

Returns using discount factor

Let's understand it with an example,suppose you live at a place where you face water scarcity so if someone comes to you and say that he will give you 100 liters of water! (assume please!) for the next 15 hours as a function of some parameter (ϒ).Let's look at two possibilities : (Let's say this is equation 1 ,as we are going to use this equation in later for deriving Bellman Equation)

One with discount factor (ϒ) **0.8 :**

$$G_t = 100 + 80 + 64 \ldots$$

Discount Factor (0.8)

This means that we should wait till 15th hour because the decrease is not very significant , so it's still worth to go till the end.This means that we are also interested in future rewards.So, if the discount factor is close to 1 then we will make a effort to go to end as the reward are of significant importance.

Second, with discount factor (γ) **0.2 :**

$$G_t = 100 + 20 + 4 \ldots$$

Discount Factor (0.2)

This means that we are more interested in early rewards as the rewards are getting significantly low at hour.So, we might not want to wait till the end (till 15th hour) as it will be worthless.So, if the discount factor is close to zero then immediate rewards are more important that the future.

> *So which value of discount factor to use ?*

It depends on the task that we want to train an agent for. Suppose, in a chess game, the goal is to defeat the opponent's king. If we give importance to the immediate rewards like a reward on pawn defeat any opponent player then the agent will learn to perform these sub-goals no matter if his players are also defeated. So, in this task future rewards are more important. In some, we might prefer to use immediate rewards like the water example we saw earlier.

Markov Reward Process

Till now we have seen how Markov chain defined the dynamics of a environment using set of states(S) and Transition Probability Matrix(P).But, we know that Reinforcement

Learning is all about goal to maximize the reward.So, let's ***add reward*** to our Markov Chain.This gives us **Markov Reward Process.**

> ***Markov Reward Process*** *: As the name suggests, MDPs are the Markov chains with values judgement.Basically, we get a value from every state our agent is in.*

Mathematically, we define Markov Reward Process as :

$$R_s = E[R_{t+1} \mid S_t]$$

Markov Reward Process

What this equation means is how much reward (Rs) we get from a particular state S[t]. This tells us the immediate reward from that particular state our agent is in. As we will see in the next story how we maximize these rewards from each state our agent is in. In simple terms, maximizing the cumulative reward we get from each state.

We define MRP as (S,P, R,γ) , where :

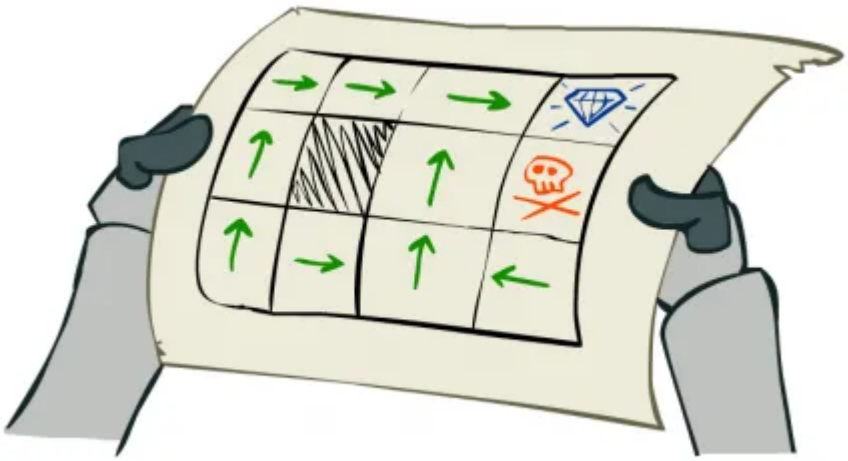
- S is a set of states,
- P is the Transition Probability Matrix,
- R is the Reward function, we saw earlier,
- γ is the discount factor

Markov Decision Process

Now, let's develop our intuition for Bellman Equation and Markov Decision Process.

Policy Function and Value Function

Value Function determines how good it is for the agent to be in a particular state. Of course, to determine how good it will be to be in a particular state it must depend on some actions that it will take. This is where policy comes in. A policy defines what actions to perform in a particular state s.



A policy is a simple function, that defines a **probability distribution** over Actions (a∈ A) for each state (s ∈ S). If an agent at time t follows a policy π then **π(a|s)** is the probability that the agent with taking action (a) at a particular time step (t).In Reinforcement Learning the experience of the agent determines the change in policy. Mathematically, a policy is defined as follows :

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

Policy Function

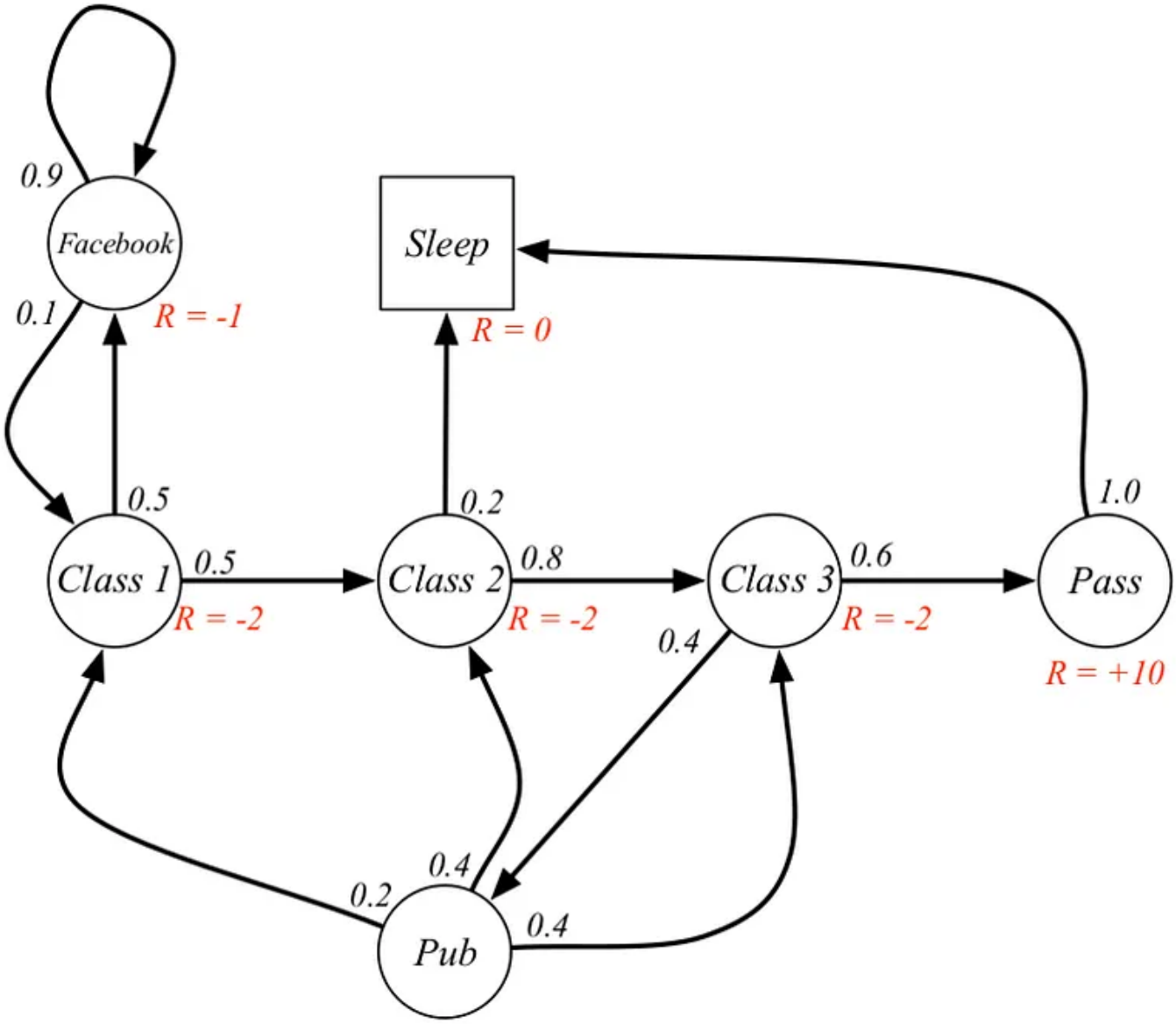
Now, **how do we find a value of a state.** The value of state s, when the agent is following a policy π which is denoted by vπ(s) is the expected return starting from s and following a policy π for the next states until we reach the terminal state. We can formulate this as :(**This function is also called State-value Function**)

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

Value Function

This equation gives us the **expected returns** starting from the **state(s)** and going to successor states thereafter, with the policy π. One thing to note is the returns we get is stochastic whereas the value of a state is **not** stochastic. It is the expectation of returns

from start state s and thereafter, to any other state. And also note that the value of the terminal state (if there is any) is zero. Let's look at an example :



Example

Suppose our start state is Class 2, and we move to Class 3 then Pass then Sleep.In short, Class 2 > Class 3 > Pass > Sleep.

Our expected return is with a discount factor of 0.5:

$$G_t = -2 * -2 * 0.5 + 10 * 0.25 + 0$$

$$= -0.5$$

Calculating the Value of Class 2

Note:*It's -2 + (-2 * 0.5) + 10 * 0.25 + 0 instead of -2 * -2 * 0.5 + 10 * 0.25 + 0.***Then the value of Class 2 is -0.5 .**

Bellman Equation for Value Function

Bellman Equation helps us to find **optimal policies** and **value functions.** We know that our policy changes with experience so we will have different value functions according to different policies. The ***optimal value function is one that gives maximum value compared to all other value functions.***

Bellman Equation states that value function can be **decomposed** into two parts:

- Immediate Reward, R[t+1]
- Discounted value of successor states,

Mathematically, we can define Bellman Equation as :

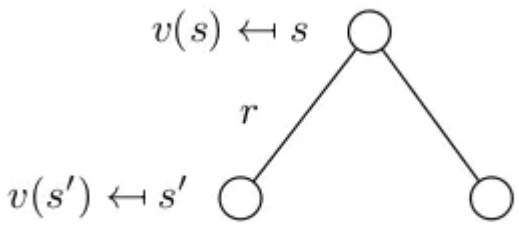
$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

Bellman Equation for Value Function

Let's understand what this equation says with a help of an example :

Suppose, there is a robot in some state (s) and then he moves from this state to some other state (s'). Now, the question is **how good it was for the robot to be in the state**(s). Using the Bellman equation, we can that it is the expectation of **reward** it got on leaving the state(s) plus the **value of the state** (s') he moved to.

Let's look at another example :



Backup Diagram

We want to know the **value** of state s. The value of state(s) is the reward we got upon leaving that state, plus the discounted value of the state we landed upon multiplied by the transition probability that we will move into it.

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

Value Calculation

The above equation can be expressed in matrix form as follows :

$$\begin{aligned} v &= \mathcal{R} + \gamma \mathcal{P} v \\ (I - \gamma \mathcal{P})\, v &= \mathcal{R} \\ v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R} \end{aligned}$$

Bellman Linear Equation

Where v is the value of state we were in, which is equal to **the immediate reward plus the discounted value of the next state multiplied by the probability of moving into that state.**

The running time complexity for this computation is $\mathbf{O(n^3)}$**.** Therefore, this is clearly not a practical solution for **solving larger MRPs** (same for **MDPs**).In later Blogs, we will

look at more efficient methods like **Dynamic Programming** (Value iteration and Policy iteration), **Monte-Claro methods,** and **TD-Learning**.

We are going to talk about the Bellman Equation in much more detail in the next story.

> ***What is Markov Decision Process ?***
>
> ***Markov Decision Process** : It is Markov Reward Process with a decisions.Everything is same like MRP but now we have actual agency that makes decisions or take actions.*

It is a tuple of (***S***, ***A***, ***P***, ***R***, γ) where:

- S is a set of states,
- A is the set of actions agent can choose to take,
- P is the transition Probability Matrix,
- R is the Reward accumulated by the actions of the agent,
- γ is the discount factor.

P and R will have slight change w.r.t actions as follows :

> ***Transition Probability Matrix***

$$\mathcal{P}^{a}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

Transition Probability Matrix w.r.t action

> ***Reward Function***

$$\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

Reward Function w.r.t action

Now, our reward function is dependent on the action.

Till now we have talked about getting a reward (r) when our agent goes through a set of states (s) following a policy π. Actually, in Markov Decision Process(MDP) the policy is the mechanism to take decisions. So now we have a mechanism that will choose to take an action.

Policies in an MDP depend on the current state. They do not depend on history. That's the Markov Property. So, the current state we are in characterizes history.

We have already seen how good it is for the agent to be in a particular state(State-value function). Now, let's see how good it is to take a particular action following a policy π from state s (Action-Value Function).

State-action value function or Q-Function

This function specifies **how good** it is for the agent to take action (a) in a state (s) with a **policy π.**

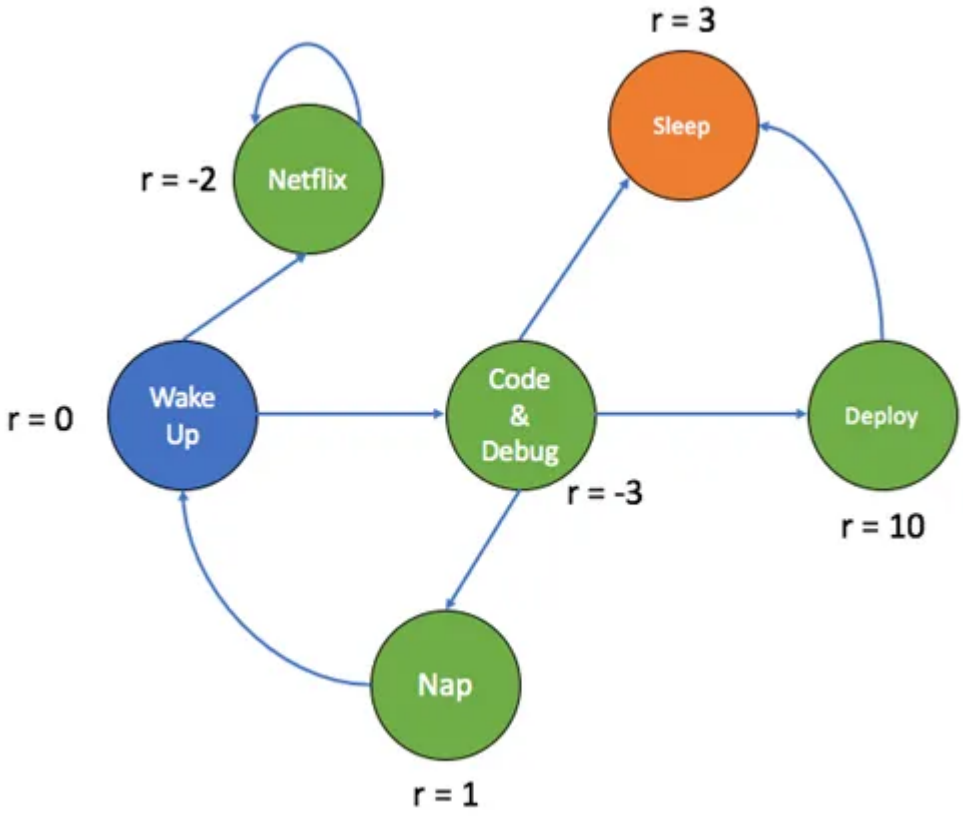
Mathematically, we can define the **State-action** value function as :

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t=s, A_t=a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \,\middle|\, S_t=s, A_t=a\right].$$

State-action value function

Basically, it tells us the value of performing a certain action(a) in a state(s) with a policy π.

Let's look at an example of the Markov Decision Process :



Example of MDP

Now, we can see that there are no more probabilities. In fact, now our agent has choices to make like after waking up, we can choose to watch Netflix or code and debug. Of course, the actions of the agent are defined w.r.t some policy π and will get the reward accordingly.

. . .

Fantastic!

Congratulations on sticking till the end! 👍

Till now we have talked about building blocks of MDP, in the upcoming stories, we will talk about and **Bellman Expectation Equation, More on optimal Policy and optimal value function,** and **Efficient Value Finding method i.e. Dynamic Programming** (value iteration and policy iteration algorithms) and programming it in **Python.**