## 1 Markov Decision Process (MDP)

MDP's are a mathematically idealized form of RL problems, finite MDP constitute of a finite set of States ($\mathcal{S}$), finite set of Actions ($\mathcal{A}$) and finite set of possible rewards ($\mathcal{R}$), also $s' \in \mathcal{S}^+$ is used to represent the set of states plus the terminal state , and every element in those sets have a well defined discrete probability.

A sample trajectory looks like $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \dots$ where an action $A_t \in \mathcal{A}$ in State $S_t \in \mathcal{S}$ in the environement returns a next state $S_{t+1} \in \mathcal{S}$ and a reward $R_{t+1} \in \mathcal{R}$.

The **dynamics** of the MDP are defined as

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{1}$$

which is a deterministic discrete probability distribution. Where each next state and reward are completely dependent only on the preceding state, and no state before that (restriction of S to be expressive). And we call the state to have **Markov Property**.

We can express some other functions using 1

$$p(s'|s, a) \doteq \sum_{r \in \mathcal{R}} p(s', r|s, a) \tag{2}$$

$$r(s, a) \doteq \mathbb{E}[R_t|s, a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a) \tag{3}$$

$$r(s, a, s') \doteq \mathbb{E}[r_t|s, a, s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a}{p(s'|s, a)} \tag{4}$$

]The general rule being anything that cannot be changed arbitrarily by the agent is considered to be outside of it and part of the environment, like robot limbs, reward signals etc. The agent-environment boundary represents the limit of the agents absolute control not of its knowledge.

Some important points I think . we should know

1. Reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want to do

### 1.1 Return and Episode

The return $G_t$ that the agent tries to maximize is often defined as

$$G_t \doteq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots \tag{5}$$

$$G_t = R_t + \gamma G_{t+1} \tag{6}$$

$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{7}$$

$$= \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \text{ for episodic cases} \tag{8}$$

where gamma is defined as the discounting factor to avoid running into infinities.

### 1.2 Unifying notation for episodic and continuing tasks

WE can use the same notation for episodic tasks by putting the constraint that at the ending time T, the agent enters an absorbing state with zero reward and we condition following as $T = \infty$ or $\gamma = 1$ but not both.

### 1.3 Policies and Value Functions

**Value Function** : of state $s$ under policy $\pi$ denoted by $\mathrm{v}_\pi(s)$ defined as is the expected return starting in that state and following $\pi$ from there.

$$\mathrm{v}_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \forall s \in \mathcal{S} \tag{9}$$

**Important** : The value of the terminal state, if any, is always zero. Whereas $\mathrm{q}_\pi(s, a)$ is defined the expected run of taking action $a$ in $s$ and then following policy $\pi$.

$$\mathrm{q}_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{10}$$

### 1.4 Recursive form of $\mathrm{v}_\pi(s)$ and $\mathrm{q}_\pi(s, a)$

Bellman equation for value function under policy $\pi$. The value function $\mathrm{v}_\pi$ is the unique solution to its Bellman equation.

$$\mathrm{v}_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)[r + \gamma\, \mathrm{v}_\pi(s')] \forall s \in \mathcal{S} \tag{11}$$

$$= \mathbb{E}_\pi[q(s, A_t)] \tag{12}$$

$$\mathrm{q}_\pi(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma \sum_{a}^{'} \pi(a'|s')\, \mathrm{q}_\pi(s', a')] \forall s \in \mathcal{S} \tag{13}$$

$$= \mathbb{E}_\pi[R(s, a) + \gamma\, \mathrm{v}_\pi(S_{t+1})|S_t = s, A_t = a] \tag{14}$$

### 1.5 Optimal Policies and Value Functions

A better or equal policy $\pi$ over $\pi'$ is defined as $\pi \geq \pi'$ iff $\mathrm{v}_\pi(s) \geq v_{\pi'}(s) \forall s \in \mathcal{S}$. There is always one policy better than a policy and an optimal policy $\pi^*$ which is better than any other policy(can be more than one). and share the same state value function. i.e. $v_*(s) \doteq \max_\pi \mathrm{v}_\pi(s)$ and $q_*(s, a) \doteq \max_\pi \mathrm{q}_\pi(s, a)$ and $q_*(s, a)$. $= \mathbb{E}[R_{t+1}. + \gamma v_*(S_{t+1})|S_t = s, A_t = a]$ The value of a state under an optimal policy must equal the expected return for best action from that state.

$$v_*(s) = \max_{a \in \mathcal{A}} q_{\pi*}(s, a)$$

$$= \max_a \sum_{s', r} p(s', r|s, a)[r + \gamma . v_*(s')] \tag{15}$$

Similarly for the q value function we get.

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a]$$

$$= \sum_{s', r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{16}$$

A policy that is greedy with respect to $v_*$ will be an optimal policy. Having the knowledge of $q_*(s, a)$ eliminates the need to do a one step search on the actions, and value of future states, and also the environment dynamics.

## 2 Dynamic Programming

DP can be used to compute optimal policies given a perfect model of the environment as an MDP , but their highly

compute intensive.

### 2.1 Policy Evaluation

The value function for a policy is know as policy evaluation and normally equation 9 can be thought of as a system of $|S|$ equations, where each term is a linear dependent on the other variables, and there are $|S|$ unknowns. There are also two methods to solve this, 1. Iterative 2. Analytical, we describe the iterative method. We choose initial approaximation for each state $v_0$ (except for terminal state, which is 0) and use the bellman equation for the update rule.

$$v_{k+1}(s) \doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}|S_t = s] \quad (17)$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \quad (18)$$

All the updates done on DP algorithms are called expected updates because they are based on expectation of the next states rather than sampling of the next states.

### 2.2 Policy Improvement

We compute a value function as to improve the existing policy. Determining $\mathrm{v}_\pi$ for a policy $\pi$, we want to know wether we can improve on the policy i.e. take action $a \neq \pi(s)$ and improve our value function in general. One way of seeing this is taking action $a$ in $s$ and then following policy $\pi$ afterwards, which can be written as : $\mathrm{q}_\pi(s, a)$ where a might not be sampled from $\pi$. Criterion being if this greater than $\mathrm{v}_\pi(s)$ and then we can replace this step with always taking action as it will always imprive the value function for $v_{\pi'}(s)$, where the new policy can be taking action $a = \pi'(s)$ and then following $\pi$ for rest.

**Policy Improvement Theorem** $\pi$ and $\pi'$ are 2 deterministic policies, such that $\forall s \in \mathcal{S}$

$$\mathrm{q}_\pi(s, \pi'(s)) \geq \mathrm{v}_\pi(s) \quad (19)$$

Then the policy $\pi'$ must be as good as, or better than $\pi$. This is a generalisation of the above statement.

$$v_{\pi'}(s) \geq \mathrm{v}_\pi(s) \quad (20)$$

Proof the same has been expanded in great detail later. Now we can extend the single change in action to all states, considering a new greedy policy $\pi'$s given by,

$$\pi'(s) \doteq \underset{a}{\text{argmax}}\ \mathrm{q}_\pi(s, a) \quad (21)$$

, where we can distribute the policy probability equally on the multiple best action it is okay until and unless all the suboptimal actions get a 0 probability value. This meets with the condition of the Policy Improvement Theorem 19. This process of greedyfying is often called as policy improvement.

**Getting the Optimal Policy** : Suppose new policy $\pi'$ is as good as and not better than $\pi$ i.e. $\mathrm{v}_\pi = \mathrm{v}_{\pi'}$ and from 21 we get $\forall s \in \mathcal{S}$.

$$\mathrm{v}_{\pi'}(s) = \max_{\mathrm{a}} \sum_{s',r} p(s', r|s, a)[r + \gamma\, \mathrm{v}_{\pi'}(s')]$$

Which matches the Bellman Optimality equation there $\mathrm{v}_{\pi'} = v_*$

### 2.3 Policy Iteration

The process of alternating between policy evaluation and policy improvement to approach the optimal policy, is called as policy iteration. A policy $\pi$ can be improved using $\mathrm{v}_\pi$ to get a new policy $\pi'$, as summarized in this diagram

$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}$

, where $\xrightarrow{E}$ means policy evaluation and $\xrightarrow{I}$ implies policy improvement

### 2.4 Value Iteration

The policy evaluation part is most of the time a very compute intensive operation and doing it multiple times only makes it more worse. We dont need exact convergence, we can work with a few updates to te polciy evalutaion process. Special case being when policy evaluation is stooped after on e sweep and called as **value iteration**. We can obtain it by turning the bellman optimality equation into an update rule, and it is also similar to policy evaluation update it differs in taking a max on a.

$$v_{k+1}(s) \doteq \max_{\mathrm{a}}\ \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}|S_t = s, A_t = a] \quad (22)$$

$$= \max_{\mathrm{a}} \sum_{\mathrm{s'},\mathrm{r}} \mathrm{p(s', r|s, a)}[r + \gamma v_k(s')] \quad (23)$$

Same form can be written for the state-action update

$$q_{k+1}(s, a) \doteq \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_k(s', a)] \quad (24)$$

$$= \sum_{\mathrm{s'},\mathrm{r}} \mathrm{p(s', r|s, a)}[r + \gamma \max_{a'}\ q_k(s', a')] \quad (25)$$

### 2.5 Asynchronous DP

DP suffer from the problem going in a deadlock of sweeping through the entire state space even for a single value iteration. We can avoid this by updating the states no fixed order to maybe focus on states that are more important for the current context, or ignoring those state that are all together not required for the optimal behaviour. This can also be used to learn in an online fashion.

### 2.6 Generalized Policy Iteration

PI consists of 2 simultaneous, interacting process, i.e. policy evaluation and policy improvement. GPI refers to the general notion of PE and Policy Improvement. All RL algorithms try to solve this only we try to estimate the value function in order to act greedy with respect to that , me improve the policy from that and then we again to estimate the same using the new improved policy to improve it further. When both the process stabilise we reach the optimal policy and the optimal value function. And we say a policy has been found when it is greedy with respect to its own value function.

### 2.7 Efficiency of Dynamic Programming

Efficiency of DP : DP can be said to be a polynomial time algorithm in the number of states (s) and the number of action (k), being considerably better than some of the search methods who might have to search through $k^n$ policies.s

## 3 Proofs

Proofs for $\mathrm{v}_\pi(s)$ and Policy Improvement Theorem are attached separately.

## References


[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.