MODULE 1.

1. The K-armed Bandit Problem.

agent ⇒ decision maker

↳ chooses between actions (k-actions)

↓

based on the choice gets reward

action-values :- expected reward of that action.

↳ $q_*(a) := \mathbb{E}[R_t \mid A_t = a] \quad \forall \; a \in \{1, 2, \ldots, k\}$

- $q_*(a)$: action value of action a
- $\mathbb{E}$: Expectation
- R_t : Reward
- $A_t = a$: Action = a
- $\forall \; a \in \{1, 2, \ldots, k\}$: for each possible action

$$q_*(a) = \sum_r p(r \mid a)\, r$$

- $q_*(a)$: action value
- $p(r \mid a)$: probability of observing that reward
- r : possible reward

Maximize this :- $\boxed{\underset{a}{\operatorname{argmax}}\; q_*(a)}$

↳ Find a.

IMPORTANT

MEETINGS / APPOINTMENTS

2. Estimating Action Values

sample-average method → Estimate action values

Action Value : $q_*(a)$

$Q_t(a)$:= Sum of rewards when a taken prior to t / no. of times a taken prior to t

$$Q_t(a) = \frac{\sum_{i}^{t-1} R_i}{t-1}$$

Value of action at time t depends on rewards of actions taken before time t.

Greedy action selection → selecting the action that currently has the largest estimated value.

Non-greedy action → sacrifice immediate reward hoping to get more info about other actions

IMPORTANT

3. Exploration vs Exploitation

⇒ Exploration vs Exploitation.

non-greedy greedy

9

10

Incremental Implementation

11

↳ for not having to store millions of data.

12

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right)$$

1

$$= \frac{1}{n}\left(R_n + (n-1)\frac{1}{(n-1)}\sum_{i=1}^{n-1} R_i\right)$$

2

3

$$= \frac{1}{n}\left(R_n + (n-1)\,Q_n\right)$$

4

$$= \frac{1}{n}\left(R_n + nQ_n - Q_n\right)$$

5

$$\boxed{Q_{n+1} = Q_n + \frac{1}{n}\left(R_n - Q_n\right)}$$

6

new estimate = old estimate + step size (Target − old estimate)

$$\boxed{Q_{n+1} = Q_n + \alpha_n\left(R_n - Q_n\right)} \qquad \alpha \in [0, 1]$$

$\alpha \approx \frac{1}{n}$ IMPORTANT

MARCH 2012

WK	M	T	W	T	F	S	S
9				1	2	3	4
10	5	6	7	8	9	10	11
11	12	13	14	15	16	17	18
12	19	20	21	22	23	24	25
13	26	27	28	29	30	31	

MEETINGS / APPOINTMENTS

Non-stationary Bandit Problem } Distribution of Rewards changes with time.

→ Solution: constant step size.

If α_n = Step size → fixed to say 0.1

then recent rewards affect more than older rewards.

$$Q_{n+1} = Q_n + \alpha_n (R_n - Q_n)$$

$$= \alpha_n R_n + (1-\alpha) Q_n$$

$$= \alpha_n R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2}$$

$$+ \cdots + (1-\alpha)^{n-1} \alpha R_1 + (1-\alpha)^n Q_1$$

$$Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i$$

$(1-\alpha)^n Q_1$: Initial action value

$\sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i$: weighted sum of the rewards at all the times

$(1-\alpha)^n Q_1$: contribution of Q_1 decreases exponentially with time.

IMPORTANT

$\sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i$: the older rewards contribute exponentially less to the sum

Exploration vs Exploitation trade off

↓ improve for long-term benefit.

↓ exploits knowledge for short-term benefit

↳ eventually we'll have more accurate estimates of action values.

To choose between explore & exploit - - -

Epsilon-Greedy Action Selection.

~~↳ Greedy Exploit with a small probability $\cdot \varepsilon$~~

↳ Explore with a small probability

↳ Exploit otherwise, i.e., higher probability

$$A_t \leftarrow \begin{cases} \underset{a}{\text{argmax}}\, Q_t(a) & \text{with prob. } 1-\varepsilon \\ a \sim \text{Uniform}(\{a_1, a_2, \cdots, a_k\}) & \text{with prob. } \varepsilon \end{cases}$$

$\underset{a}{\text{argmax}}\, Q_t(a)$:= choose a with max Q_t } Greedy

$a \sim \text{Uniform}(\{a_1, a_2, \cdots, a_k\})$:= choose randomly bet. $a_1, a_2, \cdots, a_k$ → Explore

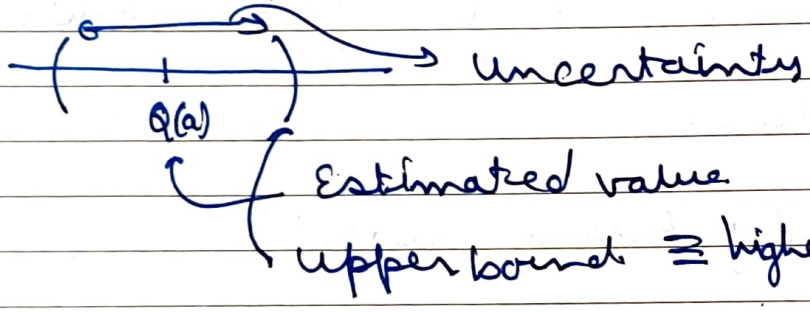
IMPORTANT

Optimistic Initial Values

→ Initially assume reward to be 1. or even more.

Upper-Confidence Bound (UCB)

↳ uses uncertainty in estimates to drive exploration.



↳ choose the action with highest upper bound. (highest possible reward)

$$A_t := \operatorname{argmax}\left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- $Q_t(a)$: action that has (highest estimated value.) ≡ exploitation.
- $c\sqrt{\frac{\ln t}{N_t(a)}}$: UCB exploration term

C → user defined: controls amt. of exploration.
≡ exploration. Eg: C = 2

MEETINGS / APPOINTMENTS

$t \equiv$ time steps ; $N_t(a)$ = times action a is taken

→ initially UCB explores more
UCB's exploration reduces over time.

→ ε-greedy always explores with ε% probability

Contextual Bandits :—

Repeatedly : 1. Observe features x
2. Choose action $a \in A$
3. Observe reward r

Goal : Maximize reward.

MARCH 2012

WK	M	T	W	T	F	S	S
9				1	2	3	4
10	5	6	7	8	9	10	11
11	12	13	14	15	16	17	18
12	19	20	21	22	23	24	25
13	26	27	28	29	30	31	

MODULE: 02

MEETINGS / APPOINTMENTS:

Markov Decision Process (MDP)

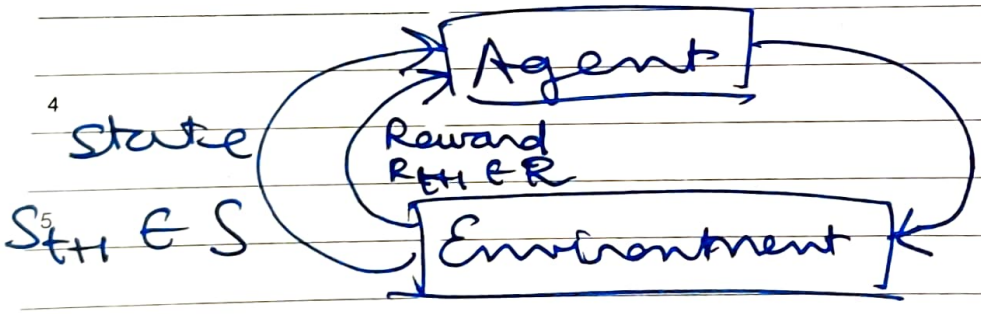
↳ takes into account that **different situations call for different actions.**

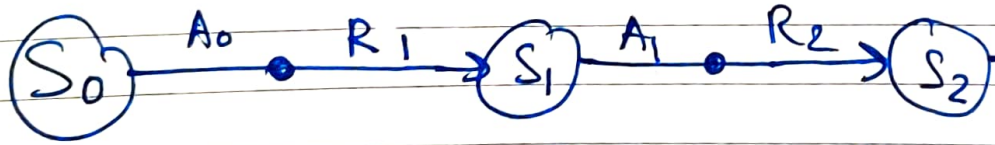
↳ K-armed bandit does not

↳ takes into account **long term impact of our decisions**

↳ K-armed bandit : immediate rewards.

State ≡ Situation. in which one action is taken.



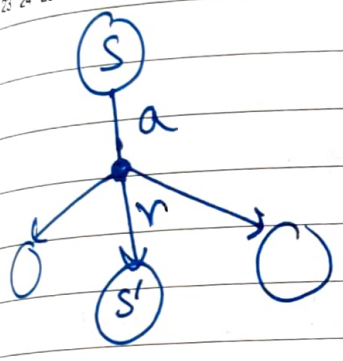


Actions influence [Immediate Rewards] & [Future States]

⇓

Future Rewards

IMPORTANT



Dynamic fun. p

$$p(s', r \mid s, a)$$

$$p: S \times R \times S \times A \to [0, 1] \quad 9$$

$$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1, \forall s \in S \quad 10$$
$$a \in A(s)$$

(Future state & reward) depend on (Present state & action)] Markov Property 12

Present state is sufficient. Older states are not that imp 1 2

Example: Task: The goal of the robot is to pick and place objects. 3 4

State: latest readings of joint angles & velocities 5

Action: amt. of voltage applied to each motor. 6

Reward: +100 when an object is successfully placed.

−1 for each unit of energy consumed.

MEETINGS / APPOINTMENTS

2. Goal of RL

Goal of an agent

$G_t := R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

return. Total future reward ⇒ maximize.

Dynamics of the MDP can be stochastic.

$\mathbb{E}[G_t] = \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T]$

$\mathbb{E}$ ↑ expected.

R_T ↑ interaction ends

Maximize expected ~~ret~~ return

Episodic Tasks

→ Break into episodes

→ Each episode begins independently of how the previous one ended.

→ Every episode has a final state: Terminal state.

After terminal state, each episode is reset.

IMPORTANT

MEETINGS / APPOINTMENTS

Reward Hypothesis.

1. Give a man a fish & he'll eat for a day.
   PROGRAMMING (GOFAI)

2. Teach a man to fish & he'll eat for a lifetime.
   SUPERVISED LEARNING (L&D)

3. Give a man a taste for fish and he'll figure out how to get fish, even if the details change.
   OPTIMIZATION (RL)

⇒ Using RL to reposition solar energy
   ↳ Moving the motors consumes energy
   ↳ Sun shining directly brings in energy

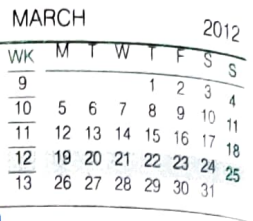
common currency or unit of cost

Goal Rewarding .... **Action-Penalty.**

↳ Goal : 1 ↳ Not Goal : −1

↳ Otherwise : 0 ↳ ~~Otherwise~~ Goal : 0

IMPORTANT



3. Continuing Tasks

Continuing Tasks vs Episodic Tasks

Continuing Tasks	Episodic Tasks
→ Interaction goes on continuously.	→ Interaction breaks naturally into episodes
→ No terminal state.	→ Each episode ends with a terminal state.
$G_t := R_{t+1} + R_{t+2} + \cdots \infty$?	→ Episodes are independent
↳ need to modify this to make finite	$G_t := R_{t+1} + R_{t+2} + \cdots + R_T$

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{k-1} R_{t+k} + \cdots$$

$0 \leq \gamma < 1$: Discount

immediate rewards contribute more.

25 Sunday

WK 12 DAY 085-281

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Rightarrow \text{finite.}$$

$$G_t \leq \sum_{k=0}^{\infty} \gamma^k R_{max} = R_{max} \sum \gamma^k = \frac{R_{max}}{1-\gamma}$$

$$G_t \leq \frac{R_{max}}{1-\gamma} \Rightarrow \text{finite.}$$

IMPORTANT

$\gamma = 0 \Rightarrow$ Short-sighted agent

$\gamma \to 1 \Rightarrow$ far-sighted agent

$$G_t = R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \dots)$$

$$\boxed{G_t = R_{t+1} + \gamma G_{t+1}}$$ Return can be defined recursively.

Specifying Policies

Policies : distribution over actions for each possible state.

Deterministic Policy

↳ Policy maps each state to a single action.

$$\pi(s) = a$$

↳ agent can select same action in multiple states and some actions may be never selected

↳ Table representation.

IMPORTANT

MEETINGS / APPOINTMENTS

Stochastic policy

Policy assigns probabilities to each action in each state.

$\pi(a|s)$: probability of selecting an action a in a state s.

$$\sum_{a \in A(s)} \pi(a|s) = 1 \quad \& \quad \pi(a|s) \geq 0$$

State has all the info required for decision making
↳ action depends only on current state & nothing else.

IMPORTANT

value of a state, under a policy π, is the expected return from that state, if we act according to π. ↳ this is a discounted sum of future rewards.

Delayed Reward.

MODULE : 03
MEETINGS / APPOINTMENTS

1. Value functions

$$v(s) := \mathbb{E}\left[G_t \mid S_t = s\right] \quad ; \quad G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

State value func. is the future reward an agent can expect to receive, starting from a particular state.

↳ Expected return from a given state

$$v_\pi(s) := \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$ ↳ state value func. w.r.t. policy π

$$q_\pi(s, a) := \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

Action value func. q_π of a state is the expected return if the agent selects action a and then follows policy π

Value functions predict rewards into the future.

→ State-value func. represent the expected return from a given state under a specific policy.

↳ Action-value func. represent the expected return from a given state after taking a specific action, later following a specific policy

IMPORTANT

MEETINGS / APPOINTMENTS

Bellman Equation.

Relate the value of the current state to the value of the future state, without waiting to observe all the future rewards.

Bellman equation for state-value function

$$v_\pi(s) := \mathbb{E}_\pi[G_t \mid S_t = s] \quad \} \text{ state value function}$$

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

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

- Defines a relationship between the value of a state: $v_\pi(s)$ & the value of its possible successor states: $v_\pi(s')$

Bellman equation for action-value function

$$q_\pi(s,a) := \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma \cdot v_\pi(s')\right]$$

- In this case the eqn does not begin with the policy selecting an action: $\sum_a \pi(a|s)$

IMPORTANT

$$q_\pi(s,a) = \sum_{s'} \sum_{r} p(s', r \mid s, a) \left[r + \gamma \, q_\pi(s', a') \right]$$

there is also a wt. of probability under π of selecting a' in the state s'

$$q_\pi(s,a) = \sum_{s'} \sum_{r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') \, q_\pi(s', a') \right]$$

→ Use the Bellman eq^n to compute value func. by writing a system of linear equations.

→ We can only solve small MDPs directly, but Bellman eqns will factor into the solutions we see later for large MDPs.

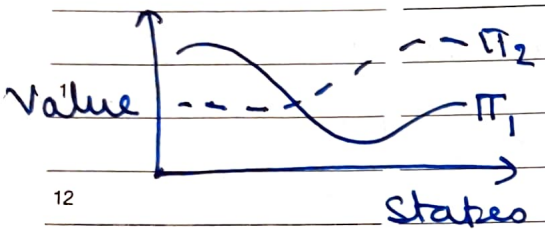
p :— Dynamic func. ⇒ May be stochastic.

s' :— all immediate next ~~step~~ states possible.

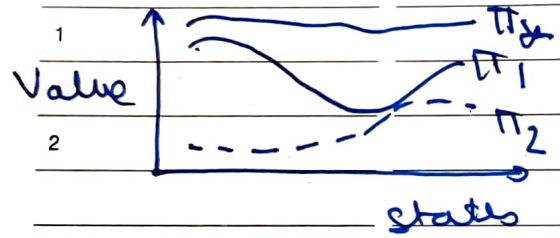
MEETINGS / APPOINTMENTS

3 - Optimal Policies

The policy specifies how an agent behaves.
Given this way of behaving, we then aim to find the value function.



In some states π_1 is better & in some π_2 is better.

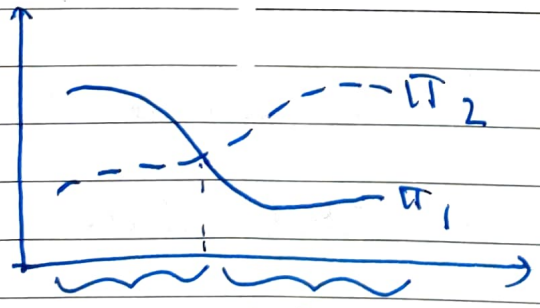


$\pi_1 \geq \pi_2$

π_1 is at least as good as π_2 or better than π_2 for every state.

An optimal policy π_* is as good as or better than all the other policies

$\Rightarrow$ π_* will have highest value in all states



$\pi^* \equiv \pi_3$ a third policy which is the policy with the highest value in the current state.

$\pi_3(s) = \pi_2(s)$

$\pi_3(s) = \pi_1(s)$

IMPORTANT

MEETINGS / APPOINTMENTS

~~#~~ Thus by this we can show that

There must always exist at least one optimal deterministic policy.

Very simple MDP → Few Deterministic Policies → Brute Force Search

A General MDP → Enormous no. of Det. Policies = A^S → ✗ Brute force search impossible

Exponential no. of possible policies makes searching for the optimal policy by brute force impossible

Bellman Optimality Equations

Optimal Value Functions

→ Bellman optimality Equations.

$\pi_1 \geq \pi_2$ iff $v_{\pi_1}(s) \geq v_{\pi_2}(s) \quad \forall s \in S$

$v_{\pi_*}(s) := \mathbb{E}\left[G_t \mid S_t = s\right] = \max_{\pi} v_{\pi}(s) \quad \forall s \in S$

All optimal policies have the same optimal-state value function } — v_*

$q_{\pi_*}(s,a) = \max_{\pi} q_{\pi}(s,a) \quad \forall s \in S \ \& \ a \in A$

↳ max. action value function } — q_*

Bellman eqn for state-value fun. —

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma v_{\pi}(s')\right]$$

By substituting the optimal policy π_* :—

$$v_*(s) = \sum_a \pi_*(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma v_*(s')\right]$$

MEETINGS / APPOINTMENTS

Now π_* will choose the action that achieves the highest value, with probability 1 and will assign probability 0 to all other actions.

$\therefore \sum_a \pi^*(a|s)$ can be replaced by $\max_a$

$$\therefore \boxed{v_*(s) = \max_a \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma v_*(s') \right]}$$

↑ Bellman optimality eq^n for v_*

$$\boxed{q_*(s, a) = \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]}$$

↑ Bellman optimality eq^n for q_*

Note :- for bellman eq^{n}s ⇒ linear system solver will yield results

But bellman optimality eq^{n}s ⇒ "max" function is involved

∴ linear system solver doesn't work.

IMPORTANT

Bellman eqⁿ :- $\pi, p, \gamma \longrightarrow v(s)$

Bellman optimality eqⁿ :- $\pi^*, p, \gamma \xrightarrow{?} v^*(s)$

↳ we don't know π^*

∴ won't work.

∴ $p, \gamma \longrightarrow v^*$

But our final goal is to find π^* not $v^*(s)$

the optimal policy — value of the optimal policy.

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

$$\pi_*(s) = \operatorname{argmax}_a \sum_{s'} \sum_{r} p(s', r | s, a) \left[r + \gamma v_*(s') \right]$$

↳ particular action a that yields the maximum value.

If there are multiple maximizing actions we could define a stochastic optimal policy that chooses between each of them with some probability

IMPORTANT

APRIL 2012

WK	M	T	W	T	F	S	S
13/18	30						
14	2	3	4	5	6	7	1
15	9	10	11	12	13	14	8
16	16	17	18	19	20	21	15
17	23	24	25	26	27	28	22
							29

MEETINGS / APPOINTMENTS

$$\pi_*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} \sum_{r} p(s', r \mid s, a) \left[r + \gamma v_*(s') \right]$$

$$\pi_*(s) = \underset{a}{\operatorname{argmax}}\, q_*(s, a) \Rightarrow \text{easier.}$$

1. Policy Evaluation & Control. MODULE 04

Policy Evaluation ⇓
determining the value func. for a specific policy

Control ⇓
Finding a policy to obtain as much reward as possible
⇓
Finding a policy that maximizes the value function.

Dynamic Programming Algorithms.

→ Use the Bellman eqⁿ to define iterative algorithms for both policy evaluation & control.

Policy Evaluation : $\pi \longrightarrow v_\pi$

Recall : Bellman eqⁿ gives a linear eqⁿ for each π

$$v_\pi(s) = \sum_{a} \pi(a \mid s) \sum_{s'} \sum_{r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

IMPORTANT

MEETINGS / APPOINTMENTS

We used a linear system solver, earlier.

$\pi, p, \gamma \rightarrow$ [Linear System Solver] $\rightarrow V_\pi$

Now we'll use DP $\pi, p, \gamma \rightarrow$ [Dynamic Programming] $\rightarrow V_\pi$

DP: iterative solution methods.

Control is the task of improving a policy.

A policy π_2 is better than a policy π_1, if π_2 is at least equal to π_1 and greater than π_1 in at least 1 state, i.e., $V_{\pi_2} \geq V_{\pi_1} \Rightarrow \pi_2$ better than π_1

When we can improve a policy any more, it means, our current policy is the optimal policy π_*

$\pi, p, \gamma \rightarrow$ [DP] $\rightarrow V_\pi$ Policy Evaluation

$p, \gamma \rightarrow$ [DP] $\rightarrow \pi_*$ Control.

DP needs access to the dynamics function p.

IMPORTANT

Iterative Policy Evaluation.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]$$

$$\Downarrow$$

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma v_k(s')\right]$$

iteratively refine our estimate of the value func. This will produce a sequence of better & better approximations to the value func.

If we reach $v_{k+1} \approx v_k$, then this is the v_π

For any v_0 $\quad v_k \xrightarrow[k \to \infty]{} v_\pi$

⇒ Array of size equal to no. of states.

Maintain ~~to~~ 2 such arrays V, V' ($\Downarrow$ V_k, $\Downarrow$ V_{k+1})

Input π, the policy to be evaluated.

$V \leftarrow \vec{0}$, $V' \leftarrow \vec{0}$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in S$

$V'(s) \leftarrow \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

$\Delta \leftarrow \max\left(\Delta, |V'(s) - V(s)|\right)$

$V \leftarrow V'$

until $\Delta < \theta$ (a small +ve no.

IMPORTANT Output $V \approx v_\pi$

Policy Improvement.

$$\pi_*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} \sum_{r} p(s', r \mid s, a) \left[r + \gamma v_*(s') \right]$$

↑ greedy action

$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} \sum_{r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

$$q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s)) \quad \forall s \in S \Rightarrow \pi' \geq \pi$$

If we take action a, according to π' and then follow policy π.

We take the action under π

π' is better than π

Policy Improvement Theorem

$q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s)) \; \forall s \in S \rightarrow \pi' \geq \pi$

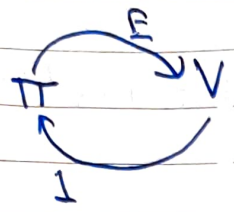
$q_\pi(s, \pi'(s)) > q_\pi(s, \pi(s))$ for at least one $s \in S \rightarrow \pi' > \pi$

The greedified policy is a strict improvement unless the original policy was already optimal.

IMPORTANT

MEETINGS / APPOINTMENTS

Policy Iteration.



E: Evaluation I: improvement

Policy iteration algo to find the optimal policy

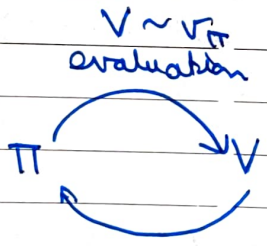
E : Evaluation (Iterative Policy evaluation)
I : Improvement (Greedy policy improvement)

$$\pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} V_{\pi_2} \xrightarrow{I} \pi_3 \cdots\cdots \xrightarrow{I} \pi_*$$

$\pi_2 \geq \pi_1$ $\pi_3 \geq \pi_2$

Sequence of better & better policy, unless the policy was already optimal.

Note : $\pi_2, \pi_3, \ldots, \pi_*$ } → Finite no. of deterministic policies.



If $\pi_{k+1} = \pi_k$
then $\pi_k = \pi_*$: optimal policy
policy is greedy w.r.t. its own value function.

We alternate between evaluating the current policy and greedifying to improve the policy

IMPORTANT

Pseudo code.

1. Initialization.

$V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily $\forall\ s \in S$

2. Policy Evaluation.

Loop :

$\Delta \leftarrow 0$

Loop for each $s \in S$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s', r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (small +ve no. determining accuracy of estimation)

3. Policy Improvement.

policy-stable $\leftarrow$ true

For each $s \in S$:

old-action $\leftarrow \pi(s)$

$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$

If old-action $\neq \pi(s)$, then policy-stable $\leftarrow$ false

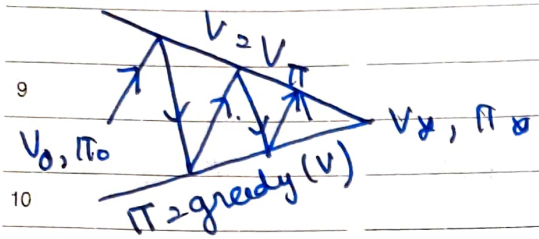
If policy-stable, then stop & return $V \approx v_*$ and $\pi \approx \pi_*$;

else go to 2.

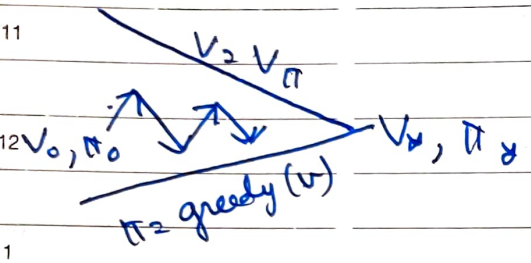
IMPORTANT

MEETINGS / APPOINTMENTS

8. **Flexibility of Policy Iteration.**



The policy iteration algo runs each step all the way to completion.



Each evaluation step, brings our estimate a little closer to the value of the current policy, but not all the way.

Each policy improvement step, makes our policy a little more greedy, but not totally greedy

⇒ **Generalized Policy Iteration algorithms**

1) Value Iteration.

→ We sweep over all the states & greedify w.r.t. the current value function.
But, we do not run policy evaluation to completion.

Systematic sweep through the states } Synchronous DP.

Update the value of states in any order } Asynchronous DP

IMPORTANT

Efficiency of DP

Monte Carlo Sampling alternative method for learning a value function.

↳ **Iterative Policy evaluation** is the DP sol^n to prediction of policy evaluation problem.

↳ **Monte Carlo**: sample based alternative.

The value of each state can be treated as a totally independent estimation problem.

value: $v_\pi(s) := \mathbb{E}_\pi[G_t | S_t = s]$ } Expected return from a given state.

Monte Carlo method:
- → Gather a large no. of returns under it
- → Take their average.
- → This will eventually converge to state value.

This could deal with a lot of randomness.

Bootstrapping: we can use the other value estimates, we have already worked so hard to compute. Using the value estimates of successor states to improve our current value estimate.

↳ Much more efficient than **Monte Carlo method**

estimates each value independently.

IMPORTANT

Brute force search as an alternative to Policy iteration to find optimal policy.

→ Evaluates every possible deterministic policy, one at a time.
→ Then pick the one with highest value.

There are finite no. of deterministic policy & there always exists an optimal deterministic policy.

Total no. of deterministic policies is exponential in the no. of states : $|A|^{|S|}$
↳ Brute force search can take a very long time

Policy iteration takes polynomial time in $|S|$ & $|A|$

Therefore,

1) DP using bootstraping requires much less computation than Monte Carlo

2) DP for policy iteration is exponentially faster than the brute force search.

DP is way better & efficient.

IMPORTANT

MEETINGS / APPOINTMENTS

ADP: Approximate Dynamic Programming.

Dynamic Programming - Summary.

1. Policy Evaluation (Prediction)

$$v_{k+1}(s) := \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_k(s')\right]$$

2. Policy Improvement Theorem.

3. Policy Iteration. $\Rightarrow v^*, \pi^*$

↳ Initialization

↳ Policy Evaluation : $v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_k(s')\right]$

↳ Policy Improvement : $\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$

4. Value Iteration $\Rightarrow v^*, \pi^*$

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v(s')\right]$$

IMPORTANT