RL Lecture 3 :

Dynamic Programming

Dynamic → sequential nature of problem (which we have)

Programming → optimize a program ← policy (for us)

Idea → Break into sub-problems → combine solutions of sub-problems.

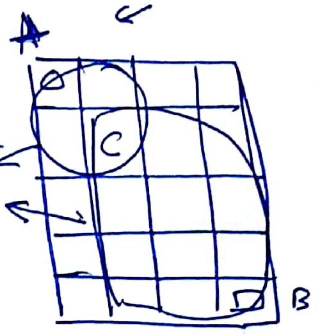
Problems should have 2 properties

① Optimal substructure → problem consists of smaller similar problems

② Overlapping subproblems

The subproblems should occur multiple times.

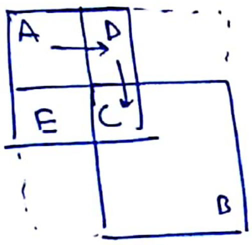
eg.



smallest distance from A to B, can be viewed as smallest distance from A to C & then C→B.

similar subproblems

eg.



Suppose I tried path A→D and then went from D→C and I need to solve the problem of C→B now.

However, if I go from A→E & then E→C.

I still need to solve the same problem for C→B. ← This is an example of overlapping subproblems.

MDP → satisfies both → Bellmann equatn gives recursive decomposition

In DP → solutions can be ~~~~ cached.

In RL, Value functn → cache of the informatn already figured out about the state → then work the way backwards to get the result.

Recall : Planning → knowledge of MDP already given (eg. rules of a game) and then solve the MDP.

2 types of problems :

① Prediction → output the value functn v_π.

I/p → MDP / MRP and policy

② Control → MDP given but we need to figure out the most important / optimal policy

Op → optimal value functn v_π (hence, also the optimal policy

I/p → MDP.

Policy Evaluation (MDP & Policy given, what is the rewar

Bellman expectatn equation to be used in an iterative manner.

$v_i \to$ value functn at each iteration

$$v_1 \to \begin{bmatrix} v_{1s_1} \\ v_{1s_2} \\ \vdots \\ v_{1s_n} \end{bmatrix} \to \text{initial value function} \to 0 \text{ is a safe default.}$$

Then, using the 2-step lookahead, find the next new value functio

$v_2 \to \ldots$ so on.

$v_\pi \to$ final optimal value function.

Synchronous Backups:

$$\begin{bmatrix} v_{1s_1} \\ \vdots \\ v_{1s_n} \end{bmatrix} \xrightarrow{\text{update all the states}} \begin{bmatrix} v_{2s_1} \\ \vdots \\ v_{2s_n} \end{bmatrix} \ldots \to$$

So, at each step, we are updating all the stat in the value function at the last iteration → synchron backup.

at each iteratn $k+1$:

→ update all states from previous iteratn

→ $v_{k+1}(s)$ from $v_k(s')$

→ $s' \to$ successor state of s.

This does converge!!

Scanned by CamScanner

[Expectatn backup → as Expectation over backups.
Rather than max → in the case of optimal policy]

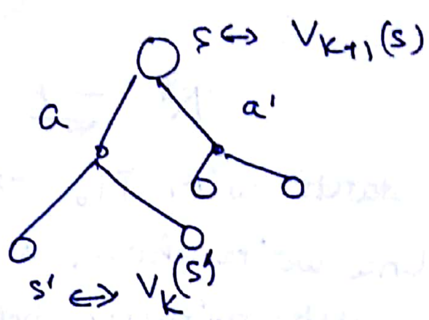
Use the 2-step lookahead tree. ~~[illegible]~~.

~~$V_{k+1}(s) \leftrightarrow s$~~

(Vector)

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

$V_k(s')$ → Suppose $V_K = \begin{bmatrix} V_{KS_1} \\ \vdots \\ V_k(s') \\ \vdots \end{bmatrix}$



$s \to s'$ → following the policy given

Here, the iterative algorithm is that, at each state s, to compute the V_{k+1} value for it, we look at the values of the value function for all its successors (1 step look-ahead) from the previous iteration

This is guaranteed to converge.

Also, even though we have one policy (eg. random policy), the value function essentially helps us get a better policy just by looking at the rewards and value functn of the next step.

Policy iteration → improve the policy using a feedback mechanism to feed in the improved policy obtained through the value function.

Policy given → π

① Evaluate the policy
② Improve by acting greedily wrt V_{π}

$\pi' = \text{greedy}(V_{\pi})$ → whichever gives better value function, pick that!

This process always converges to π^*.
(atleast 1 deterministic optimal policy exists).

Policy evaluation $\longrightarrow$ Policy improvement

$\pi^* \geq \pi$

$\pi^* \rightleftarrows V^*$.

starts with π_0 → convergence rate independent of π_0

Once we've chosen a policy greedily, we anyway now have a deterministic policy (Think!) → in step 1.

① Let's start with a deterministic policy : $a = \pi(s)$

② $\pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$ → was in state s, took action a and then followed policy π

③ $q_\pi(s, \pi'(s))$ → what will be the value function.

$= \max_{a \in A} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = V_\pi(s)$ → as from state s, followed followed π to get next action and followed π from there onwards

∴ Atleast improves in one step following $\pi'(s)$

④ $V_{\pi'}(s) \geq V_\pi(s)$

$V_\pi(s) \leq q_\pi(s, \pi'(s))$ → unroll it

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

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

$\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s]$

$= V_{\pi'}(s)$.

Scanned by CamScanner

RL. Lecture 3.

So, proved that following the greedy policy, we would atleast atleast equal value, if not, more. Still, haven't told that it'll keep increasing. I go to optimal.

If we have equality → improvements stop!

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

then Bellman optimality eq^n satisfied

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

$\pi \to$ optimal policy.

∴ Policy iteration solves MDP.

Question → If in the 3rd iterat^n itself, we can get the optimal values, iterations 4 to ∞ are wasted. → Any way to truncate the iterative procedure / do an approximation?

Modified Policy Iteration → ① add an early stopping criterion eg. if the Bellman eq^n value difference in consequent iterations is < ε.

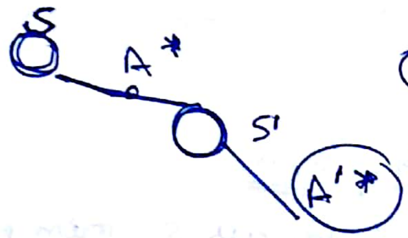
② Train / Run for only K iterations. eg (K=3 above).

Extreme case → K=1 → value iteration (most famous use of DP).

Optimal policy

① Take optimal action 1st → A_*

② wherever you land → the next state, because of this action, it should be optimal from there too.



$\pi(a|s) \to$ optimal when:

① $V_\pi(s) = V_*(s)$

② For all reachable s', $V_\pi(s') = V_*(s')$.

Scanned by CamScanner

Deterministic value Iteration:

Assumption → already know the best solution for the next step. → s' → $V_*(s')$ → use it to find the optimal solⁿ for the previous step.

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

↓

can be found by one-step lookahead.

Difference betⁿ policy iteratⁿ & value iteration

Policy iteration → start with random policy, find the value functⁿ and improve policy based on this value function.

Value Iteration → Start with random value function, and keep improving that value function

Policy Iteration

① Initialization → $\pi(s) \in A(s)$ (arbitrary)

② Policy evaluation:

Repeat

$\Delta \leftarrow 0$

for all $s \in S$: $a = \pi(s)$ — as becomes deterministic in greedy choice

$$V_{k+1}(s) = \cancel{\max \sum \pi} \; R_s^a + \gamma \sum_{s'} P_{ss'}^a V_k(s')$$

$$\Delta = \max(\Delta, |V_{k+1}(s) - V_k(s)|)$$

if $\Delta < \theta$ → stop

③ Policy improvement:

for all $s \in S$

$$\pi(s) = \max_a q_\pi(s,a)$$

if no change in $\pi(s)$ for all s, then policy stable and stop. Else go to ②.

Scanned by CamScanner

Value Iterat^n

→ Initialization : $V_*(s) = 0 \;\forall\; s \in S$ eg.

→ Repeat for all $s \in S$:

$$V_*(s) = \max_a \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a V_*(s')\right).$$

if change for all s, very small ($<\epsilon$). → stop.

→ for all s,
$$\pi(s) = \arg\max_a \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a V_*(s')\right).$$ policy updated just once

We are not solving the full RL problem as MDP's knowledge already given.

Value Iteration → any intermediate value (before converging) may not actually be the value functn for any policy, i.e.. may not be V_π for any π.

But at the end, we get the value functn for the optimal policy.

Bellman expectatn equation

→ $V_{k+1}(s) = \sum_{a \in A} \pi(a/s) \left[R_s^a + \gamma \sum_{s'} P_{ss'}^a V_k(s')\right]$

Since $\pi(s)$ becomes deterministic, $\pi(a/s) = 1$ for one action and 0 for the rest $a \in A$. (leading to this)

Extensions to DP

Till now synchronous.

Asynchronous → pick any state and update its value function → $V_{k+1}(s)$ (for eg.)

now immediately use this value for the calculation of value function for other states instead of $V_k(s)$.

⇒ Break iteration.

→ Better computation

① In-place

Basically, as soon as we update a value iteration ↑ use it directly

② Prioritized sweeping → use mag. of Bellman error → to choose which states to update 1st.

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

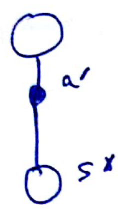
③ Real time DP → Use only states that are actually visited by the agent → use agent's experience → real-world

$$S_t = \max_{a \in A} \left(R_{S_t}^a + \gamma \sum_{s'} P_{S_t s'}^a V(s') \right)$$

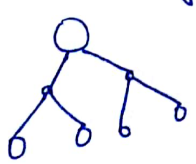
DP → does full width backups → i.e. goes through all the actions and subsequently all the states. to compute its prediction

→ not feasible and we need to know the full model dynamics

Instead → we'll do → sample backups → sample an action, sample subsequent states ... etc and then do the backup instead of full



instead of



(+) helps overcome curse of dimensionality

Scanned by CamScanner

eg. of sampling with DP

function approximator

Approximate DP → Approx. value function → $\hat{V}(s, w)$

Fitted Value Iteration

Repeat

① Sample $\tilde{S} \subseteq S$ ← states sampled

② for each $s \in \tilde{S}$,

$$\tilde{V}_k(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a \hat{V}(s', w_k) \right).$$

use $\{\langle s, \tilde{V}_k(s) \rangle\}$ → to train $\hat{V}(s', w_{k+1})$.

Contraction mapping → confirms the convergence of value & policy iteration.

Vector space →

$$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \rightarrow \text{dimension} = \#\text{states} = |S|$$

Want to show that Bellman backup brings value functions closer in subsequent iteration and hence, it converges.

Distance betn state value functions u & v. → ∞ norm

$$\Rightarrow \|U - V\|_\infty = \max_{s \in S} |u(s) - v(s)|$$

Bellman expectatn operator → T^π → brings value functions closer by atleast γ-times → γ-contraction

$$T^\pi(v) = R^\pi + \gamma P^\pi v$$

$$\|T^\pi(u) - T^\pi(v)\|_\infty \le \| \gamma P^\pi \|v - u\|_\infty \|$$

$$\le \gamma \|u - v\|_\infty$$

now, if u, v are not already same, then ↑ earlier dist was only $\|u - v\|_\infty$.

← following contraction mapping theorem → for vector space V, closed under operator $T(v)$, T → γ-contractn,

- T converges to a unique fixed point
- linear convergence rate → γ..

iterative policy evaluatn converges to V_π.

∴ Policy iteratn → V_*.

Scanned by CamScanner

1 Bellmann optimality backup operator → T^*

$T^*(V) = \max_a (R^a + \gamma P^a V)$ → similarly a γ-contraction

→ hence, ~~policy~~ value iterate also converges.

Scanned by CamScanner