Planning by Dynamic Programming

A) Dynamic : sequential or temporal component to the problem.
We try to solve it step by step, something changes in the intermediary & we change our step or action accordingly

Programming : Optimising a "program" i.e a policy

We use DP to solve large & complex problems by breaking them into subproblems & then solve them & combine solutions to subproblems

Dynamic Programming is a very general solution method for problems which have two properties

→ Optimal substructure :
principal of optimality applies where solving the subproblems actually leads to an optimal overall problem

→ Overlapping Subproblems :
the subproblems keep appearing again & again, solving or learning to solve subproblems helps us learn something.
solution thus can be cached & reused

MDP's satisfy both these properties

* The Bellman equation gives the recursive decomposition
* The value function stores & reuses solutions

1) Dynamic Programming assumes full knowledge of the MDP

It is used for Planning in an MDP

→ For prediction : (if we move around randomly how much reward do we get)

Input : MDP $\langle S, A, P, R, \gamma \rangle$ & policy π
or : MRP $\langle S, P^{\pi}, R^{\pi}, \gamma \rangle$

Output : value function V_{π}

→ For control :

Input : MDP $\langle S, A, P, R, \gamma \rangle$

Output : optimal value function V_*
& : optimal policy π_*

(Checkout Section 1 to make sure of (1-k))

1) Policy Evaluation

Problem : evaluate a given policy π

Solution : iterative application of Bellman Expectation backup

We start off with an arbitrary value function
i.e $V_1 = 0$ & then we do a look ahead
Bellman eq & iterate till

$V_1 \longrightarrow V_2 - - - - \longrightarrow V_{\pi}$

We'll do this through synchronous backup i.e we sweep through all states of our bellman equation

Synchronous Backup

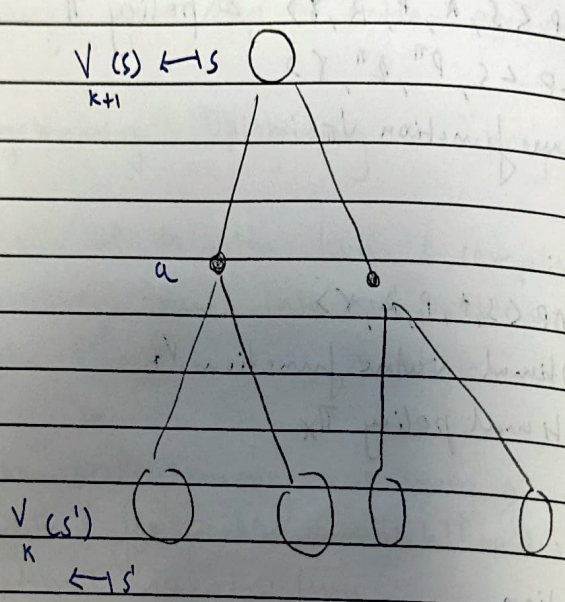
-> At each iteration $k+1$

-> For all states $s \in S$

-> Update $v_{k+1}(s)$ from $v_k(s')$

-> where s' is the successor state of s

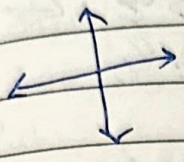
D) The Bellman Expectation Equation

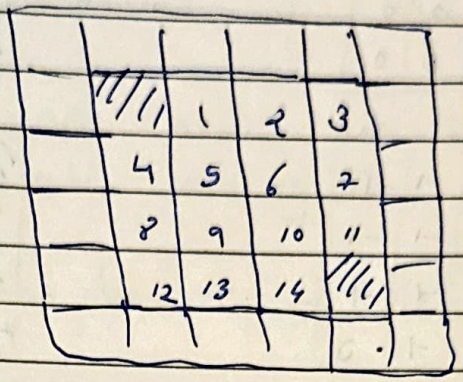


$$V_{k+1}(s) = \sum \pi(a|s) \left(R_s^a + \gamma \sum P_{ss'}^a V_k(s') \right)$$

$$v^{k+1} = R^{\pi} + \gamma P^{\pi} v^k$$

1) Evaluating a Random Policy in the small grid World





- Undiscounted MDP (i.e $\gamma = 1$)
- Non terminal states = 1 14
- One terminal state
- Actions leading out of grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows a random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = \frac{1}{4}$$

Here we're basically trying to answer what if we randomly keep moving, when would we reach the greyed box or how long would it take in expectation?

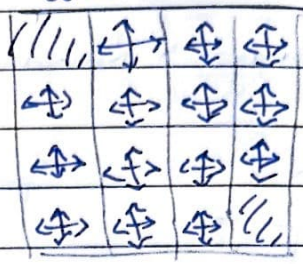
Sundaram PAGE No: DATE:

V_K for the Random policy

Greed Policy w.r.t V_K

$k=0$

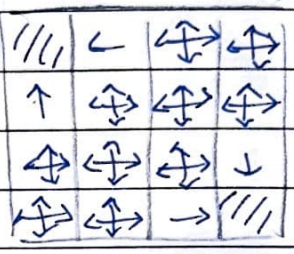
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0



← Random Policy

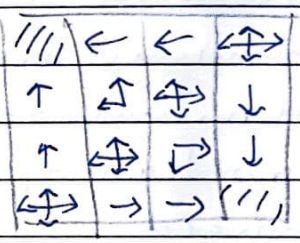
$k=1$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0



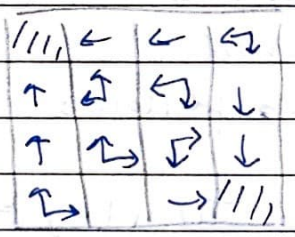
$k=2$

0	-1.7	-2	-2
-1.7	-2	-2	-2
-2	-2	-2	-1.7
-2	-2	-1.7	0



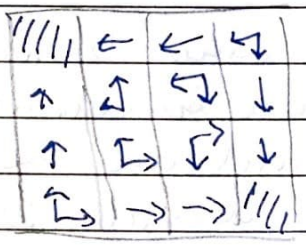
$k=3$

0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0



$k=10$

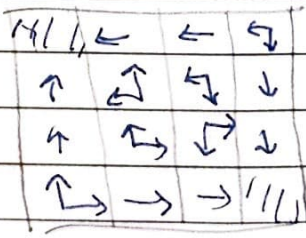
0	-6.1	-8.4	-9
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9	-8.4	-6.1	0



← Optimal Policy

$k=\infty$

0.0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0.0



In the above diagrams the left colum boxes symbolize the reward or value of each state, at the start we initialize $k=0$ V_k with 0.

But you can see how we iterate through all states using the bellman expectation equation

Eg for $k=1$, the box ★ is calculated as

$$V_{k=1}^{★} = (-1+0)\times\frac{1}{4} + (-1+0)\times\frac{1}{4} + (-1+0)\times\frac{1}{4} + (-1+0)\times\frac{1}{4}$$

-1 the reward for taking a step 2 0 is the previous value of the state we go into after the action & we do it randomly for each dir" & avg it

$$\therefore V_{k=1}^{★} = -1$$

You slowly start to notice how values update as each iteration & progresses.

With $k=\infty$, the values seem to have converged & you notice on avg it takes about 18-20 steps if you follow a random policy in this grid world

Now, based on the V_k we calculate, the right column boxes tell us that instead we try to pick actions greedily depending on our reward we start to notice that the directions on the converged $V_{k=\infty}$ are same as for $V_{k=3}$ which implies that we find an optimal policy well befor $k = \infty$

*) Policy Iteration

We wish to make our policy better - How do we do it?

→ Given a policy π

We evaluate the policy π

$$V_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

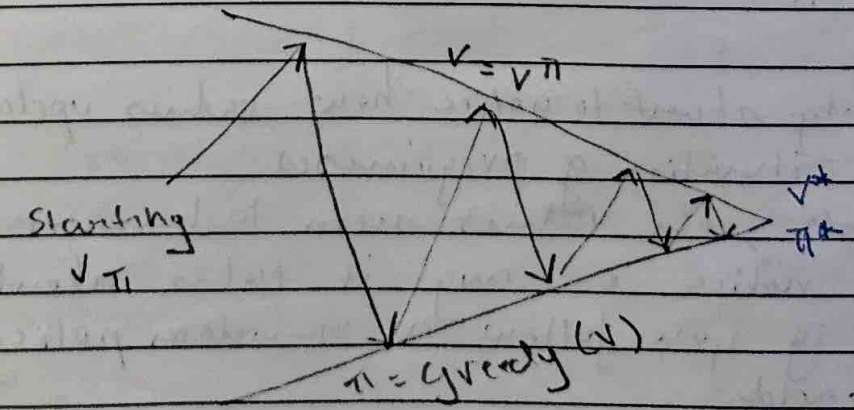
Improve the policy by acting greedily wrt to V_π

$$\pi' = \text{greedy}(V_\pi)$$

→ In a small gridworld improved policy was optimal, $\pi' = \pi^*$

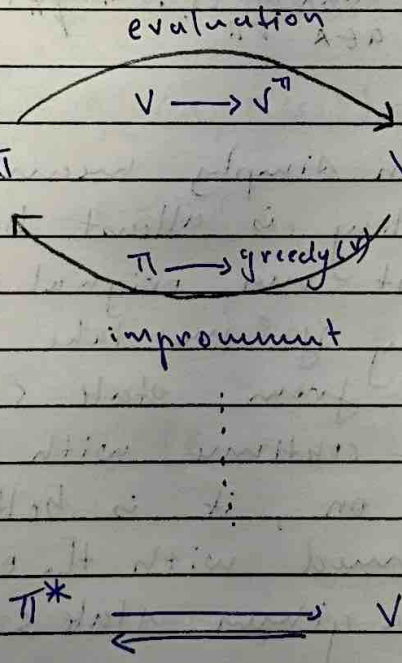
→ In general, need more iterations of improvement / evaluation

→ But this process of policy iteration always converges to π^*



→ We start of with some arbitary input to our value function like 0 & some policy
So we shall do policy evaluation on our up arrows & policy improvement on the down arrows

→ So first we calculate our value function wrt to our orignal policy then we act greedily or update policy depending on our value function & we repeat this till we converge to optimal value function & policy



So it does not matter where we start with what value function or policy we will always end up / converge to the optimal policy or value function

4) Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can then improve the policy by acting greedily

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\, q_\pi(s,a)$$

- Taking the argmax improves the value from any one state s over atleast one step.

$$q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s,a) \geq q_\pi(s, \pi(s)) = V_\pi(s)$$

This above equation simply means to determine if our greedy policy is atleast better for one step, than just our original policy i.e. $q_\pi(s, \pi'(s))$ says if we take a greedy step $(\pi'(s))$ from state s i.e a single step & continue with the original policy π thereon, it is better than if we had continued with the original policy $V_\pi(s)$ from the given state s

$\therefore$ $q_\pi(s, \pi'(s))$ tells us simply how much value we get if we start in s, follow action 'a' for one step (the greedy policy π') & then follow π afterwards

$V_\pi(s)$ tells us the value we get following π from s

→ Since $\max_{a \in A} q_\pi(s,a)$ gives us the max value of action from state s so that does tell us $q_\pi(s, \pi'(s))$

So now over all actions has to be atleast as good as our particular action we choose based on π

It thus improves the value function, $V_{\pi'}(s) \geq V_\pi(s)$

$$V_\pi(s) \leq q_\pi(s, \pi'(s)) = E_{\pi'}[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s]$$

$$\leq E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s]$$

$$\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s]$$

$$\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s] = V_{\pi'}(s)$$

This tells us that if we pick this greedy policy, the total amount of reward we get by being greedy is atleast as good as the policy we had before greedyfying it.

→ If improvement stops,

$$q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s,a) = q_\pi(s, \pi(s)) = V_\pi(s)$$

Then the Bellman optimality eq^n has been satisfied

$$V_\pi(s) = \max_{a \in A} q_\pi(s,a)$$

Thus, $v_\pi(s) = v_*(s)$ for all $s \in S$

So thus if policy $v_\pi(s) = \max_{a \in A} q_\pi(s, a)$

then $v_\pi(s)$ is optimal & π is optimal

4) Modified Policy Iteration

In the earlier grid world example we noticed that we already found the optimal policy at $k=3$, then why $k = 4 \ldots$ do value iterations needed?

∴ Does policy evaluation need to converge to v_π?

- We could introduce a stopping condition
  - e.g. ϵ-convergence of value function, if we are close in the changing value's of the bellman equation by a tiny amount ϵ we stop.
- Or simply we stop after k iterations
- Why not update policy every iteration? i.e stop after $k=1$
  This is equivalent to value iteration

1) Principal of Optimality

Any optimal policy can be subdivided into two components

- An optimal first action A_*, so if your first action is optimal & you follow an optimal policy from where you end up, the overall behaviour is optimal

- Theorem

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if & only if

→ For any state s' reachable from s

→ π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$

1) Deterministic Value Iteration

→ If we know the solution to subproblems $v_*(s')$

→ Then solution $v_*(s)$ can be found by a one step look ahead

$$v_*(s) \leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \right)$$

The idea of value iteration is to apply these updates iteratively

Intuition: start with final rewards & work backwards

4) Value Iteration

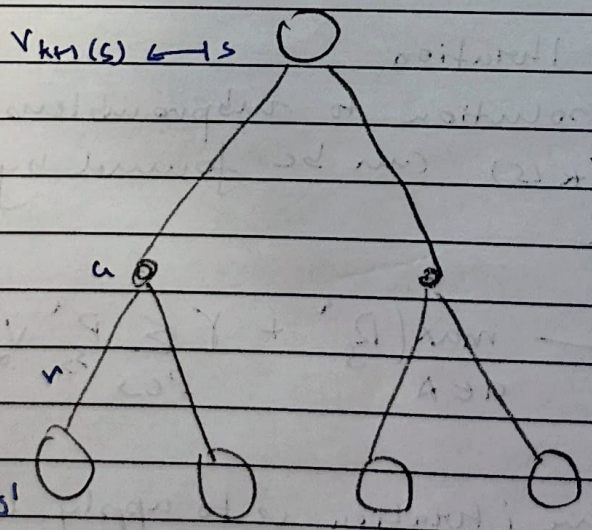
Problem: find optimal policy π

Solution: iterative application of bellman optimality ~~equation~~ back up

$$V_1 \longrightarrow V_2 \longrightarrow \dots V_*$$

Using Synchronous backups

- At each $k+1$ iteration
- For all states $s \in S$
- Update $V_{k+1}(s)$ from $V_k(s')$
- Unlike policy iteration, there is no explicity policy
- Intermediate value function may not correspond to any policy



$$V_{k+1}(s) = \max_{a \in A} \left(R_s^a + \gamma \sum P_{ss'}^a V_k(s') \right)$$

$$V_{k+1} = \max_{a \in A} R^a + \gamma P^a V_k$$

4) Understanding it All:

Policy Evaluation:
Here we are given a fixed policy π & we compute V_π

$$V_{k+1}(s) = \sum_a \pi(a|s)\left(R_s^a + \gamma \sum P_{ss'}^a V_k(s')\right)$$

The policy does not change here, we are given a policy that tells us how & with what probability we take action (eg Random in grid world eg or rather $\frac{1}{4}$ in each dirⁿ N, S, W, E) based on this fixed policy we keep calculating V_k's for $k = 1, 2, \ldots \infty$ till the values converge & thus $V_\infty = V_\pi$ or the evaluation for that particular policy.
Basically answering the question if we commit to this policy forever, how good is it?

Policy Iteration & Improvement
Now the next step is to ask knowing how good your policy is how do we improve it?
For this first we look at the policy improvement theorem where we define

$$q_\pi(s,a) = R_s^a + \gamma \sum P_{ss'}^a V_\pi(s')$$

& a new policy $\pi'(s) = \arg\max_a q_\pi(s,a)$

which guarantees $V_{\pi'}(s) \geq V_{\pi}(s)$ $\forall s$

Now why, $q_\pi(s,a)$ tells us the value of taking an action a now & then following π however $q_\pi(s, \pi'(s))$ tells us instead of taking action a based on the policy we take a based on $\pi'(s)$ which is $\arg\max_a q_\pi(s,a)$ & then following π

Choosing the action with the maximum q_π cannot logically make things worse, since the $V_\pi(s')$ of the later part of the equation is characterizing all future rewards thus the max $q_\pi(s,a)$ cannot be exceeded by the arg. $q_\pi(s,a)$ ↓ $\pi'(s)$

using the policy improvement theorem tells us if we act greedily we get a policy that is never worse than before

Now comes the Improvement
First we take a policy any initialized policy, then valuation on that policy π thus V_{π_1}, V_{π_1} $k=1$, V_{π_1} $k=2$ ... till V_{π_1} $k=\infty$ or it converges & we get V_{π_1}, then we update our policy π_1 to the new π_2 based on greedily selecting the action, & we repeat process till π stops changing & we get same policy back.

This leads to many evaluations & policy updates & can be expensive so one possibility is modified Policy iteration.

Here we either evaluate till $k = 1, 5, 10$ steps & then update policy greedily & repeat.

However there is one extreme case $k = 1$ this is value iteration.

Value Iterations means we evaluate the policy for just one sweep $k = 1$ & then the evaluation is processed by acting greedily immediately & we never store policy.

i.e

$$v_{k+1} = \max_a \left(R_s^a + \gamma \sum P_{ss'}^a v_k(s') \right)$$

Once we have v^* or as v converges, finding optimal policy is trivial because

$$\pi^*(s) = \arg\max_a \left(R_s^a + \gamma \sum P_{ss'}^a v^*(s') \right)$$

M)

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Eq"	Iterative Policy Evaluation
Control	Bellman Expectation Eq" + Greedy policy Improvement	Policy Iteration
Control	Bellman Optimality eq"	Value Iteration

- Each of these algorithms are based on state-value function $V_\pi(s)$ or $V_*(s)$
- So Time complexity is $O(mn^2)$ per iteration for m actions & n states because there are n states & for each n states there are n possible successor states
- Could also be applied to action-value function $q_\pi(s,a)$ or $q_*(s,a)$
- Complexity $O(m^2n^2)$ per iteration

N) Asynchronous Dynamic Programming

In this one all states are backed up in parallel, Asynchronous DP backs up states individually & in any order

We apply the appropriate backup for each selected state

This reduces computation significantly

This is guaranteed to converge provided all states are continued to be selected

Ideas for Asynchronous DP

→ Inplace DP

Synchronous value iteration stores two copies of the value function for all s in S

$$V_{new}(s) \leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'} V_{old}(s') \right)$$

$$V_{old} \leftarrow V_{new}$$

Inplace only stores one copy of value function for all s in S

$$V(s) \leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V(s') \right)$$

So now once we compute the V(s) for a state if that is part of the computation for another state, we use its updated value & not its previous or original value

So what matters here is the order of updation, so if you have an order of where you update the state closer to the goal & keep going backwards, this helps you be efficient. & Thus order of updation really matters.

→ Prioritized Sweeping

So given the fact ordering is important we need to come up with a mannor to identify which states are really important & which one should you update first

& we can use the magnitude of the Bellman error to guide our state selection eg

$$\left| \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V(s') \right) - V(s) \right|$$

so we want to propogate those states first which have the highest ensuing Bellman error

Real time - DP

Have a real agent to get relevant state information use the agents experience to guide state selection