Reinforcement Learning-2

Photo by Franck V. on Unsplash

#REINFORCEDSERIES

Understanding Reinforcement Learning- 2

What are the Markov Process, Markov Reward Process, and Markov Decision Process?

This is the second article on Reinforcement Learning. In the previous article, we looked at a brief discussion on Markov Processes, and the basic terms and terminologies associated with Reinforcement learning. In this article, we will talk about the World or Environment Representation, Markov Processes, and its variants.

Models used to represent the World

We have previously discussed, in the case of model-based reinforcement learning agent, we try to model the world and interact with it, instead of interacting with the actual world or

environment where the agent operates. So, to model the environment, we use three main representational processes, namely:

1. Markov Process
2. Markov Reward Process
3. Markov Decision Process

As discussed previously, to represent an environment or a world we mainly need a probability transition matrix and a reward matrix. The processes also have the above features making them suitable to represent reinforcement learning environments.

Markov Process

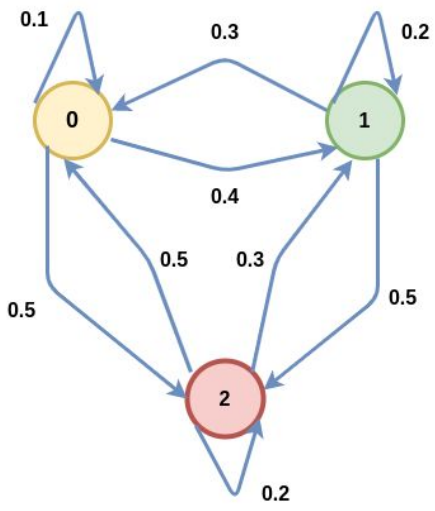
A Markov Process is a memory-less random process which is given by a sequence of random states following the Markov assumption Property.

Markov Process is represented by a two-tuple. It is given by (S, P).

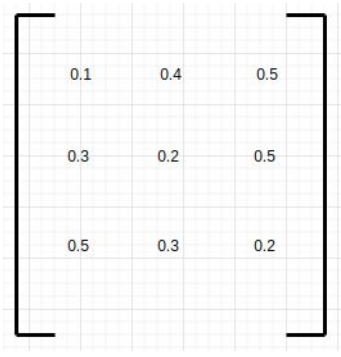
1. S is the finite state of sets
2. P is the transition dynamics or the probability transition matrix.

$$P -> Pr(S(t+1)) = s' \mid S(t) = s)$$

Markov process is a random process that has no actions and no rewards. For clarity, we can imagine a gas trapped in a container. It needs no outside action and rewards to change the state.



The image may be used to represent a Markov process. It will have a representational probability matrix, given by,



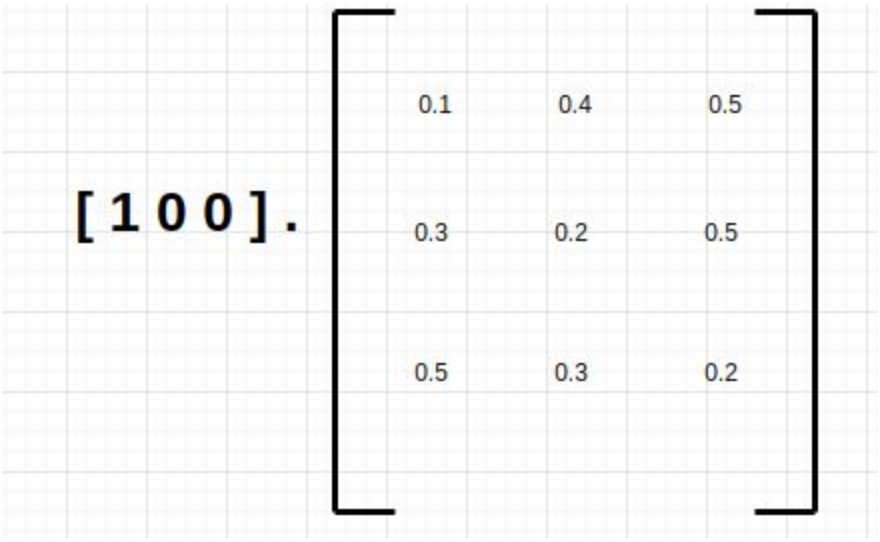
And the set of states will be given by [0,1,2]

If our initial state is 0, the state is represented as [1 0 0] in the vector form. The state shown by [1 0 0] is termed as Classical State because here the agent is determined to be present in one distinct state, here S0.

The probability distribution for the next state Pr(S') is given by:

$$Pr(S') = S.P$$

Where S' is the next state, S is the current state and P is the transition matrix.



The obtained probability distribution is [0.1 0.4 0.5]. It is evident that the distribution obtained is the probability of transitions of State S0. The obtained probability distribution is called the quantum state. The quantum state is a transition phase between two classical states.

Next, we sample from the obtained probability distribution to get the next classical state. The probability distribution states that the chance of landing on state 0 is 10%, the chance for landing on state 1 is 40% and for 2 its 50%. So, we have to randomize the sample generator accordingly. Now, if we obtain state 1 as the randomly sampled state the next classical state becomes state 1 represented as [0 1 0], This way random sampling is done and the states keep shifting in case of Markov processes.

This is all about the Markov processes, Next, we move to Markov Reward Process.

Markov Reward Process

Markov Reward Process consists of Basic Markov Chain or Process associated with Rewards. It is defined by a four tuple (S, P, R, gamma)

1. S represents the final set of states
2. P represents the transition dynamics
3. R represents the rewards gained on reaching each state
4. Discount Factor gamma.

Markov Reward Process has no action involved. So, in Markov Reward Process also the agent changes states and shifts without any action from the agent.

Markov Reward Process is associated with a **value function** for each step. We have discussed this previously. The value function is the probabilistic sum of discounted future rewards that may be obtained from that particular state.

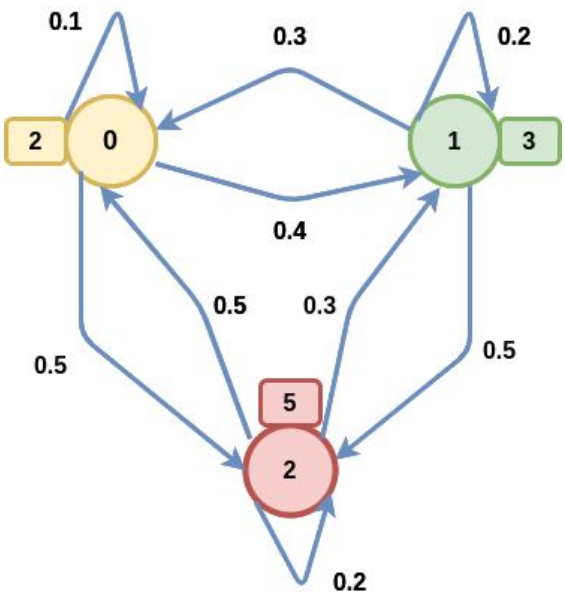
Return at time t (G^t) is the discounted sum of future rewards from time t to the final step of that episode. The "episode" can be explained like a clock cycle. 1 episode is equivalent to the number of timesteps taken by the agent to reach a final goal state or hit the maximum number of timesteps allowed by the user.

$$G(t) = R(t) + gamma * R(t+1) + gamma^2 * R(t+2) + ... + gamma^n R(t+n)$$

The value function of state s (V(s)) is the expected or probabilistic return from a starting state s.

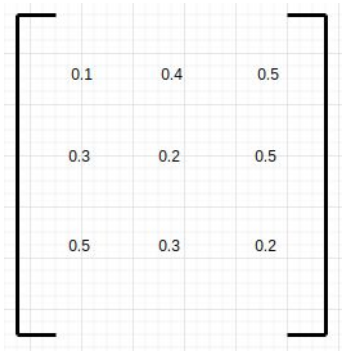
$$V(s) = E[G(t) \mid S(t) = s] = E\,[\,R(t) + gamma * R(t+1) + gamma^2 * R(t+2) + ... \mid S(t) = s\,]\,|$$

Sample Representation:



The given environment expressed as Markov Reward Process is given by:

P or transition matrix:



S of the set of states:
[0, 1,2]

R or reward matrix:

[2, 3, 5]

Gamma or discount Factor: 0.9

Our target is to calculate the value function for every state s, in an MRP to determine how good the states of the MRP.

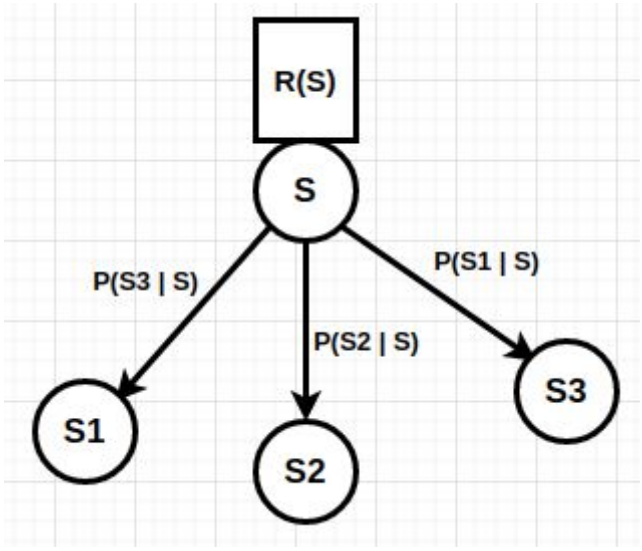
Calculation of the value function.

The value function of an MRP is (n x 1) matrix, where n is the number of states because it has the individual values for every state existing in the MRP.

The value function for the state ‘s’ satisfies the equation:

$$V(s) = R(s) + gamma.\Sigma P(s' \mid s).V(s')$$

This is called the **Bellman Operation**. We will see this in the future. The first part of the equation gives the current reward at state s. The second part of the equation accounts for discounted future rewards. The state s’ belongs to set of all states S for an MRP.



For the figure, the equation formulates to:

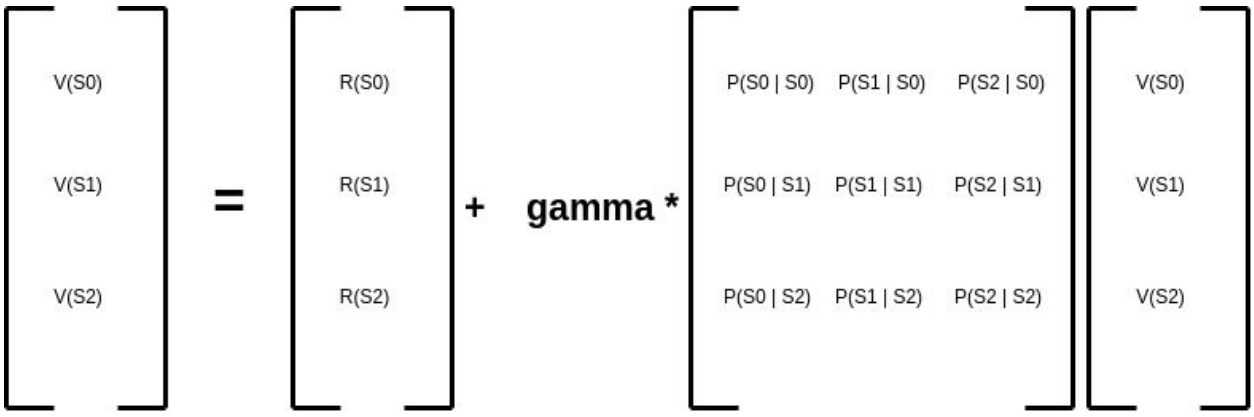
$$V(S) = R(S) + gamma * \{P(S1|S).V(S1) + P(S2 \mid S).V(S2) + P(S3 \mid S).V(S3)\}$$

There are two main methods to calculate the value function:

1. Calculating using Analytic method
2. Calculating using Dynamic Programming

Analytic Method

It is an algebraic based method to obtain V(s).



$$V = V + gamma * PV$$

Simplifies to:

$$V - gamma * PV = R => V(1 - gamma * P) = R => V = (1 - gamma * P)^{-1}R$$

So, the analytic solution is $V = (1 - gamma * P)^{-1}R$

The time complexity of the method is $O(T^3)$

Dynamic Programming Method

The dynamic programming method is based on the iterative backward flow of information until convergence. In this method, we initialize two value function vectors V_curr and V_prev. V_curr is a vector of dimension (N x 1) having all 0s, and V_prev is a vector of Dimension (N x 1) having all 1's. Then we update the vectors in each step using the equation.

$V\_curr = R + gamma * PV\_prev$, until V_curr or the value function converges. Mathematically, we keep updating up to a point where L2 Norm of (V_curr- V_prev)< *Epsilon.* Epsilon is a constant with a very small value, mostly 1x 10^-2 to 1x 10^-8. So, the pseudo-code is:

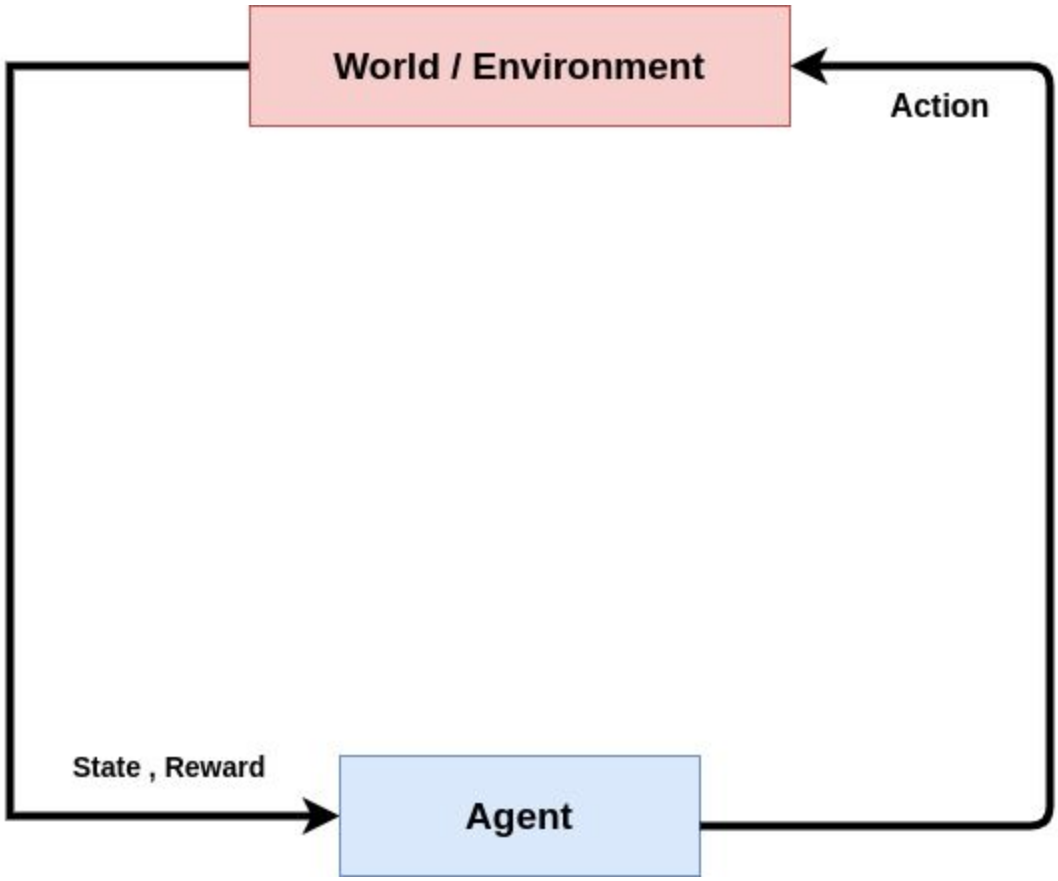
V_curr={0}
V_prev={1}

While ||V_curr-V_prev||>Epsilon:
    V_prev=V_curr
    V_curr= Reward+ gamma P.V_prev*

On convergence, the value function almost remains the same.
Now, both MP and MRP are self-driven processes i.e, they don't consider the action of the agent. Next, we will look at MDP which considers the action of the agent,

Markov Decision Process

Markov Decision Process is a Markov Reward Process with actions. So here the full Reinforcement Learning setting is followed.



The World represented by MDP provides the Agent with State and Reward. The Agent responds with action. The MDP considers the action and provides back a reward corresponding to the action and a new state to the agent.

MDP is represented by a 5 tuple: (S, A, P, R, gamma)

1. S represents the set of states
2. A is the action space for the agent. In easier words, it is the set of actions that the agent can take in order to interact with the world.
3. P represents the transition probability dynamics
4. R represents the Reward Dynamics
5. Gamma represents the discount factor.

When the action steps in, the probability matrix and reward matrix will completely vary depending on the actions. As we have already seen in the definition of the Reinforcement Learning setting, the world provides a next state and reward based on the action taken by the agent.

To understand this, let's go through very practical experience. Say, we start from a very source location. We have four movements moving forward, moving backward, turning left and turning right, which defines our action space. The real environment and the city routes and tracks pose as the environment and we function as the agent. We have a destination location to reach, which is our goal state, so it will have a reward there. Now, from practical experience, we have seen there can be several routes between a source and a destination.

The question is how the routes differ from each other? The answer is, they differ in the action sequences taken by the agent to follow the route. Say, for a route A, we follow the action sequence [forward, left, right, forward], and for a Route B, we follow the action sequence [left, forward, right, forward]. If we consider each crossing of the roads a state (for the sake of simplicity), we can be sure that we will face different states in both the routes or at least they will vary in order. So, we can conclude taking different action sequences will lead us to the same destination, but the routes are taken and the states faced will be different.

The **action sequences** are called **policies**. Different policies result in different routes. Some of them may reach the goal state, some of them may not. Our target is to reach the goal state optimally, as a result, we need to follow an optimal route and consequently get an optimal policy.

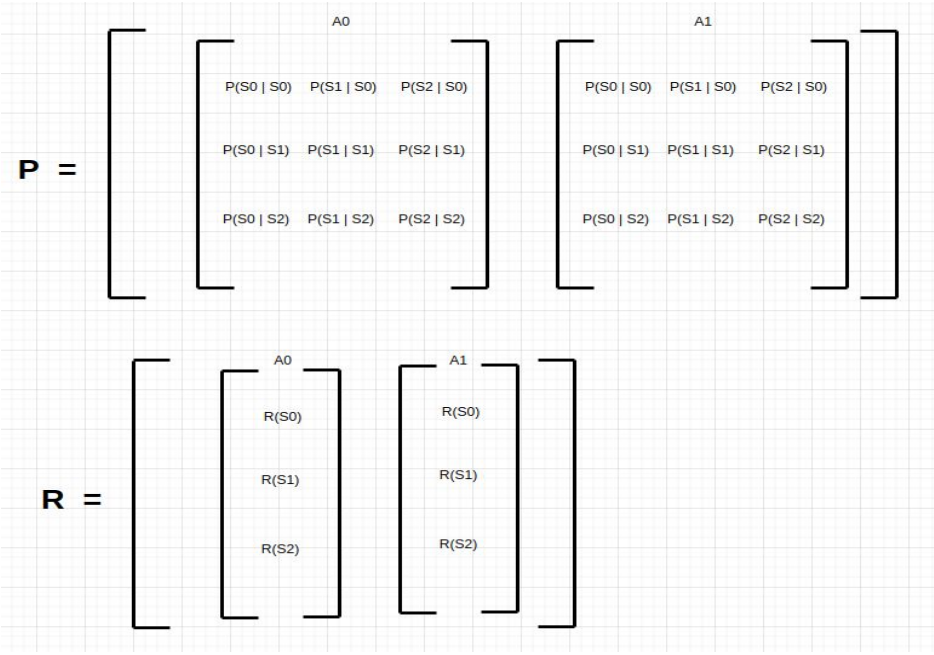
Again, it is pretty evident that, standing at a crossing, if we take a left turn, the results or the upcoming crossings will not be the same with the upcoming crossings if we take a right turn or decide to move straight without a turn, at the crossing,

I think now we are clear how the states and rewards will vary with actions, and correspondingly the routes vary with the **policies** taken. Our main goal is to find the best policy.

So, let's see how, an MDP declaration reflects all the points.

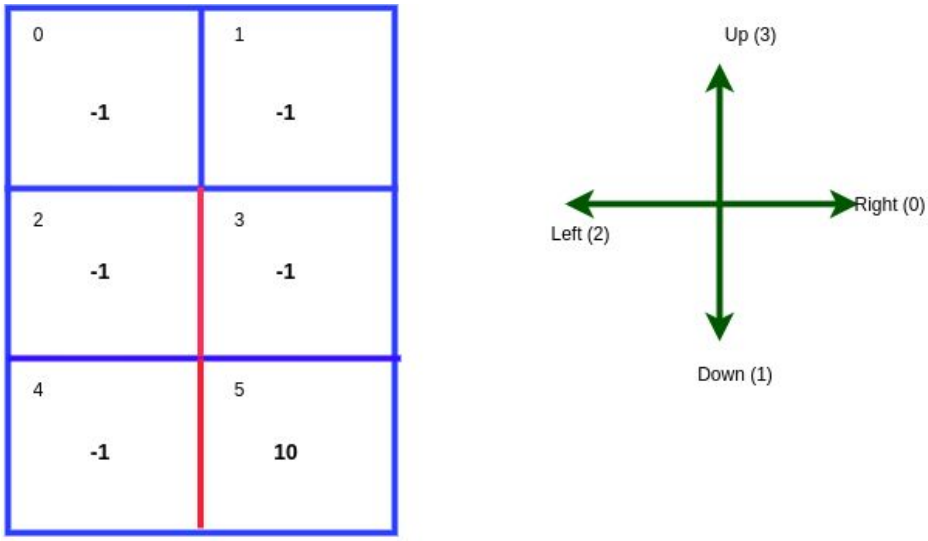
Here our Probability Transition matrix will have a shape of A x S x S, our Reward Matrix will have the shape A x S x 1, where S will be the dimension of the set of states and A will be the dimension of the set of actions.

We will have separate reward and probability transition matrix for each action, which shows the rewards and the next state if the agent takes that corresponding action at the given state.



This is how we define a system with 3 states and 2 actions. States are given by (S0, S1, S2), and actions are given by (A0, A1). We can see we have a separate transition and reward matrix for all the different actions. The matrices corresponding to a particular action is similar to our declarations for MRP.

Let's see for a defined problem statement.



Let's say this is our MDP problem statement. Red walls are the block walls. And the diagram shows the action space.

```
Ptr=np.array([
    [
        [0,1,0,0,0,0],
        [0,1,0,0,0,0],
        [0,0,1,0,0,0],
        [0,0,0,1,0,0],
        [0,0,0,0,1,0],
        [0,0,0,0,0,1],
    ],
    [
        [0,0,1,0,0,0],
        [0,0,0,1,0,0],
        [0,0,0,0,1,0],
        [0,0,0,0,0,1],
        [0,0,0,0,1,0],
        [0,0,0,0,0,1],
    ],
    [
        [1,0,0,0,0,0],
        [1,0,0,0,0,0],
        [0,0,1,0,0,0],
        [0,0,0,1,0,0],
        [0,0,0,0,1,0],
        [0,0,0,0,0,1],
    ],
    [
        [1,0,0,0,0,0],
        [0,1,0,0,0,0],
        [1,0,0,0,0,0],
        [0,1,0,0,0,0],
        [0,0,1,0,0,0],
        [0,0,0,1,0,0],
    ],
])
```

```
R=np.array([
    [
        [-1],
        [-1],
        [-1],
        [-1],
        [-1],
        [10]
    ],
    [
        [-1],
        [-1],
        [-1],
        [10],
        [-1],
        [10]
    ],
    [
        [-1],
        [-1],
        [-1],
        [-1],
        [-1],
        [10]
    ],
    [
        [-1],
        [-1],
        [-1],
        [-1],
        [-1],
        [-1]
    ],
])
```

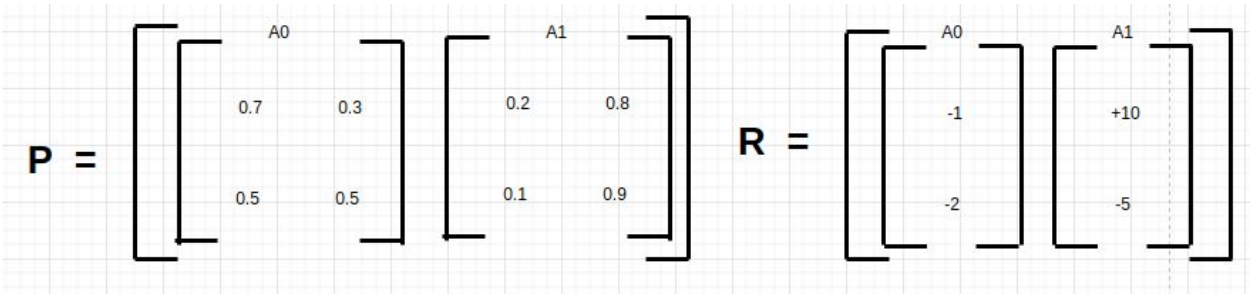
The above two images give the declaration of the probability transition matrix and the reward matrix for the given statements. These are deterministic approaches considering a no-slip property, i.e, if action a1 at state s1 leads the agent to state s2, it will do so with a 100% probability or in no way the agent will land at any other state than s3.

Let’s visualize an instance. We can see if an agent is present at state 0, and we take the 0th action (move right) we move to state 1 and have a reward of -1. The matrix corresponding to the 0th position of Ptr(transitional Probability matrix) represents action 0. The 0th row of the 0th matrix represents the classical vector of the next state that would be reached if the agent takes

action 0 at the state 0. Here it is [0 1 0 0 0 0] i.e, it represents the 1st state, which is true. Similarly, the 0th row of the matrix present in the 0th position of the R(reward matrix) represents the reward gained by the agent if it takes action 0 at state 0. In this case, it is -1, which is true from the actual visualization. One thing to take note is that in MDP, we represent each action as a one-hot encoding of the action space. So, action 0 is represented by [1 0 0 0].

Here, with action, the transition dynamics and reward matrices change. Let's see how so.

We will consider a smaller example to study this.



S = (S0, S1) A = (0, 1)

Say, initially we are at the state S0. The classical representation vector [1 0]. Now if we take action 0 at S0. Action 0 vectorized is represented by [1 0].

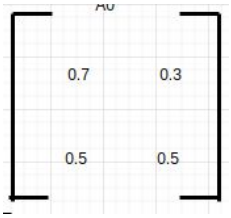
To generate the next state, we first need to select the reward and transition dynamics matrix corresponding to the action. For that,

$T\_dr = A \; x \; P$ and $R\_dr = A \; x \; R$

A has dimension 1 X Action space(A) here [1 0]
P has dimension A x S x S. After multiplication the obtained matrix will be 1 X S X S. So, to obtain the actual dynamics we need to get T_dr[0]

Similarly, R_dr has dimension 1 x S x 1. So, R_dr[0] is taken.

P_tr= T_dr[0], in our case is given by:



The R_tr= R_dr[0] in our case is:

[-1 -2]

After this, steps are similar to MP and MRP. The P_tr and R_tr are multiplied by state vectors. For state S0, the vector is [1 0]

After multiplication, we obtain the transition probabilities, for a particular state under a particular action.
Here for state S0 under action 0, the **transition probabilities are [0.7 0.3] and the reward is -1.** Next, we sample from the probability distribution to obtain the next state. There is a 70% probability to obtain state S0 and a 30% chance of going to state S1.
This way MDP operates.

Evaluation and Control

Now, **how we judge which policy is better?** If you remember, in the last article, the aim of reinforcement learning is to produce the maximum reward while reaching the goal state. Here, we are stating our goal is to find the best policy. So, they must be related. Let's see how.

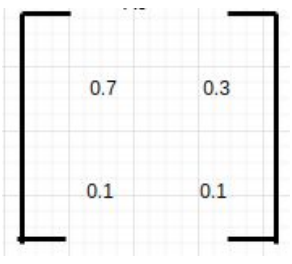
"A policy is judged on the basis of the value function it produces for all the states in an MDP." In MRP, we have seen the agent was a self-driven one, i.e, it was transiting to a state from a previous one based on a declared probability matrix which was fixed. But, now, we have seen in MDP, we need to first obtain the probability matrix and the dynamics matrix corresponding to the taken action. After we obtain the matrices the methods are the same. This is the only difference between and MDP and MRP. In MRP, the transition matrix does not vary, in MDP it varies with the action taken.

If we think very carefully, A MDP with a fixed policy becomes an MRP. For example, if we declare that at S0 action taken will be 0, and at S1 action taken will be 1, we declare a mapping from a state to action, i.e, we are providing a policy. Now, as we have fixed the function,

For State S0, we will always sample from the probability [0.7, 0.3] as that is the probability distribution we obtain on performing action 0 on state 0, and we will always obtain a reward of -1.

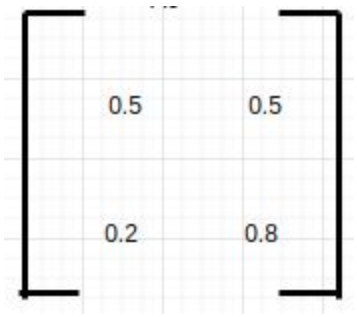
Similarly for S1, we will obtain the probability matrix as [0.1, 0.9] and obtain a reward of -5.

So, we can redeclare our MDP under policy PI_1=[0, 1] as:

P = 

and **R = [-1 -5]**

Similarly, now if we change our policies to PI_2= [1, 0] i.e, action 1 at state S0 and action 0 at state S1, the MRP created changes to

P = 

and **R = [-2 +10]**

So, we have seen how on changing the policy the MRP changes.

Evaluation, according to the definition is about finding the value function under a given policy. We have seen under a given policy value function is no different than an MRP, and we know how to find the value function of an MRP using both the dynamic programming and analytical method.

Control is the method to find the best or the most optimal policy for a reinforcement learning problem. We know the best policy is the one for which the value function of the MDP is maximum.

Given an MDP with S states and A actions. The number of policies possible A^S.

This is because we have S states and can take ‘A’ actions on every state.

Policy Iteration

It is a method to find the best policy evaluating the value functions produced by each of them. This is kind of a naive method search. It generates all the possible policies, finds the value function for each of the policies, it then picks the policy with the maximum value function. Basically it iterates over all possible policies.

So, the algorithm can be marked as:

1. *Generate all policies ‘pi’ possible with a given action space for some n number of states.*
2. *record=[]*
3. *For Policy P in PI:*
    a. *V(PI) <- Calculated value function under Policy PI*
    b. *Append Tuple (P, V(PI)) to record*

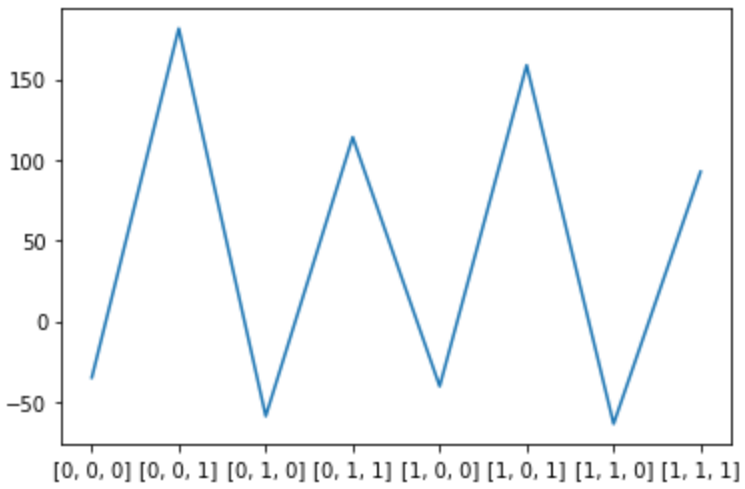
Let’s see an example,

```
P_tr=np.array([
    [
        [0.1,0.5,0.4],
        [0.2,0.3,0.6],
        [0.3,0.35,0.35]
    ],
    [
        [0.2,0.6,0.2],
        [0.1,0.1,0.8],
        [0.5,0.25,0.25]
    ]
])
```

```
R_tr=np.array([
    [
        [-1],
        [2],
        [-3]
    ],
    [
        [-2],
        [-1],
        [10]
    ]
])
```

Consider the above problem, Action given by: [0, 1] and States are given by: [0, 1, 2].

Now, after policy iteration, if we plot all the policies and their corresponding value functions, we get,



We can observe that at Policy [0, 0, 1] the value function is the maximum.

From observations, we can say [0, 0, 1] is chosen as the optimal policy for the statement.

Challenges with policy iteration:

The main challenge with policy iteration is, it iterates over all the policies. As we can observe, we already achieved the highest policy at the 2nd iteration still make all the iterations. This is quite a naive method and for a problem with a large number of states and explode the number of policies will explode owing to combinatorial explosion. So, the process will not conclude ever.

Policy Improvement

Policy improvement is a modified version of the policy iteration to solve the explosion issue. It works on the convergence theorem.

The method only focuses on the policies that increase the value functions. In simpler words, if we consider the previous statement that we considered for policy iterations, we will get a non-decreasing monotonic curve which will converge at a point after some iteration, instead of iterating over all the policies found using the combination.

To get through, we will need to know a few things

State-Value function Q

Q under a policy PI is defined as:

$$Q(PI)(s,a) = R(s,a) + gamma * \Sigma P(s' \mid s,\ a)V(PI)(s')$$

So, Q function under a policy PI, for a state s under action a is given by the sum of reward at state s under action a and the discounted probabilistic sum of the value functions of the next state s' under policy PI.

The computation of the next policy and value function is calculated as:

$$PI_{i+1}(s) = argmax\ (\ Q(PI_i)\ (s,a))\ \forall\ s\ \in S$$

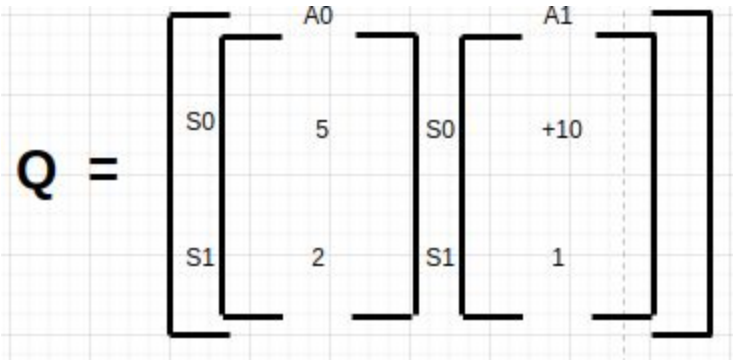
$$V(PI_i)(s,a) = max\ (\ Q(PI_i)\ (s,a))\ \forall\ s\ \in S$$

Explanation: In this case, we don't pick a particular policy from all the combinations and try to find the value function for each possible policy. Previously for policy iteration, we applied actions to an MDP, generated the corresponding MRP, and then we generated the MRP and found the value function.

Here, we generate the value function for all the actions from a given state and try to select the action, that gives the best value function from the state. The Q function provides this information. So, here policy is not fixed, it is devised.

Let's see in simple words, P has dimension (A x S x S), V (S) is of dimension (S x 1). So, Q function is of format (A x S x 1). So, for every action, there is a corresponding value function from a given state.

Visualizing by an example,



Say, for a problem with 2 states and 2 actions, this is the Q value. Next,

We apply the equation to select the PI and the V(S). We apply the equations on the actions axis or axis 0. So, for the State S0, we pick the value +10 as obtained by action A1 as its the maximum value produced, the other option being 5 obtained by action A0. So, the value function we obtain is

V = [10, 2] That is the maximum value obtained for both the states obtained by taking action A1 at S0 and action A0 at S1. For picking the actions, we pick the index corresponding to the maximum value function.

As a result, our policy becomes **[1, 0]** which represents action 1 at state 0 and action 0 on state 1.

Now, as we chose only the best and the highest value, we can be very sure the value never decreases and the next policy is better than the previous policy.

So, here we can apply the convergence property because after a point we will find the best policy and the value function will not change. When we reach the convergence we can say to have reached the optimal policy.

There are two ways to do the mentioned:

1. Modified Policy Iteration approach
2. Value iteration approach.

Both of them differ in the convergence condition.

1. For the policy iteration approach, convergence is said to have been achieved if the policy does not change int two consecutive iterations.
2. For the value iteration approach, convergence is said to be achieved if the L2 norm of the difference of the previous value function and the current value function is less than a given constant epsilon.

It has been observed that the value iteration often performs better and converges to more optimal value and the policy iteration may get stuck at a lower optimum point.

Algorithm for Value iteration:

Q=np.zeros(P.shape)
V=np.zeros(R.shape)
V_prev=np.ones(R.shape)
epsilon=0.000000001
log=[]

While || V - V_prev || >epsilon:
$Q = R + gamma * PV$
V_prev=V
$V = max(Q)\ on\ axis = 0$
$PI = argmax(Q)\ on\ axis = 0$
log.append(PI, V.sum())

Best_policy= sort(log)[-1] # last index

Algorithm for Policy iteration:

Q=np.zeros(P.shape)
V=np.zeros(R.shape)
Pi=np.zeros(R.shape)
V=np.zeros(R.shape)
Pi_prev=np.ones(R.shape)

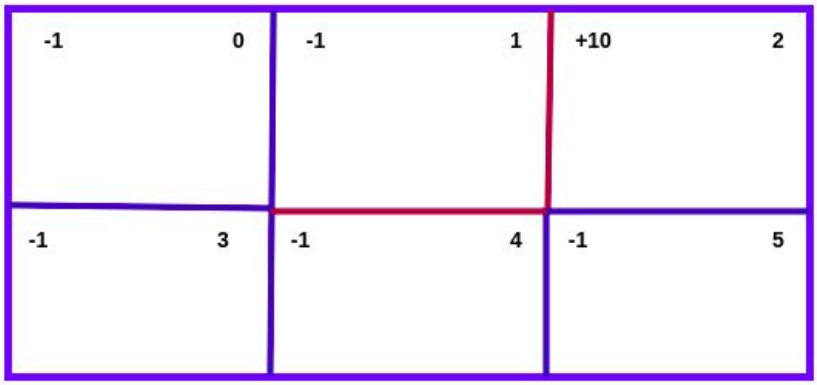
epsilon=0.000000001
log=[]

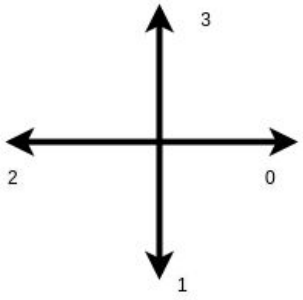
While Pi != Pi_prev:
$Q = R + gamma * PV$
Pi_prev=Pi
$V = max(Q)\ on\ axis = 0$
$PI = argmax(Q)\ on\ axis = 0$
log.append(PI, V.sum())

Best_policy= sort(log)[-1] # last index

Now, let’s demonstrate the method for a particular problem state.

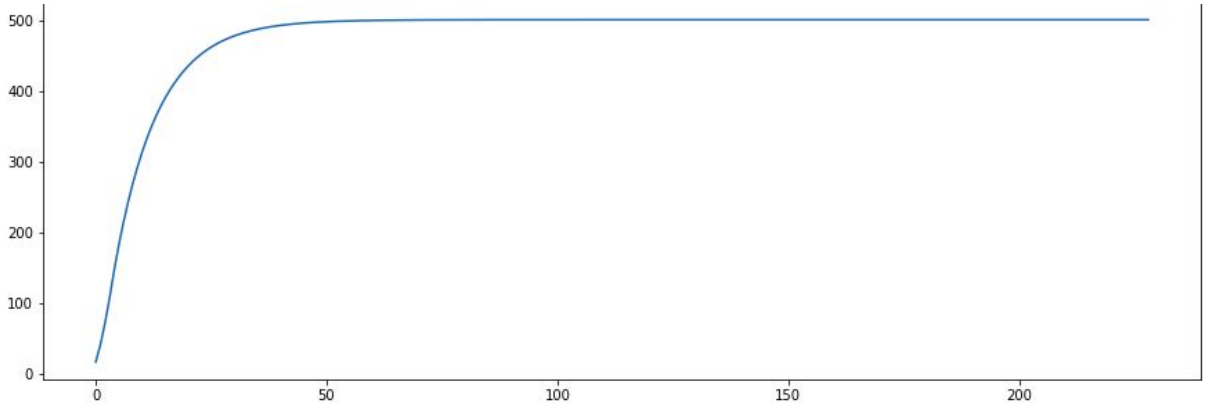
Problem:





Value Iteration:

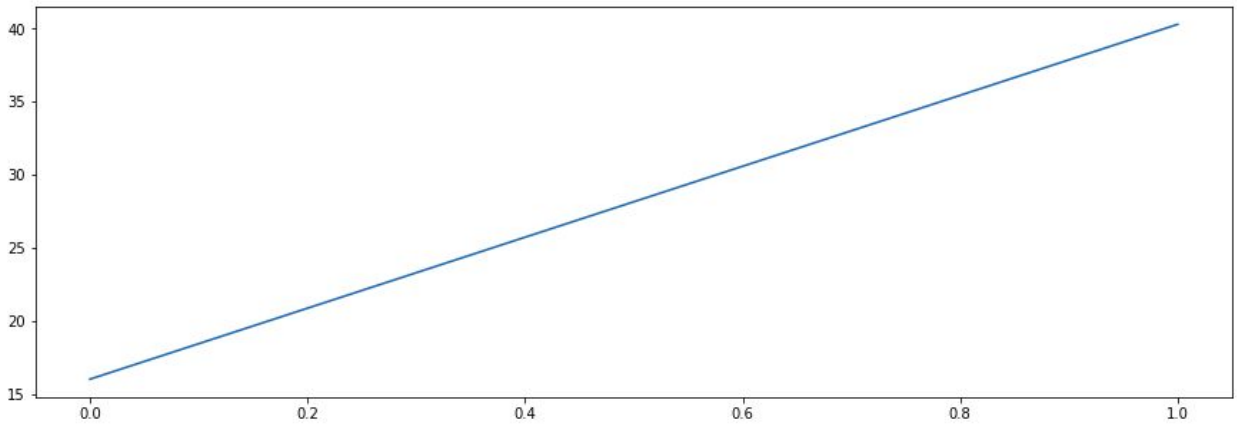
Best policy: [1 2 0 0 0 3]
Sum of the value function: 500



The function converges after 500 iterations.

Policy Iteration:

Best policy: [0 0 0 0 0 3]
Sum of the value function: 40



The function converges on 1 epoch

So, we can see value iteration performs much better.

NOTE: The update functions:

$$PI_{i+1}(s) = argmax\ (\ Q(PI_i)\ (s,a))\ \forall\ s\ \in S$$

$$V\ (PI_i)(s,a) = max\ (\ Q(PI_i)\ (s,a))\ \forall\ s\ \in S$$

Are based on deterministic approaches i.e, the condition under which we move from one state to another state with 100% accuracy on an action.

For Probabilistic approaches, we can't use 'argmax' function. There we use, softmax function.

Conclusion

In this portion, we have seen MP, MRP, MDP, Evaluation, and Control for model-based operations and policy iterations and improvement.