Key Concepts in RL

Agent : Agent takes actions

Environment : World in which agent exists & operates

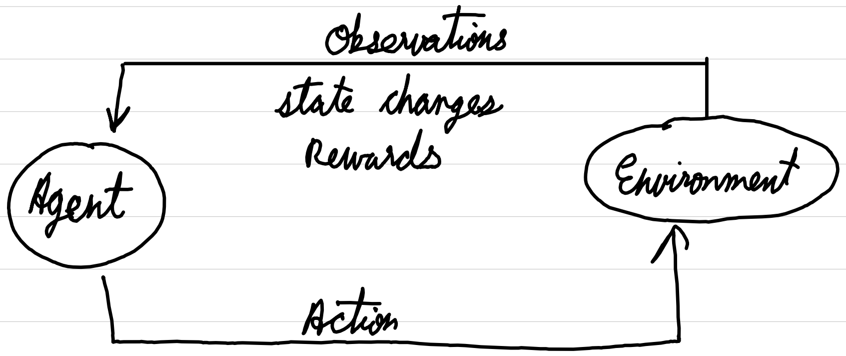
Actions : A move that the agent can make

Action space A : set of possible actions an agent can make in the environment
Action space can be discrete or continuouse

Observations : Input from the environment

state : situation which the agent percieves

Reward : Feedback that measures the success or failure of the agents action



Policy : which action to do in which state (π) [agents policy]

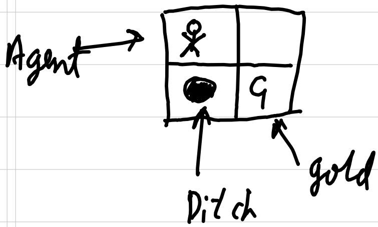
Value function : How good the situation is.
Prediction of future reward

Model : Predicts what environment will do next
eg atari game - prediction of opponent

Return : Total reward obtained by the agent

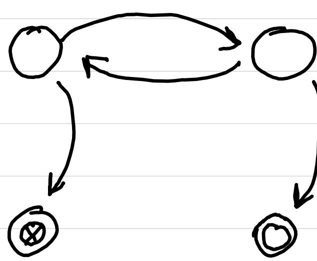
Example of environment

Consider a simple environment



goal of agent is to obtain gold without falling in ditch

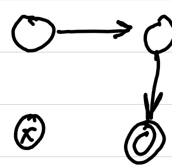
Possible actions



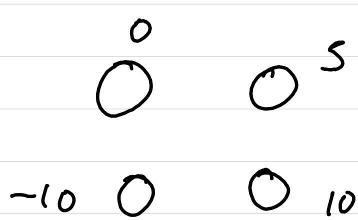
Reward = +10

Reward = −10

Optimal policy



Optimal value function



Total Reward

Agent wants to maximize the reward

At any time t, the agent has a total reward

$$G_t = \sum_{i=t}^{\infty} \gamma^i r_i = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total reward at time t also called G_t

reward returned at time i

disount factor

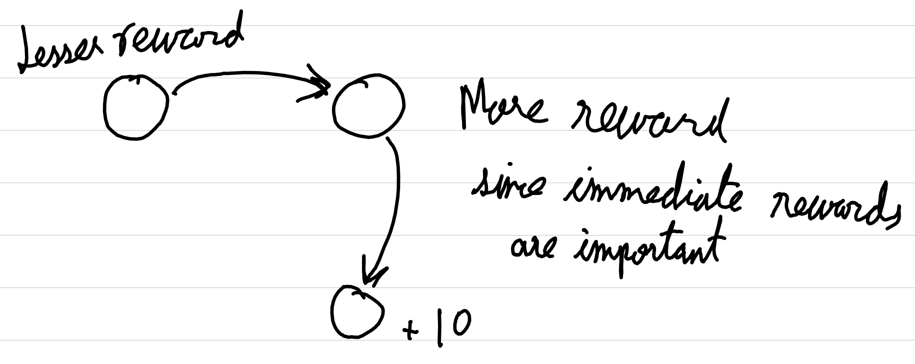
In order to give more importance to immediate rewards, a γ factor is added

$(0 \le \gamma \le 1)$ discounting factor

G_t is the total discounted reward

γ close to 0 leads to myopic evaluation

γ close to 1 leads to far sighted evaluation



Intermediate rewards can be written as

$$R(s, a, s')$$

Current state

action taken

Next state

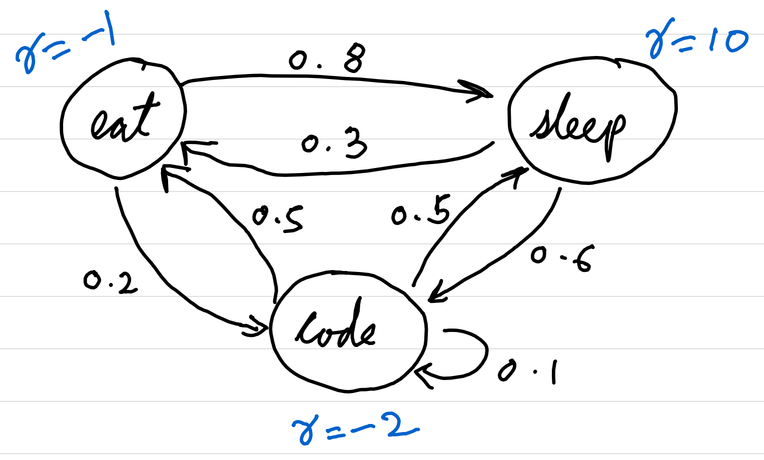
Rewards can be assigned to states (as in Markov reward process)

or to actions (as in Markov decision process)

Total discounted reward is also known as return

Markov Reward Process

Consider a simple Markov chain



We need the agent to spend maximum time in state sleep.

Hence we put the reward in state sleep.

We put negative rewards in other states

In order to prevent the agent from going round and round in intermediate states, negative weights are used.

Now we can perform a random walk and calculate the reward

eg eat - eat - sleep $= -2 + -2 + 10 = 6$
eat - sleep - code - code $= -2 + 10 + -2 = 4$

A Markov reward process is a markov chain with values.

Tuple $\rightarrow \langle S, P, R, \gamma \rangle$

$S \rightarrow$ states

$P \rightarrow$ state transition p

$\gamma \rightarrow$ discount factor

$R \rightarrow$ Rewards

In markov reward process, the model is entirely random walk.

We are just interested in the rewards (V) of the state

there is only P, no π

Used to compute the value function.

Value function in MRP

Since MRP is a random walk, we dont have any control over where the agent will go.

Where agent will go is decided by transition probability P.

$P(s|s')$ define the likelyhood that the agent goes from state s to s'

$$V(s) = R(s) + \gamma \sum P(s'|s) V(s')$$

Goodness of state = Reward obtained + Weighted prob sum of Goodness of next possible states

V depends upon V itself. So how to calculate it? It can be done by two methods

1. Eigenvalues → costly
2. D.P. → Better

Dynamic Programming

Dynamic – sequential or temporal component

Programming – optimizing a linear program.

Method of solving complex problem by breaking into small steps.

Solve subproblems → Combine solutions

Output from previous state is retained to make the next step.

Optimization of policy is done. Random initial policy is updated to improve the agent.

In order to be solved by dynamic programming the problem must satisfy 2 properties –

1) Optimal substructure
   Problem can be decomposed.
2) Overlapping subproblems
   Subproblems recur many times
   subproblems can be cached & reused

MDPs satisfy both properties

Bellman equations gives recursive decomposition
Value function stores and reuses values

D.P. is a model based technique requiring knowledge about P & R in the environment

MRP Bellman equations

We know that for MRP

$$V(s) = R(s) + \gamma \sum P(s|s') V(s')$$

In Matrix format

$$\begin{bmatrix} V(1) \\ V(2) \\ V(3) \\ \vdots \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ R(3) \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & & \\ P_{11} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ V(2) \\ V(3) \\ \vdots \end{bmatrix}$$

$$V = R + \gamma P V \quad \text{—} \quad 1$$

$$(1 - \gamma P) V = R$$

$$\therefore \quad V = (1 - \gamma P)^{-1} R$$

Solving via eigenvectors for the Bellman equations — time complexity $O(n^3)$

Only possible to solve for small no of states

for others use Monte carlo, dynamic and iterative methods

DP for MRP

In DP, we use the past values to find the next values and converge at the optimal values.

$$V_{k+1}(s) = \sum_{s \in S} P(s'|s)\left[R(s,s') + \gamma V_k(s')\right]$$

In Matrix form

$$V_{k+1} = R + \gamma P V_k$$

D.P. assumes that full knowledge of MRP is present

Iterative application of Bellman expectation backup

$$V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V_\pi$$

At each iteration :

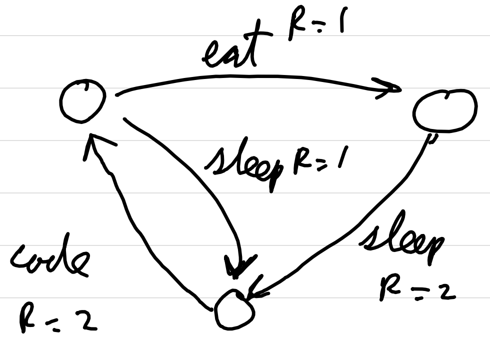
for all states $s \in S$

Update $V_{k+1}(s)$ from $V_k(s')$

(s' is successor state of s)

Markov Decision Process

Instead of rewarding the states, we will reward actions



Here decision, policy comes into picture

Tuple < S A P R γ >

A → Actions.

It incorporates actions & policies.

The objective is to calculate the optimal policy that maximizes expected return

Where agent will go is decided by π & P

π → Policy : where agent wants to go
ie which action to select

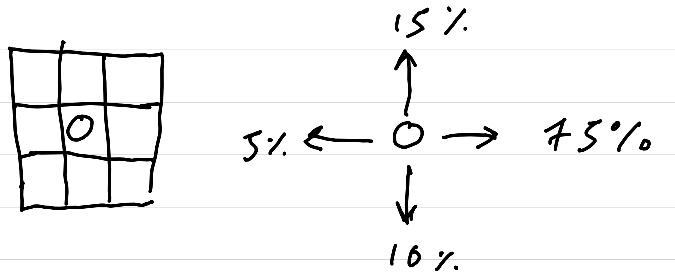
P → transition probability : where the environment takes the agent.

eg π may be choose action [up]

But for (up) action P for reaching state 2 may be 70% while 30% of times it may fall into ditch.

Policy (MDP)

Policy is a strategy used by agent



It can be thought of as probability to choose action.

The policy decides what actions the agent must take in order to reach the goal

Initially the policy can be random and then improved iteratively

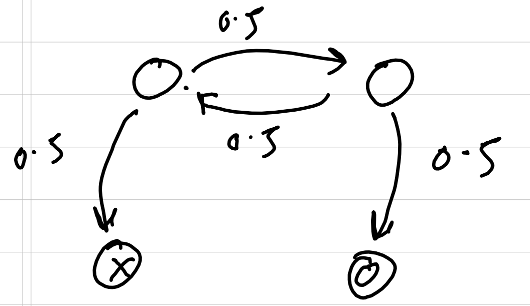
There are two types of policies

Deterministic → always consider 1 action eg up $\pi = [up]$

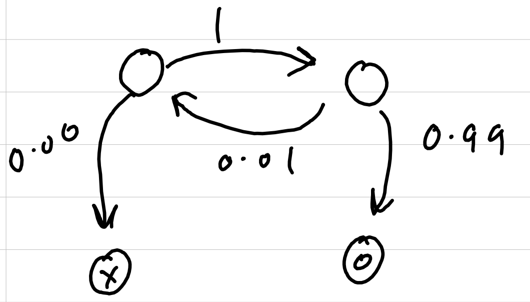
Stochastic → consider probability of transitions

$$\pi = [0.15, 0.75, 0.1, 0.05]$$

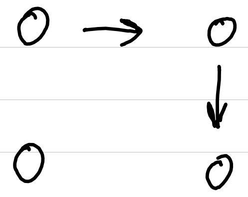
eg a random inital policy may be



But optimized policy may be



for determinstic



Deterministic policy may make moves predictable

V function in MDP

Used to evaluate goodness or badness of a state

Prediction of future reward

$V_\pi(s) = E_\pi\{G_t \mid S_t = s\}$

Value function w.r.t. Policy π = Expected total rewards to be obtained in the state

If π is stochastic then V_π is weighted probability of transitions.

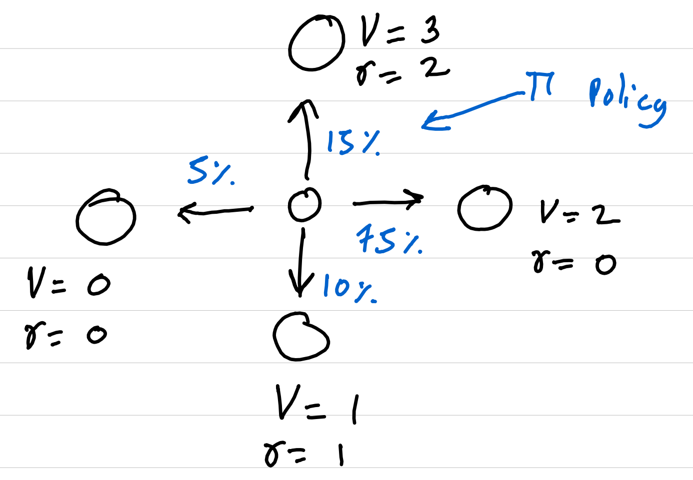
$$V = \sum_{\text{for all actions } a} \text{Probability of choosing action } a \text{ from } \pi \times \left[\text{Reward obtained Now} + \text{Expected reward of Next state from } a\right]$$

The expected reward of Next state obtained from a is nothing but V of Next state

If we want to weigh γ, we can multiply V by γ

for now assuming that agent can go wherever it wishes
ie. No wind
$P(s,a,s') = 1$

eg board games



at $\gamma = 1$

$V = 0.15 \times (3+2) + 0.75 \times (2+0) + 0.1 \times (1+1) + 0$

V needs policy π because π will decide the behaviour of agent in the environment

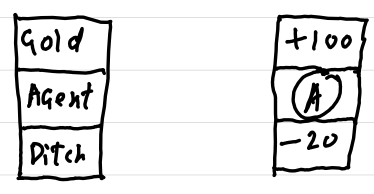
If policy π is deterministic then there will be a single action

V_π = Reward obtained + Expected reward of next state

for MDP, V always dependant on π

V is measure how good the states are w.r.t. π. Environment (P, R) also comes into picture

for MRP, V only depends on environment



Agent is at slippery area where up action may cause to slip & fall into ditch with probability 0.4

for MRP, $\uparrow 0.3$ $\downarrow 0.7$ then $V = 100 \times 0.3 - 20 \times 0.7$

Since the agent has high chance of slipping, we say state is bad

for MDP, let P up: $\uparrow 0.6$ $\downarrow 0.4$
down: $\downarrow 1$

If π $\uparrow 0.9$ $\downarrow 0.1$ then V is still good

$V = 0.9 \times (0.6 \times 100 - 20 \times 0.4) + 0.1 \times 1 \times -20$

$= 44.8$

But if π itself is bad

π : $\uparrow 0.1$ $\downarrow 0.9$ Then there is no hope that agent will reach gold

$V = 0.1 \times 0.6 \times 100 - 0.1 \times 0.4 \times 20 - 0.9 \times 1 \times 20 = -12.8$

even if the environment permits success 60% times, agent has not learnt properly and takes success only $0.1 \times 0.6 = 0.06 = 6\%$ of times

So V of state is still less, even lesser than random policy.

There is no use of the gold being there since agent cannot exploit it

So from view of π, V is still bad.

In MDP, V depends on the policy used and hence denoted by V_π

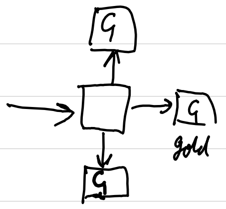
That is, goodness or badness of state depends on π & environment both

γ value (discount factor)

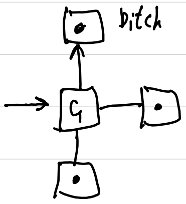
State is "good" if it gives me high rewards on reaching it

AND

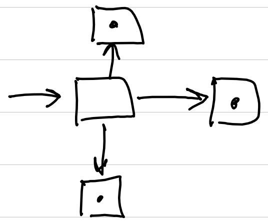
The next states that can be reached from that state are also "good"



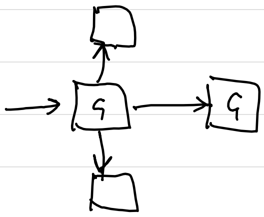
R is low but Next states are good (A)



R is high but Next states are bad (B)



R, Next state bad (C)



R, Next states good (D)

If γ is reduced, then more focus will be given to immediate Rewards

at High γ, future rewards prioritized

$\gamma = 0$ (B), (D) Greedy

$0 < \gamma < 1$ (D) Near sight

$\gamma = 1$ (D), (A) far sight

$\gamma > 1$ (A) future rewards

Policy Iteration

Goal of RL is to find the policy π & V.
Policy can be chosen to maximize the expected rewards. or going to the best states always.

V needs π to evaluate the goodness

π needs V to find best path

$\pi \rightleftarrows V$

chicken & egg problem
expectation maximization

Without chicken egg, the equation can be directly solved, but is infeasable.

In RL, EM is called as →

① Policy evaluation (find V) $\pi \to V$

② Policy improvement (find π) $\pi \leftarrow V$

This is called as

(A) Policy iteration - Randomly select policy π
find Value function V
Improve policy ....

$\odot \to \pi \rightleftarrows V$ .... until convergence to V^* π^*

V^* & π^* are optimized value function & policy π

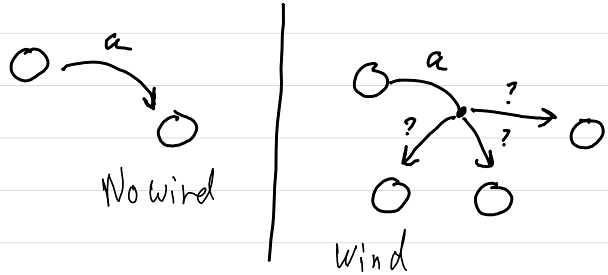
$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{M} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{M} \cdots\cdots \pi_* \to V_*$$

This method is the dynamic programming method.

Transition probabilities

This is used when taking an action does not guarantee going to a state

eg windy travel.



$$P(\underbrace{s'}_{\text{Next state}} \mid \underbrace{s}_{\text{current state}}, \underbrace{a}_{\text{action}})$$

eg $P(X \mid Y, a) = 0.8$ means that there is 80% chance of going from state X to state Y after taking action a.

So while finding V_π, these transition probabilities must also be taken into account

$$V = \sum_{\text{all actions}} \text{Probability of choosing action } a \text{ from } \pi \times \sum_{\text{all states from } a} \text{Probability of that action going to state } s' \times \left[\text{Reward} + V_{\text{Next}} \right]$$

Mathematical Notation of Policy iteration

Policy evaluation → finding V wrt π

Probability weights that agent choses action a while in state s (strategy of agent)

stochastic policy

Probability that on choosing an action a from state s agent lands on state s'. These are known beforehand from how the environment functions

Transition Probabilities

discount factor

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \left[\sum_{s' \in S} P(s'|s,a) \times \left[R(s,a,s') + \gamma V^{\pi}(s') \right] \right]$$

an action a

set of all possible actions

a Next state s'

set of all states

Reward obtained from transition s→s' by action a

goodness of Next state s'

Goodness of current state

Mathematical Notation of Policy iteration

Policy evaluation → finding V wrt π

for deterministic policy, there will always be a single action

$\pi(a|s)$ will not be a probability list but rather a single action.

Hence we dont need to take sum of all but rather only one action $\pi(s)$

$$V_\pi = \sum_{s' \in S} P\left[s', s, \pi(s)\right] \cdot \left(R(s, \pi(s), s') + \gamma v^\pi(s')\right)$$

Sum only once here

Q function

Q function captures the expected total future reward an agent in state "s", can recieve by executing a certain action "a"

$$Q(S_t, a_t) = \mathbb{E}(G_t \mid S_t, a_t)$$

Q function — state — action — Expectation function — total reward

Q function takes in a state and a possible action and tells how much total reward will be obtained.

Choose the action that maximizes future reward

Same as V function, but V function is goodness for a state while Q function is goodness for a state action pair.

Also known as action value function.

$$Q^{\pi}(s \mid a) = \sum_{s \in S} P(s' \mid s, a)\left[R(s, a, s') + \gamma V^{\pi}(s')\right]$$

V Q π P explaination

$V \rightarrow$ How good a state is
$Q \rightarrow$ How good an action (from a state is)
$\pi \rightarrow$ Probability that agent will take an action
$P \rightarrow$ Probability of going to a state from an action

If we look at form of equations, something like

$$V = \sum \pi \sum P \times (r + \gamma v')$$

represents Weighted $\underset{(\text{wrt } \pi)}{\text{sum}}$ of expected reward of actions

$$Q = \sum P \times (r + \gamma v')$$

represents expected reward of an action

which is Weighted $\underset{(\text{wrt } P)}{\text{sum}}$ of expected reward of next state

expected reward of a state $= r + \gamma v'$
$=$ actual reward $+$ goodness of state

$r(s, a, s')$ depends on current state s
action taken a
next state s'

or can be simply reward of (s, a) or (s')

Note that we are dealing with two probabilities

P — environment
π — agent policy

Hence both are weighted.

In some cases P may be fixed as 1

Mathematical Notation of Policy iteration

Policy improvement → π from V^{π}

Get the action that greedily selects the action that maximizes the expected reward

for deterministic policy

$$\pi'(s) = \max_{a \in A} \sum_{s' \in S} P(s' | s, a) \left[R(s, a, s') + \gamma V_{\pi}(s') \right]$$

for stochastic policy we need to consider the expected return from the action and state

$$\pi(s|a) \rightarrow \sum_{s' \in S} P(s' | s, a) \left[R(s, a, s') + \gamma V_{\pi}(s') \right]$$

stochastic → sum of return for action a
$\pi(s, a)$

Deterministic → Sum of return for all actions
$\pi(s)$

Choose best action

By using Q function, we can write

$$\pi(s|a) \rightarrow Q^{\pi}[s, a)$$

However there is a problem that this is not normalized.

If we want to normalize it then we can use softmax function (Boltzman function) to normalize it

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \longrightarrow \frac{e^{z_i}}{\sum e^{z_j}} \longrightarrow \begin{array}{c} \text{Prob} \\ \begin{bmatrix} 0.02 \\ 0.90 \\ 0.03 \\ 0.01 \\ 0.02 \end{bmatrix} \end{array}$$

This is done to ensure that the output values are in range (0,1) while summing up to 1

$$\text{Hence} \quad \pi'(a|s) = \frac{e^{Q^{\pi}(s,a)}}{\sum_{b \in A} e^{Q^{\pi}(s,b)}}$$

Value Iteration

Instead of going EM on V & π, why not combine it in single step?

calculate optimal value of V first and then go to π

This is called

② Value it iteration

$$V(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\left(R(s,a,s') + V(s')\right)$$

Optimal value of state is the maximum expected reward obtainable by choosing the best action and then following the optimal policy thereafter.

Goodness of current state = Reward Now + Goodness of Next best state

or more correctly

Goodness of current state = Expected reward obtained by choosing best action

$\downarrow$

action which gives best expected reward.

At last the policy can be found out

$$\pi^*(s) = \max_{a \in A} \sum_{s \in s'} P(s|s',a)\left(R(s,a,s') + \gamma V^*(s')\right)$$

Here $\pi^*(s)$ will be deterministic policy

If there is a tie, only then it will be stochastic

Policy Iteration	Value iteration
Random policy	Random value function
$\pi\ v\ \pi\ v\ \pi\ v\ \dots\ \pi^*\ v^*$	$v\ v\ v\ v\ v\ \dots\ v^*\ \pi^*$
Less computation	More computation
Faster	slower
Few iterations	Many iterations
Prediction (π, v) + Control problems (π, v, a)	Control problems

Both are
1. Guranteed to converge
2. Variations of Bellman updates
3. One step look ahead

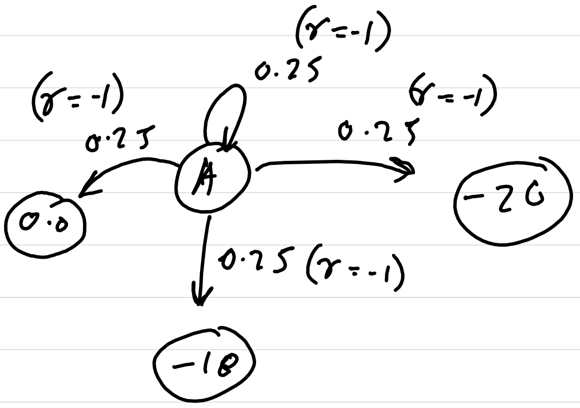
Bellman equations can be solved directly through eigenvalues, but it is a very time consuming method $O(n^3)$ hence dynamic programming methods are used

Q1 Consider the grid world problem where agent follows equiprobable random policy in deterministic transitions ($P=1$) transactions that leave the grid are treated as unchanged. $r=-1$ for all transitions. Use Bellman equations to find V of state A.

0.0	$\boxed{A}$	−20
−14	−18	−20

} V table with V(A) missing

→ since $P=1$ & $\pi = \{0.25, 0.25, 0.25, 0.25\}$



$$V(A) = \sum_{S} \pi(A,a,s') \sum_{S} P(s,a,s')\left(r(s,a,s') + V(s')\right)$$

Since $P=0$ for all states except s

$$V(A) = \sum_{S} \pi(A,a,s')\left(r(A,a,s') + V(s')\right)$$

Since $r=-1$ for all,

$$\begin{aligned} V(A) = \; & 0.25 \times (-1 + 0) \\ & + 0.25 \times (-1 + V(A)) \\ & + 0.25 \times (-1 + -20) \\ & + 0.25 \times (-1 + -18) \end{aligned}$$

$$V(A) = -10.5 + 0.25 \times V(A)$$

$$+0.75\, V(A) = 10.5$$

$$\therefore V(A) = -14$$

DP for Value iteration MDP

We can similarly denote the equations in matrix form

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

$P^{\pi} \rightarrow$ state transition probability matrix under policy π

$$P^{\pi}_{ss'} = \sum_{a \in A} \pi_{s,a} P_{s,a,s'}$$

R^{π} is the reward vector under policy π

$$R^{\pi}_{s} = \sum_{a \in A} \pi_{s,a} \sum_{s' \in S} P_{sas'} R_{sas'}$$

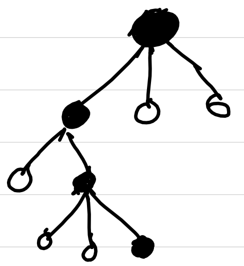
So similarly, for MDP, Dynamic programming can be used to solve the Bellman equations by step by step V_1, π_1

$V_2 \, \pi_2$

$\vdots$ so on

$$V \rightleftarrows \pi$$

Dynamic programming solves MDPs through systemic algorithms like policy iteration and value iteration



explores all options of a visited state

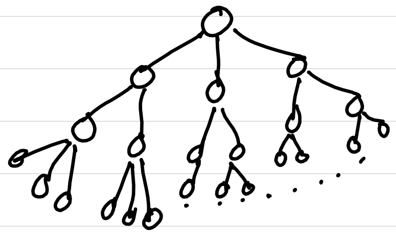
Repeat s'

Visit state s_1

search all possible next states from s_1

choose state s' that gives best result

It is different from exaustive search



explore all options of all states

Dynamic programming requires knowledge of the environments' transition probabilities $P(s, a, s')$

Most important equation →

Goodness of state = Prob of going to next state × (Reward of Next state + Goodness of Next state)

Bellman equations

MRP

V calculation

$$V(s) = R(s) + \sum_{s \in s'} P(s,s')\, V(s')$$

MDP

Policy iteration

$$\pi \rightleftarrows V$$

Stochastic Policy

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s|s',a)\left[R(s,a,s') + \gamma V(s')\right]$$

$$Q(s,a) = \sum_{s' \in A} P(s|s',a)\left[R(s,a,s') + \gamma V(s')\right]$$

$$\pi(s,a) = \frac{e^{Q(s,a)}}{\sum_{b \in A} e^{Q(s,b)}}$$

$\pi(a)$ returns probability matrix
$\pi(s,a)$ is a probability

Deterministic Policy

$$V(s) = \sum_{s' \in S} P(s|s',\pi(s))\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

$$\pi'(s) = \arg\max_{a} \sum_{s' \in S} P(s|s',a)\left[R(s,a,s') + \gamma V^{\pi}(s')\right]$$

$\pi(s)$ returns action

Value iteration

$$V \rightarrow \pi^*$$

$$V = \max_{a} \sum_{s' \in S} P(s|s',a)\left[R(s,a,s') + \gamma V(s')\right]$$

$$\pi^* = \arg\max_{a} \sum_{s' \in S} P(s|s',a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

combines to a single step