Markov Modelling

A Preprint

Kshitij Bichave[*]

March 3, 2021

Keywords Reinforcement Learning · Markov-chain · Markov-Reward · Markov Process · Return · Value Function · Return · Discount Factor · Markov Decision Process · MDP Control

Markov Process/Chain:

Sequence of random states with Markov property (that is, $p(s_{t+1}|s_t, a_t) = p(s_{t+1}|h_t, a_t)$)

Definition of Markov Process:

- S is a set of states, such that $s \in S$
- P is a dynamics model which speicifies $P(s_{t+1} = s'|s_t = s)$

Discount Factor:

Used to avoid infinite return so that expected sum of return is bounded. If $\gamma = 0$, agent only cares about immediate reward. If $\gamma = 1$, future reward is a beneficial as immediate reward. γ can be set to 1 if the horizon is finite

Markov Reward Process (MRP):

Markov reward process is a Markov chain with addition/inclusion of reward term. Because MRP includes rewards and rewards can weighted for how much we care about the future rewards, γ is included in MRP.

Definition of Markov Process:

- S is a set of states, such that $s \in S$
- P is a dynamics model which specifies $P(s_{t+1} = s'|s_t = s)$
- R is a reward function $R(s_t = s) = \mathbb{E}[r_t|s_t = s]$
- Discount factor $\gamma \in [0, 1]$

Return & Value Function:

Horizon: Number of time steps in each episode, can be infinite of finite.

Return G_t (for MRP) is a discounted sum of rewards from time step t to horizon. $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...$

State Value Function $V(s)$ (for MRP) is the expected return from starting state s. $V(s) = \mathbb{E}[G_t|s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s]$. The State value function can be deterministic or stochastic. If deterministic, the expected return is always the same, because there is only a single next state the agent can go to.

Computing Value for Markov Reward Process:

[*]GitHub: github.com/kbichave | Email: k.bichave@gmail.com | LinkedIn:linkedin.com/in/kshitijbichave

Using simulation:

- Take starting state distribution and generate large number of episodes
- Average returns
- Concentration inequalities bound how quickly average concentrates to expected value

Advantage: Does not require assumption of Markov

Given Markov property:

$V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V(s')$.

Here, $R(s)$ is the immediate reward and the second term in the equation is the sum of future rewards discounted by discount factor γ.

Using Dynamic Programming (iterative algorithm):

- Initialize $V_0(s) = 0$ for all s
- for all k = 1 until convergence (condition for convergence could be $|V_k - V_{k-1}| < \epsilon$, where ϵ is some allowed error term
  - For all $s \in S$, $V_k(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V_{k-1}(s')$

Markov Decision Process (MDP):

Markov Decision Processes are Markov Reward Process with addition/inclusion of actions

Definition of Markov Decision Process:

- S is a set of states, such that $s \in S$
- A is a set of actions, such that $a \in A$
- P is a dynamics model for each action which specifies $P(s_{t+1} = s'|s_t = s, a_t = a)$
- R is a reward function $R(s_t = s, a_t = a) = \mathbb{E}[r_t|s_t = s, a_t = a]$
- Discount factor $\gamma \in [0, 1]$

MDP is defined as a tuple: (S, A, P, R, γ)

MDP Policies:

Policies specifies which action to take in each state. The policies can be deterministic or stochastic. That is, does the same action when in that state, or has a distribution over actions and thus may land in different states.

Policy: $\pi(a|s) = P(a_t = a|s_t = s)$

MDP + Policy:

Markov Reward Process, MRP $(S < R^\pi, P^\pi, \gamma)$

$R^\pi(s) = \sum_{a \in A} \pi(a|s)R(s, a)$

$P^\pi(s'|s) = \sum_{a \in A} \pi(a|s)P(s'|s, a)$

This implies that we have fixed policies, simulation, analytic soulution or dynamic solution could be used to compute value of the policy.

Dynamic programming now becomes:

- Initialize $V_0(s) = 0$ for all s
- for all k = 1 until convergence (condition for convergence could be $|V_k - V_{k-1}| < \epsilon$, where ϵ is some allowed error term
  - For all $s \in S$, $V_k^\pi(s) = r(s, \pi(a|s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(a|s))V_{k-1}^\pi(s')$

This is also called Bellman backup for a particular policy.

MDP Control:

MDP control now talks about not just evaluating policies but, learning, that is computing the optimal policy, $\pi*(a|s) = \text{argmax}_a V^\pi(s)$.

There exists an unique optimal value function Optimal policy for a MDP.

In an infinite horizon problem optimal policy is deterministic and stationary (does not depend on time step).

Given 7 discrete states and 2 action, the number of deterministic policies for this example are 2^7.

Optimal policy is not always unique. But value function is unique.

MDP Policy Iteration:

In policy iteration, you keep track of what the optimal policy might be, we evaluate it's value and then improve it.

- set $i = 0$
- Initialize π_0 randomly for all states s
- While $i == 0$ or $\|\pi_i - \pi_{i-1}\| > 0$ (L1-norm, measures if the policy changed for any state):
  - $V^{\pi_i} \leftarrow$ MDP V function policy evaluation of π_i
  - $\pi_i + 1 \leftarrow$ Policy improvement
  - $i = i + 1$

In order to define how, policy could improve, define state-action value $Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V^\pi(s')$. That is take action a, then follow the policy π.

Policy Improvement:

After we have a policy from policy evaluation, and thus we have value of it. That is, we know V^{π_i}. We also know what the dynamics and reward model is. So now,

Compute the state-action value of policy π_i

For $s \in S$ and $a \in A$:

$Q^{\pi_i}(s,a) = R(s,a) + \gamma \sum P(s'|s,a)V^{\pi_i}(s')$

Compute new policy π_{i+1} for all $s \in S$. That is choose a policy such that it maximizes Q^{π_i}.

$\pi_{i+1}(s) = \text{argmax}_a Q^{\pi_i}(s,a)\ \forall s \in S$

In theory, $\text{argmax}_a Q^{\pi_i}(s,a) \geqslant Q^{\pi_i}(s, \pi_i(s))$. Because, $a = \pi_i(s)$ and thus will be same of better according to argmax. This is guaranteed because of monotonic improvement of policy. That is, something is monotonic is new is equal or better then older.That is $V^{pi_i+1} \geqslant V^{\pi_i}$ with strict inequality if π_i is sub-optimal, where π_{i+1} is the new policy we get from improvement on π_i. So, if $\pi_{i+1} = \pi_i$ for some i, then the policy can never get better beyond π_i.

Bellman Equation and Bellman Backup Operator

$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V^\pi(s') \leftarrow$ Bellman equation

Can be thought of Bellman backup operator. That is when applied to value function, return a new value function and improves the values if possible. BV yields a value function over all states s.

$BV(s) = \max_a R(s,a) + \gamma \sum_{s \in S} p(s'|s,a)V(s')$

Value Iteration:

Idea is that maintain optimal value of starting in a state s if have a finite number of steps k left in episode. Iterate to consider longer and longer episodes. Value iteration assumes you have optimal value and policy only if you are going to act for k time steps (finite horizon).

- set $k = 1$
- Initialize $V_0(s)$ randomly for all states s
- Loop until convergence:
  - For each state s,

$V_{k+1}(s) = \max_a R(s,a) + \gamma \sum_{s \in S} p(s'|s,a) V_k(s')$.

That is Bellman backup on value function, $V_{k+1} = BV_k$.

- $\pi_{i+1}(s)$ $\operatorname{argmax}_a R(s,a) + \gamma \sum_{s \in S} P(s'|s,a) V_k(s')$

Bellman operator can be initialized by fixing the policy B^π, instead of taking max_a.So to do policy evaluation, repeatedly apply operator until V stops changing.

Value iteration will converge if discount factor $\gamma < 1$ or ends up in terminal state with probability 1. Therefore, it can be said that Bellman backup is contraction if $\gamma < 1$. If applied to to two different value functions, distance between value functions shrinks after applying Bellman equation to each.