Temporary Difference

Saturday, May 7, 2022 10:36 AM

→ Value iteration: requires P_{MT} and f_R to be known to compute either $V(S)$ or $Q(S,a)$ by approximation (Bellman Equations)

either given or learnt (program random)

↳ (Iteration)

In memory we needed: P, R and V/Q

→ Steps:

1. 100 random transitions to compute P and R — always an approximation (learned)
2. With P and R, we do 1 value iteration
3. Again 100 random to adjust P and R

... repeat

we can do this 10,000 times in the beginning and then do all the → value iterations together (same)

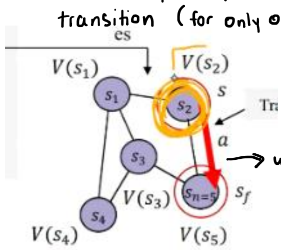
Another way: Temporary Difference

↳ Adjustment approximation per transition

→ Temporary Difference is another way of solving the Bellman Equations

↳ Where each $V(S)$ cell or $Q(S,a)$ row is calculated transition by transition:

means ↳ We modify only one $V(S)$ / $Q(S,a)$ value during each transition (for only one state per transition/at a time).



→ when we transition from S_2 to S_f using action a: receives r, and with r plus the transition $S_2 \to S_f$ we adjust $V(S_2)$

advantage: we no longer need P_{MT} in advance (implicit in V/Q)

If we want the agent to learn the whole $V(S)$ the agent needs to experience all states (unlike in Value Iterations)

[We adjust $V(S_4)$ when the agent is in $V(S_4)$ and transitions: to calculate the whole $V(S)$ if we got, say 10 states, we need 10 transitions, so that the ten $V(S)$ are updated.

$$\underbrace{V(S)}_{\text{Truer value}} \leftarrow \underbrace{V(S)}_{\text{previous/current value}} + \alpha \Delta V(S)$$

each $V(S)$ is approximated step by step ↓ a kind of value iteration

error → increment in the direction of $V(S)$: in the direction that fixes the error } the error will tend to zero as we transition

α ↳ decides how much of the error we consider

↳ α can be greater than 1

↳ $\alpha \in [0, \infty]$

where

$$\Delta V(S) = [\underbrace{f_R(S,a,S_f)}_{\text{immediate reward}} + \gamma V(S_f)] - V(S)$$

$\gamma V(S_f)$ → also approximated

$[f_R(S,a,S_f) + \gamma V(S_f)]$: what we think/expect V to have

$V(S)$: what we currently have

Since $V(S)$ doesn't tell us the action (unlike Q) and we need to choose the a with the reward function ↓ needed

$$Q(S,a) \leftarrow Q(S,a) + \alpha \Delta Q(S,a)$$

$$\Delta Q(S,a) = [\underbrace{f_R(S,a,S_f)}_{\text{the immediate reward}} + \gamma \max_{a_f} [\underbrace{Q(S_f, a_f)}_{Q(S,a)\ \text{(approximated)}}]] - Q(S,a)$$

we don't need to know the reward function completely ↳ not needed

α → defines the speed of learning

→ α small: slow learning, takes more time to reach desired value

→ α big: fast learning, takes less time to reach desired value, but if it reaches this value, it oscilates in a larger interval than it would with a small α.

Method

a) For $V(S)$ version $V(S) \leftarrow V(S) + \alpha \Delta V(S)$ where $\Delta V(S) = [f_R(S,a,S_f) + \gamma V(S_f)] - V(S)$

↳ The transition model function does not intervene, only reward func. ↳ needed

To Solve by Temporary Difference Iterations:

↳ This method makes the agent take random actions and we adjust V(s) value per transition of the agent during those random actions OR we put random values in V(S) and start to learn.

↳ when we compute max() in V(s) to know the action, that max is over all possible V(s) values for all possible actions in the world.

↳ 2 actions max (V(a₁), V(a₂))

involved in a transition

$s \to s_f$ with $r = -0.04$

	initial state sF1	a1=← s1	a2=→ s2	s3	sF2
fR(s)=	-10	0	-0.04	-0.04	10
		s →	s_f		
V(s)=	0.02	0.1	0.6	0.5	0.3

$f_R(s,a,s_f)$: needed (may be learned during)

start with random values ∈ [0, 1]

value to change (current)

to know where to move go to s_1 and see max (a_1, a_2) generated with random nums

since we take a_2 → we move to right and thus adjust 0.1 using −0.04 and 0.6

example

γ=	0.9
α=	0.8

$\boxed{\text{cell} = 0.1 + \alpha((-0.04 + \gamma(0.6)) - 0.1)}$ (1)

= 0.42 instead of 0.1 now

when you write this the 0.09 will change too.

To decide which a: max of $(V(a_1), V(a_2))$ all actions

MAX = a_2 →

-9.982	x	0.09				
	-9.982	x →	0.5			
		0.09	x	0.41		
			0.5	x	10.27	
				0.41	x→	10.27

Here is the only place where we use f_R (not in the adjustment)

$-10 + \gamma(0.02)$

$-0.04 + \gamma(0.6)$

↳ in the adjustment we use r (immediate) not f_R
↓
precise (world gives it)

Next: agent moves to $s_2 \to s = s_2$

the only way to know there is a 10 reward is if the agent experiences a transition that gives 10. From then on, the values adjust with back propagation, so that they guide the agent with an optimal politic approximation.

	sF1	a1=← s1	a2=→ s2	s3	sF2
fR(s)=	-10	0	-0.04	-0.04	10
			s :→	s_f	would adjust 0.5
V(s)=	0.02	0.42	0.6	0.5	0.3

γ=	0.9
α=	0.8

to change with (1)

Thus (result of → info is used)

cell = $0.6 + \alpha((-0.04 + \gamma(0.5)) - 0.6)$

= 0.498 instead of 0.6 now

MAX = →

-9.982	x	0.378				
	-9.982	x	0.5			
		0.378	x	0.41		
			0.5	x	10.27	
				0.41	x→	10.27

once s = final state, we adjust the 0.3 and we put the agent(s) in s(0) = s_1 → new iter and adjust again etc.

How do we stop? after each V(s) adjustment (all), test the agent with that politic and count how many times (%) it reaches Goal

85%: we finish that V(s) is approx 'nice'.

the final V(s) will be $\boxed{\text{close}}$ to the optimal V(s)