1 MARKOV DECISION PROCESSES

1.1 Introduction

- In reinforcement learning an AI learns how to optimally interact in a real-time environment using time delayed labels. called rewards as a signal

- The markov decision process is a mathematical framework for R.L using states, and actions. and rewards
  ↳ Through interacting w/ the environment. an AI will learn a policy which will return an action for a given state of highest reward.

1.2 The Bellman Equation

- Important Bellman Concepts

  • STATE: numeric rep of what the agent is observing @ particular point of time in environment
  Eg: Raw pixels on a screen (in a game environ)

  • ACTION: Input, agent provides to environ: calc by applying policy to current state.
  Eg: combination of control buttons being pressed @ given time.
  : Analog i/p from joystick

  • REWARD: If Feedback signal: reflect how well the agent is performing.
  Eg: coins collected, enemies skilled ...

  • In R.L, given current state we in, choose optimal action which will max the long-term expected reward provided by environ.

  • Dr. Richard Bellman: Father of dynamic programming - class of algos simplify complex probs by breaking into sub probs and solving recursively

- What do Bellman Eq answer ?

- Given state one is in, assuming best possible action taken @ each step what long term reward can one expect ?
- What is the VALUE of the STATE ?

∴ The bellman. Eq helps eval expected reward relative to adv or dis adv of each state.

- The equation

- $V(s) = \max_a(R(s,a) + \gamma V(s'))$

  $V(s)$: Value of a given state

  $R(s,a)$: Reward of optimal action a in state s

  γ : Discount factor (diminishes reward overtime)

  $V(s')$: Value @ the next state.

Value of given state = max action (of all action avail in the state, pick action that max value) . Reward of optimal action in state s to which we add to that the value of the next state times γ . Here's where the recursive behaviour comes about, this continues until terminal state is arrived

- The above equation would entail us exploring every possible action to arrive @ best long term reward . This type of brute force techniques might be obsolete w/ tasks that have too many moves to account for.

- Example :

  → if $\gamma = 1$ the V=1 @ all steps and agent will move around aimlessly.

  → γ : hyperparam

  → successful values b/w 0.9 & 0.99

  → small γ short-term thinking

  → large γ long-term thinking

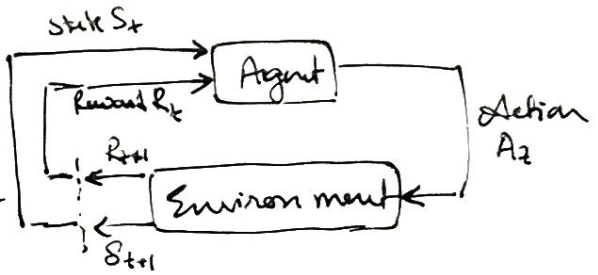
V=0.81 0+0.9(0.9)	V=0.9 0+0.9(1)	V=1 1+0.9(1)	+1
V=0.73 0+0.9(0.81)		V=0.9	-1
V=0.66 0+0.9(0.73)	V=0.73	V=81	V=0.73

Eg $\gamma = 0.9$

$V(s) = \max_a(R(s,a) + \gamma V(s'))$

– The rubic's cube

- State $S = \{r, b, g, o, w, y\}$ $6 \times (3 \times 3)$
- $\mathcal{A}(s)$: avail actions to agent @ current state s
- Time t assumed to be discrete. $\in \mathbb{N}\{1, 2, \ldots\}$
- Seq of steps $S_1\ A_1\ S_2\ A_2\ \ldots$
  - ↳ The trajectory is describes what's happening but no info on the goal of agent.



- Feedback should be into. to do help agent understand connection b/w its current actions & achievement of goal. In RL this takes the form of a scalar signal called Reward.
  - ↳ Seq: $S_1\ A_1\ R_2\ S_2 A_2\ R_3\ \ldots$
  - ↳ Expected tot reward (+1 state following action solved cube −1 otherwise)

$$\sum_{t=1}^{T} R_{t+1} = -(T-1) + 1 = -T + 2$$

T: no. of moves agent rep to reach solution

Lower T higher the reward : push the agent to do better.

- The common principle underlying R.L technique is the reward hypothesis
  - ↳ Goals and purpose can be well thought as maximization of expected value of cumulative sum of rewards

– Markov Decision Process (MDPs)

- Most of systems messier than rubic's cube: evolution of a system might be stochastic instead of deterministic
  - ↳ A specific pair of (state, action) may lead to diff rewards and give general formula
    each of its own prob. We need to take all possibilities into consideration

- R.L systems is approached as a Stochastic process.
  - ↳ Deterministic models: o/p of model fully determined by param values and initial condi
  - ↳ Stochastic models: process inherent randomness: same set of params and initial condi leads to diff results
- Seq of actions states & rewards: $S_1\ A_1\ R_2\ S_2 \ldots$ } Random variables
- The environment fully known once properly specified.
  - ↳ prob of receiving reward r @ time $t+1$ knowing system history upto that moment:

$$P(R_{t+1} = r \mid A_t = a_t, S_t = s_t, A_{t-1} = a_{t-1} \ldots\ldots S_1 = s_1)$$

for each $t \in \mathbb{N}$, for $s_t, s_{t-1} \ldots s_1 \in S$ & for all $a_i \in A(s_i)$ $i \in \{1 \ldots t\}$

  - ↳ Prob of ending up in state s @ time $t+1$ knowing system history upto that moment:

$$P(S_{t+1} = s \mid A_t = a_t \ldots \ldots S_1 = s_1)$$

for ...

  - ↳ We will always assume rewards are markovian: only dependent of on latest pair of state-action.

$$P(R_{t+1} = r \mid A_t \ldots S_1 = s_1) = P(R_{t+1} = r \mid A_t = a_t, S_t = s_t)$$

& the random variable S_t is also assumed to be markovian

  - ↳ The environment is responsible for behaviour of states } rewards.
- The Agent is fully determined by prob of action a taken at time t; knowing system history upto that moment.

$$P(A_t = a \mid S_t, A_{t-1}, \ldots S_1)$$

  - ↳ Complete specification of agent behaviour is called Policy.

- Policy

- **Notations**

→ history of system; seq of states & actions observed so far.

$$h_t := (s_1, a_1, s_2, a_2 \dots a_{t-1}, s_t)$$

$s \in S$ $\quad a_i \in A(s_i)$ for all $i \in \{1 \dots t\}$

H_t ⇒ history as random variable

→ Assuming action space doesn't depend on current state of the system. (makes sense, actions are always taken with intent to max reward @ next state)

$$H_1 := S$$

$$H_t := S \times A \times \dots \times S = S \times \prod_{i=1}^{t-1} (A \times S) \quad \text{if } t \in \{2, 3 \dots\}$$

$$H_1 := S$$

$$H_t := H_{t-1} \times A \times S \quad \text{if } t \in \{2, 3 \dots\}$$ } Recursion

∴ history of system upto time t: $h_t = (s_1, a_1, \dots s_t)$

- **Decision Rules** { d_t is the decision rule as it models how agent reacts acc to current simulation experience }

We know $P(A_t = a_t \mid S_t = s_t, A_{t-1} = a_{t-1} \dots S_1 = s_1)$

→ Using new history notation $P(A_t = a_t \mid H_t = h_t)$

which is prob of agent performing action @ time t ↑ knowing what has happened so far in terms of state & actions.

→ formalize it as function d_t from H_t to $P(A)$, the space of prob distribution over A:

$$d_t : H_t \longrightarrow P(A)$$

→ We shall denote by $b_{d_t}(a \mid h_t)$ or $b_{d_t(h_t)}(a)$ the prob of taking action a if our system history upto time $t = h_t$ ie

$$P(A_t = a_t \mid H_t = h_t) = b_{d_t(h_t)}(a_t)$$

- **Policy**
  - → It is the recipe underlying the behaviours of our agent : what it's programmed to do under certain circumstances.
  - → Decision rules provide a quick way to formally define it.
    - ↳ policy π : seq of decision rules.

      $\pi = (d_1, d_2, \dots)$ s.t. $d_i : H_i \to P(A)$ for each $i \in \{1, 2, \dots\}$
  - → Reward hypothesis: The goal is the maximization of expected value of the cumulative sum of a received scalar signal.

- **Returns**
  - → In R.L every action is followed by reward. (+ve / -ve). To measure overall performance of a policy $\leq$ rewards $\Rightarrow$ called Return
  - → If R_{t+1} reward after action A_t, cumulative return at t, G_t

    $$G_t := \sum_{k=t+1}^{T} R_k$$

    T: random variable denoting end of time of our simulation
  - → Sometimes it's the case where $T = +\infty$: exploration game.
    - ↳ The issue can be solved by intro discount rate $\gamma \in [0,1)$

    Discounted Cumulative return $G_t := \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$

    $\gamma \in [0,1)$ allows 1 wb to closer rewards and ensures overall reward is finite if rewards are bounded.

    $$|G_t| \leq \sum_{k=t+1}^{+\infty} |\gamma^{k-t-1} R_k| = \sum_{k=t+1}^{+\infty} \gamma^{k-t-1} |R_k| \leq M \sum_{k=t+1}^{+\infty} \gamma^{k-t-1} = \frac{M}{1-\gamma}$$

    assuming there exists $M > 0$ such that $|R_t| \leq M \ \forall t$

- **State - value function**
  - → let $\pi = (d_1, d_2 \dots)$ be policy followed by agent
    - G_t = discounted cumulative return from time t.
    - S_1 = initial state of environment.

    Then expected value of cumulative sum of received scalar signal = $V_\pi(s_0) := \mathbb{E}_\pi[G_1 \mid S_1 = s_1]$ also called S.V.F of policy π

- **Expected Reward**

→ Expected value of 1st reward, R_2 under policy π

$$\mathbb{E}_\pi [R_2 | S_1 = s_1]$$

↳ Expectation of R.V = sum of all possible values otd by its probability

↳ $= \sum_{r \in R} r \, P(R_2 = r | S_1 = s_1)$

R is a set of possible rewards.

↳ Using law of total prob this can be decomposed w.r.t set of possible actions available to agent (A_t)

$= \sum_{a \in A} P(R_2 = r | A_1 = a, S_1 = s_1) \, P(A_1 = a | S_1 = s_1)$

↳ But $P(A_1 = a | S_1 = s_1)$ is fully determined by agent's 1st decision rule $d_1 := b_{d_1(s_1)}(a)$

$$\therefore \mathbb{E}_\pi [R_2 | S_1 = s_1] = \sum_{a \in A} b_{d_1(s_1)}(a) \left(\sum_{r \in R} r \, P(R_2 = r | A_1 = a, S_1 = s_1) \right)$$

$$= \sum_{a \in A} b_{d_1(s_1)}(a) \cdot \mathbb{E}(R_2 | A_1 = a, S_1 = s_1) \quad \text{--- (1)}$$

($b_{d_1(s_1)}(a)$: agent; $\mathbb{E}(R_2 | A_1 = a, S_1 = s_1)$: environment)

The policy determines the prob to choose an action, while the environment is responsible for the expected return awarding the agent if it were to choose a specific corresponding action.

- **Expected Return** : to derive formula for expected discounted return.

→ All expectation is linear ie $\mathbb{E}[\alpha A + \beta B] = \alpha \mathbb{E}(A) + \beta \mathbb{E}(B)$

A & B are random variables, α & β real nos.

→ Then

$$v_\pi^1(s_1) = \mathbb{E}_\pi [G_1 | S_1 = s_1]$$

$$= \mathbb{E}_\pi [R_2 + \gamma G_2 | S_1 = s_1]$$

$$= \mathbb{E}_\pi [R_2 | S_1 = s_1] + \gamma \mathbb{E}_\pi [\gamma G_2 | S_1 = s_1] \quad \text{--- (2)}$$

(first term: already derived formula; second term: □)

→ Using law of tot prob and def of expected value

$$\square = \sum_{g} g \sum_{a \in A} P(G_2 = g \mid A_1 = a, S_1 = s_1)\, b_{d_1(s_1)}(a)$$

→ Repeating this again: this time w.r.t S_2:

$$P(G_2 = g \mid A_1 = a, S_1 = s_1) = \sum_{s \in S} P(G_2 = g \mid S_2 = s, A_1 = a, S_1 = s_1) \cdot P(S_2 = s \mid A_1 = a, S_1 = s_1)$$

$$= \sum_{s \in S} P(G_2 = g \mid H_2 = (s_1, a, s)) \cdot P(\text{"}) $$

→ Putting everything together, we get:

$$\square = \sum_{g} g \sum_{a \in A} \sum_{s \in S} P(G_2 = g \mid H_2 = (s_1, a_1, s)) \cdot b_{d_1(s_1)}(a) \cdot P(\text{"})$$

$$= \sum_{a \in A} \sum_{s \in S} \left[\sum_{g} g\, P(G_2 = g \mid H_2 = (s_1, a_1, s)) \right] \cdot b_{d_1(s_1)} P(\text{"})$$

$$= \sum_{a \in A} \sum_{s \in S} E_\pi [G_2 \mid H_2 = (s_1, a_1, s)] \cdot b_{d_1(s_1)} \cdot P(\text{"})$$

The diff b/w this and s.v.f is that we are computing the expected discounted cumulative return starting from $t=2$ instead $t=1$ as well as conditioning on h_2 instead of $s_1 = h_1$

→ Formalizing this by introducing the history-value function for policy π @ time t.

$$v_\pi^t(h_t) := E_\pi [G_t \mid H_t = h_t]$$

↳ Plugging this into equation □

$$\square = \sum_{a \in A} \sum_{s \in S} v_\pi^2((s_1, a, s)) \cdot b_{d_1(s_1)}(a) \cdot P(\text{"}) \quad \text{— (3)}$$

(agent; environment)

From eq (1), (2) & (3) the general formula for expected value-cumulative discounted return under policy π:

$$v_\pi^1(s_1) = \sum_{a \in A} b_{d_1(s_1)}(a) \Big(E[R_2 \mid A_1 = a, S_1 = s_1] + \gamma \sum_{s \in S} v_\pi^2((s_1, a, s)) \cdot P(S_2 = s \mid A_1 = a, S_1 = s_1) \Big) \quad \text{— (4)}$$

This is a generalized version of Bellman's eq in the framework of MDP

- **Bellman Equation**

→ Using techniques from before one can prove the following equation holds for $t \in \{1, 2, \dots\}$

$$v_\pi^t(h_t) = \sum_{a \in A} b_{d_t(h_t)}(a)\Big(E[R_{t+1} \mid A_t = a, H_t = h_t] + \gamma \sum_{s} v_\pi^{t+1}((h_t, a, s)) \cdot P(S_{t+1} = s \mid A_t = a, H_t = h_t)\Big)$$

, The problem here is: to compute v_π^1 we need to know v_π^2 which in turn requires v_π^3 ... no end assuming $T = +\infty$

↳ Things would simplify if we assume that the environ. and the agent are markovian & stationary
ie S_t & R_t are only dependent on A_{t-1} & S_{t-1}
while agent bases its A_t choice on the value of S_{t-1} w/o taking into account current time instant t.
ie $\pi = (d_1, d_2 \dots)$

∴ Eqn (4) becomes

$$v_\pi^1(s_1) = \sum_{a \in A} b_{d_1(s_1)}(a)\Big(E[R_2 \mid A_1 = a, S_1 = s_1] + \gamma \sum_{s \in S} v_\pi^1(s_1) P(S_2 = s \mid A_1 = a, S_1 = s_1)\Big)$$

A policy value function is in fact the expected cumulative discounted reward following that strategy: if $v_2^1(s) \geq v_\pi^1(s)$ for every $s \in S$ we can generally conclude τ is a better policy than π (the environ. by returning actions to take for each state)

π^* is called the optimal policy that maximise expected reward. Among all policies taken, the optimal policy is the one that maxi the amt of reward received or expected to receive. For an MDP there is no end of life time, hence you have to decide the time.
Thus policy is a guide telling which action to take for a given state. It's not a plan but returns the underlying plan of ←

Markov Decision Processes

- MDP is formally used to describe an environment for R.L where the environment is fully observable.

- The MDP

- The markov property states that : future is independent of past ~~once~~ given the present.
  ↳ a state S_t has Markov property iff

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

The state captures all relevant info from history

↳ For markov state S and successor state S', the transition prob is defined by:

$$P_{ss'} = P[S_{t+1} = S' \mid S_t = S]$$

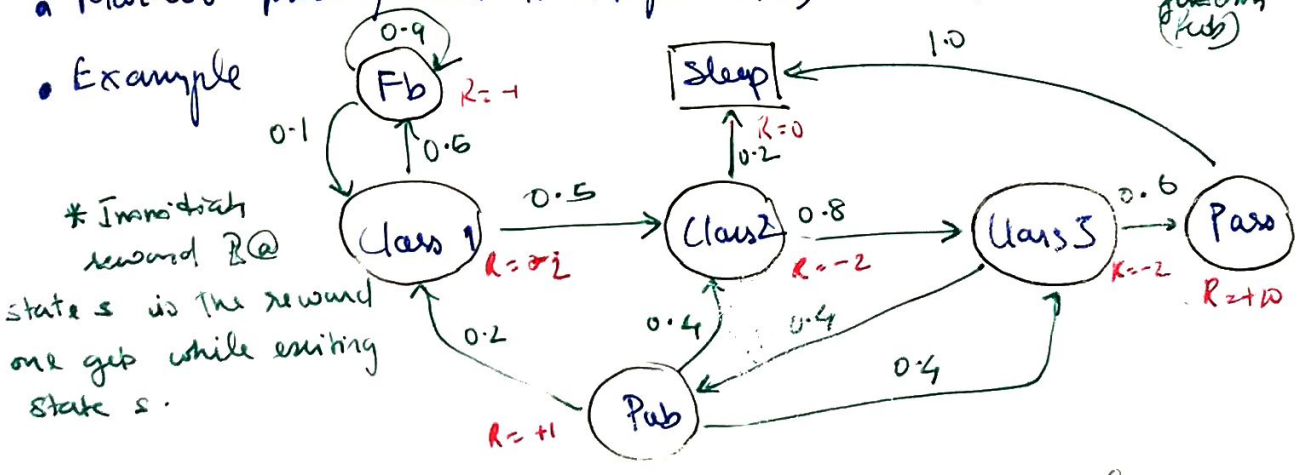
prob distribution over next possible successor state given current state.

Matrix form of all possible transitions, where each row sums to 1

$$P = \text{from} \begin{bmatrix} & \text{to} & \\ P_{11} & \cdots & P_{1n} \\ \vdots & & \vdots \\ P_{n1} & & P_{nn} \end{bmatrix}$$

- Markov Process

- Memory-less, random process ie seq of random states $S_1, S_2 \ldots$ w/ markov property
- Markov process / chain is a tuple (S, P) S: state space P: transition function (Pss')
- Example



* Immediate reward R@ state s is the reward one get while exiting state s.

- Some example sample episodes { seq of states } are:
  - ↳ C1 C2 C3 Pass Sleep
  - ↳ C1 C2 C3 Pub C2 C3 Pub C2 Sleep.
- The transition matrix for the same is

	C1	C2	C3	Pass	Pub	FB	Sleep
C1		0.5				0.5	
C2			0.8				0.2
C3				0.6	0.4		
Pass							1.0
Pub	0.2	0.4	0.4				
FB	0.1					0.9	
Sleep							1.0

- Markov Reward Process

- The MRP is a tuple (S, P, R, γ)
  - ↳ S: a finite set of states
  - ↳ P: state transition prob matrix
    $$P_{ss'} = P[S_{t+1} = s' \mid S_t = s]$$
  - ↳ R: reward function, $R_s = E[R_{t+1} \mid S_t = s]$ } Reward for only 1 step.
  - ↳ γ: discount factor, $\gamma \in [0, 1]$
- Looking @ the MDP ex in page 10, rewards have arbitrarily been added to it: you don't like sitting in class :. −2, but if you pass all you get +10.
- We care about cumulative reward, as the goal is to max the return. :. The return G_t is the total discounted reward from time step t

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} \cdot R_k$$

  - ↳ There is no expectation here cause G_t is just one sample from a set MRP of the rewards we get going through that seq.
- The value function is the long term value of being in a state. So were I to drop you in state S, how much value will you get from there on. What's the total reward you'll get from S onwards

$$V(s) = E[G_t \mid S_t = s]$$ (Env is stochastic hence E)

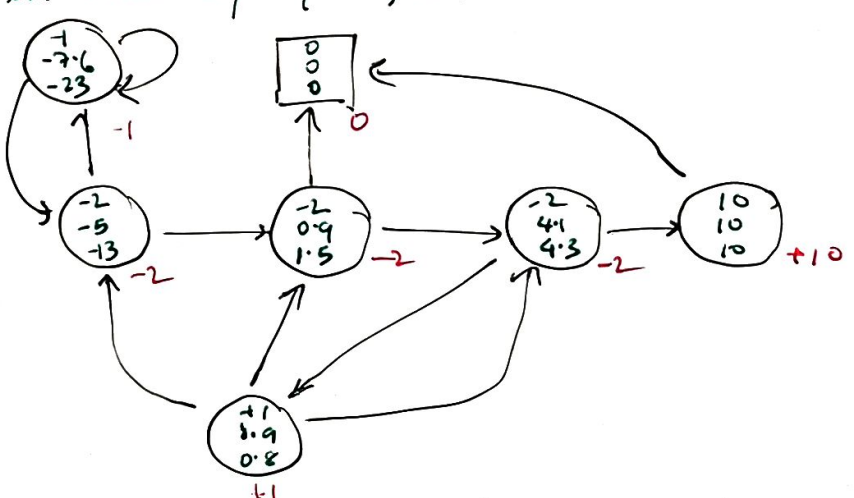
↳ Taking the student-class example: starting from $S_1 = C1$ and $\gamma = 0.5 \Rightarrow G_1 = R_2 + \gamma R_3 + \dots + \gamma^{T-2} R_T$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 \cdot 0.5 - 2 \cdot 0.25 + 10 \cdot 0.125 = -2.25$
C1 FB FB C1 C2 Sleep	$v_1 = -1 - 1 \cdot 0.5 - 1 \cdot 0.25 - 2 \cdot 0.125 = -3.125$
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = \quad = -3.41$

We have 3 start dep samples here, one way to estimate the value of the state would be take a bunch of samples and take the avg. and

∴ The returns of samples are random but the value function is not a random quantity, rather an expectation of these R.V

↳ The state-value w/ $\gamma = 0, 0.9, \& 1$



We observe as γ is increased (long sighted) we get better rewards given a particular state S_t as more wt is given to the future rewards.

- **The Bellman Equation for MRPs**

→ value function decomposed into
  ↳ Immediate reward R_{t+1}
  ↳ Discounted value of successor state $\gamma v(S_{t+1})$

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

$$= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s]$$

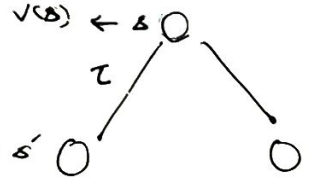
$$= E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s]$$ (Immediate reward + discounted reward from next steps)

$$= E[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= E[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s]$$ (The val. function now = expected immediate reward + discounted value @ next state)

→ $V(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} V(s')$

Expected value $E(x) = \sum x \cdot p(x)$



↳ one step look ahead function with this arg (go ahead 1 step and avg all possible outcomes together which gives value function @ s)

In the future one will notice that building of algos will take into the idea of one step look ahead.

↳ from the diagram in page 12: w/ $\gamma = 1$ & $C3 = 4.3$

using the above formula ($V(s) = R_s + \gamma \sum P_{ss'} V(s')$)

$= -2 + 1(0.6 \times 10 + 0.4 \times 0.8)$

$= 4.3$

→ Matrix representation.

$$V = R + \gamma P V$$

$$\begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} \cdots P_{1n} \\ \vdots \quad \vdots \\ P_{n1} \quad P_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix}$$

The value of state = immediate reward + the transition matrix + the values when we may end up

→ The Bellman Equation is a linear eq:

↳ can be solved directly $V = R + \gamma P V$

$I = (1 - \gamma P)^{-1} R$

↳ computational complexity = $O(n^3)$

↳ Direct solution only possible for small MRPs

↳ For large MRP's → Dynamic Programming → Temporal-difference learning

→ Monte Carlo evaluation

- Markov Decision Process

Until now one really hadn't had agency to take actions or make decisions. So let's introduce decisions.

∴ MDP is MRP with decisions.

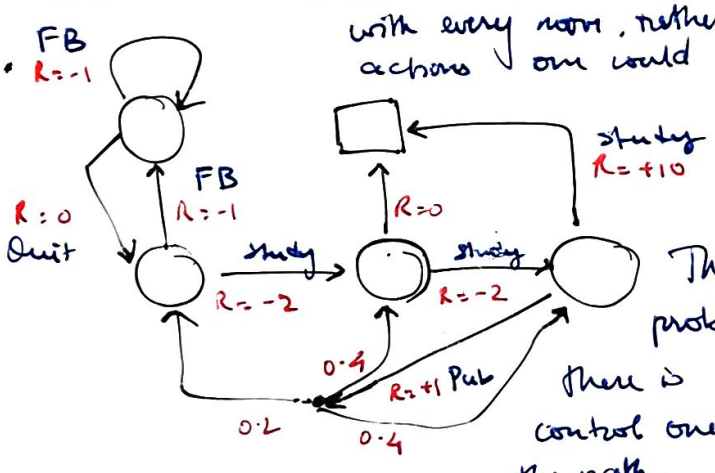
Hence MDP is a tuple (S, A, P, R, γ) where:

- S: finite set of states
- A: finite set of actions
- P: state transition prob matrix

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

- R is a reward function $R^a_s = E[R_{t+1} \mid S_t = s, A_t = a]$
- γ: discount factor $\gamma \in [0, 1]$.

- Example: Student MDP → So here there aren't probs attached with every node, rather there are actions one could decide on.



There are still some probs, but this time there is a lot more control one can exert over the path.

- Policies

→ Distributions over actions given states

$$\pi(a|s) = P[A_t = a \mid S_t = s]$$ i.e. if one is in some state s this distribution gives us the mapping.

→ A policy fully defines the behaviour of an agent

→ MDP policies depend on the current state (not history)

i.e. policies are stationary (time-dependent)

$A_t \sim \pi(\cdot|S_t), \forall t > 0$

* There aren't any rewards in the equation because the state s fully characterizes ones future rewards. The markov property means → fully captures the evolution from state s onwards.

So we are looking for a policy, which given the state one is in you'll want to take actions that'll get you the most future reward. (We don't care about rewards in the past as they are gone, all we care about is the rewards into the future)

→ Connection b/w MDP & MRP is we can always recover MRP from MDP

↳ The state seq $S_1, S_2 \ldots$ is a markov process

↳ The state & reward seq $S_1, R_2, S_2 \ldots$ is a Markov reward process $(S, P^\pi, R^\pi, \gamma)$

where

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

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

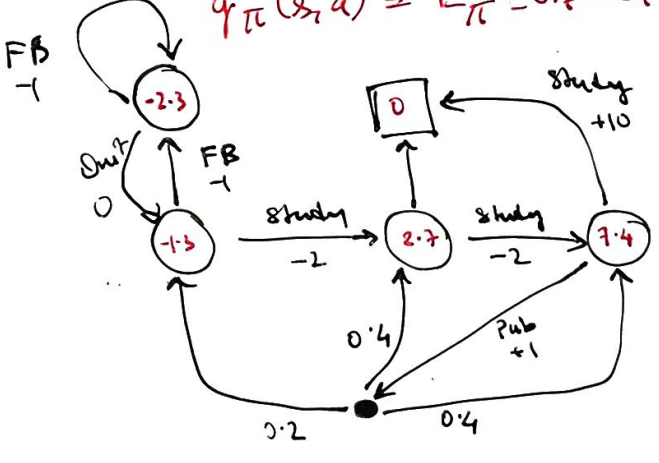
- Value Function

The state-value function $v_\pi(s)$ (value @ state s after having followed policy π) of an MDP is the expected return starting from state s and following policy π

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$ (Expectation when we sample all policies acc to policy π)

→ Action value function $q_\pi(s,a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a]$$



$v_\pi(s)$ for $\pi(a|s) = 0.5$
$\gamma = 1$

- Bellman Equation
  Expectation

→ The state value function can again be decomposed into immediate reward + discounted reward

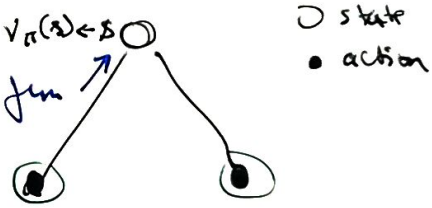
$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

→ Similarly, the action-value function can similarly be decomposed

$$q_\pi(s,a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

↳ Tells us, if I'm in one state and an action is taken then some immediate reward is received for that action and then I look @ when I end up and ask what's the action value of the state I end up in under the action I would pick from that point on.

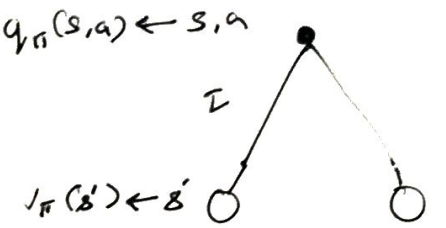
→ Understanding how v & q relate to each other. If we were in a state-val fun here. What it's saying is that we are going to avg over the $q_\pi(s,a) \leftarrow a$ actions we might take. There's some prob we might take this ● ○ action here, the prob of which is defined by the policy - (prob you'll go left, right...). For each of the actions we might take there's a q value. telling how good is it to take $A_t = a$ from $S_t = s$. So what we are doing is a 1 step look ahead saying that the state value, how good is it at the state to look ahead 1 step, look @ action values avg them together. and that tells us the value of being @ $S_t = s$



$$v(s) = \sum_{a \in A} \pi(a|s)\, q_\pi(s,a)$$

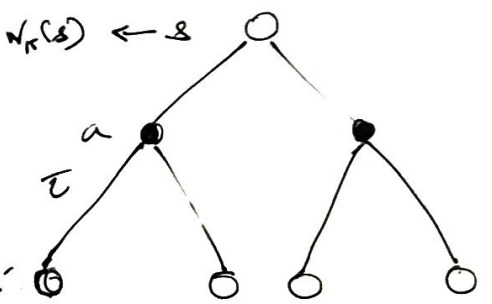
→ Understanding the opp step: starting off taking some action.

Now the root of this tree is a state and we are considering a particular action we take from that state. (I'm in this • state here, how good would it be to go right from that state) • Therefore v_π tells us how good is it to be in a particular state, q_π tells us how good is it to take an action from a given state.



$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$$

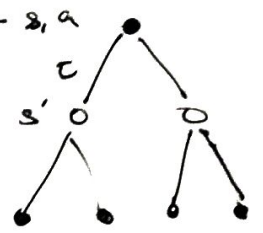
→ Putting stuff together, we get a recursion that helps us understand v in terms of itself → end up solving MDP.



At the roots of the tree, we have the val-function for a particular state - telling us how good is it to be in this state. The way we give understand that is by doing a 2 step look ahead - consider all the actions we might take next, consider all the things the environment might do to us, and for each of the thing the environ would do there would be some success state we might end up in. And we'd want to know how good is it to be in that state and carry on with the policy. If we are averaging over policy, weight each of these ^ arcs by prob the policy will select, and average these ^ ^ arcs by the transition prob that we'll end up getting blown in one direction or another. Avg all together and gives us the value of being @ the root

$$v_\pi(s) = \sum_{} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$$

→ One can do exactly the same thing $q_\pi(s,a) \leftarrow s, a$ with action values (recursive relation) by sticking diagram other way around

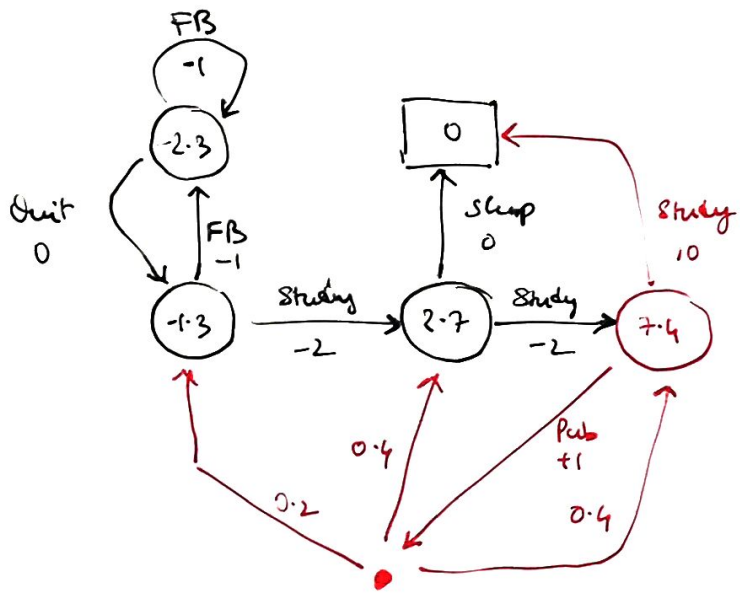


starting from a particular state or action we can look ahead 2 steps, consider which state we might end up and then consider from the state we end up @ which action might we take next and then avg over it

↳ On the previous diagram the state values relate to state values in next step and here the q values relate to the q values in the next step

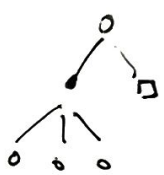
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s')\, q_\pi(s',a')$$

→ Student MDP example



Considering only the red state; class 3.

$\pi(a|s) = 0.5$



$$V(s) = \sum \pi(a|s)\left(R_{ss}^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s')\right)$$

$$= 0.5 \times 10 + 0.5\,(1 + 1\,(0.2 \times -1.3 + 0.4 \times 2.7 + 0.4 \times 7.4)$$

$$= 7.4$$

Here the bellman equation does really hold. This is all well and good but it doesn't tell us the best way to behave.

→ Matrix form of bellman eq

$$v_\pi = R^\pi + \gamma P^\pi v_\pi$$

↳ direct sol → $v_\pi = (1 - \gamma P^\pi)^{-1} R^\pi$

- Optimal Value Function

The main problem would be finding the best behaviour using MDP.

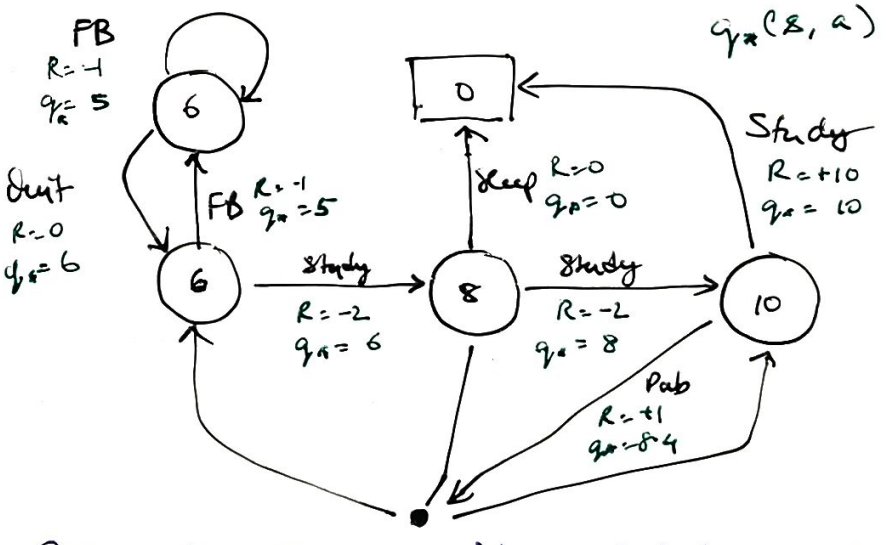
The optimal state-value function $v_*(s)$ is the max value function over all policies.

~~$v_*(s) = \max_\pi v_\pi(s)$~~ $v_*(s) = \max_\pi v_\pi(s)$

The optimal action-value function $q_*(s,a)$ is the max action-value function over all policies

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

→ MDP is solved when we know the optimal value function $q_*(s,a)$ for $\gamma = 1$



→ Optimal policy: we'd want to know what to it for one policy to be better than another policy and define a notion of optimality.

$\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$, $\forall s$] Defining partial ordering of policies

↳ For any MDP :

(i) There exists optimal policy π_* that is better or equal to all other policies $\pi_* \geq \pi \;, \forall \pi$

(ii) All optimal policies achieve the opt-val fun $v_{\pi_*}(s) = v_*(s)$

(iii) " the opt-act-val fun $q_{\pi_*}(s,a) = q_*(s,a)$

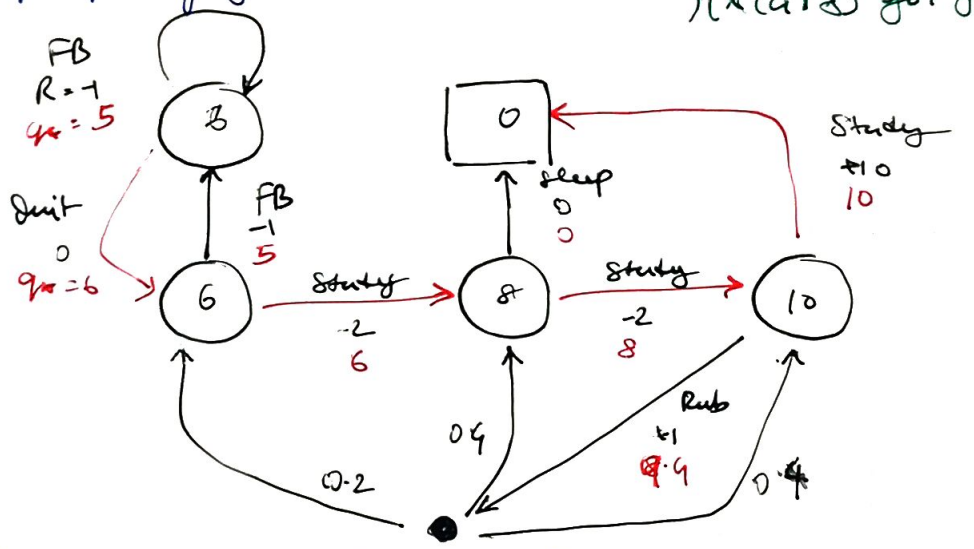
↳ Finding Opt policy : can be found by max over $q_*(s,a)$

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in A}{\operatorname{argmax}}\, q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

* There is always a deterministic opt policy for any MDP
* If we know $q_*(s,a)$ we immediately have opt-policy

→ Opt-policy for student MDP

$\pi_*(a|s)$ for $\gamma = 1$



Red are : optimal policy π^*

Pub are : $= 1 + 1(0.2 * 6 + 0.4 * 8 + 0.4 * 10)$
$= 9.4$

Study are : $10 + 1(0) = 10$

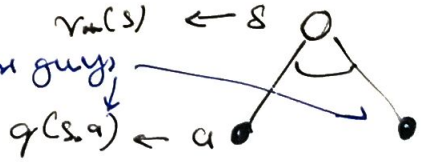
} max = 10

- **Bellman Optimality Equation**

→ Tells us how to solve MDP; how to relate opt-val fun to itself-

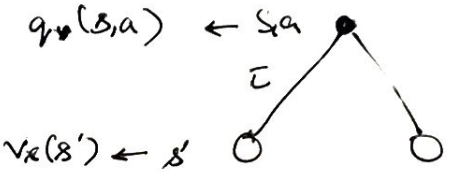
→ Instead of taking Exp val of these guys, we take the max

$v_*(s) = \max_a q_*(s,a)$



→ Optimality eq for Q^*

Now we want to know how good are the red arcs

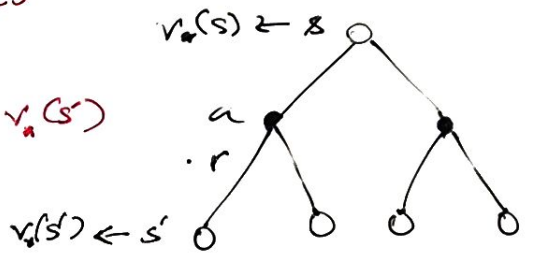


We don't control this part. But each of the state we may end up in would have some opt value

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

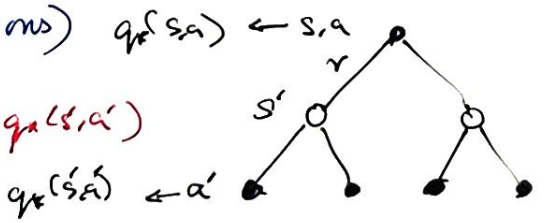
→ 2 step look ahead

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \cdot v_*(s')$$



→ 2 step look ahead (actions)

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s,a')$$



→ Solving the Bellman Opt Eq

↳ The eq: is non-linear now

↳ No closed form solution (in general)

↳ Many iterative sol: val iter, policy iter, Q-learning, Sarsa

- Extensions to MDPs

- Infinite MDPs
  - → countably infinite state and/or action spaces
- Partially observable MDPs
- undiscounted avg reward MDPs