Reinforcement learning (4#16)

* Reinforcement learning

- → MDPs
- → Value function
- → Value Iteration
- → Policy Iteration

} algorithms to solve reinforcement learning problems

* CREDIT Assignment Problem

Suppose you are playing a chess. You lost a game at 60th move. CAP tries to associate reward with every move you take. However, you get to know about the outcome of chess after 60th move.

* Formalizing RL

RL model the world using Markov Decision Process (MDPs)

MDP

is a five tuple: $(S, A, \{P_{sa}\}, \gamma, R)$

S: set of states (in helicopter example it will be the possible orientation helicopter could be in)

A: set of action (set of possible controller configuration)

P_{sa}: State-transition distribution

$$\sum_{s'} P_{sa}(s') = 1 \quad , \quad P_{sa}(s') \geq 0$$

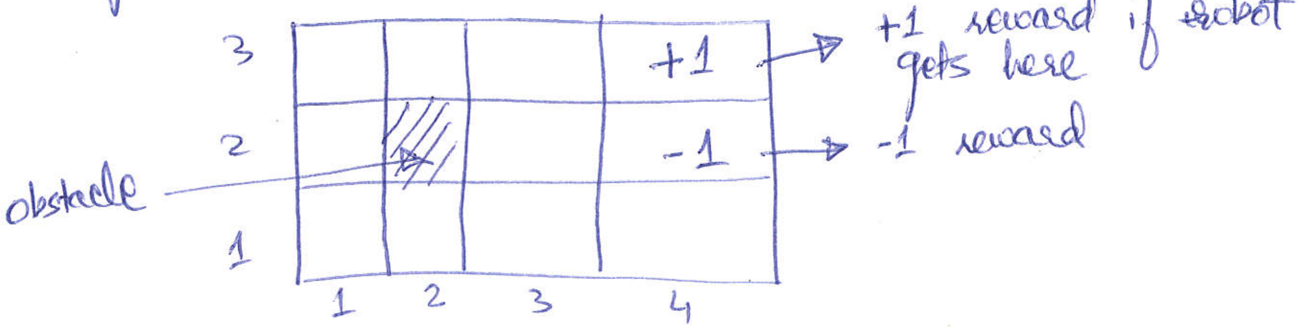
($P_{sa}(s')$ is the probability of transitioning in state s' if we take action "a" in state s).

γ: Discount factor $0 \leq \gamma < 1$

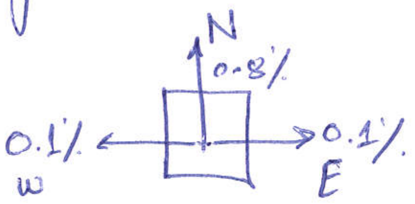
R: reward function $R: S \longrightarrow \mathbb{R}$ (real-numbers)

Running Example for Peter Russel book (Robot Navigation Example)

③



- We have 11 - states here (Robot can be in any of these places)
- Robot can take 4 actions: $A = \{N, S, E, W\}$
- Robot is stochastic dynamic as well, so we will model it as well



(If we command the robot to move "N", then there is 80% chance that robot will move N & 10% that it will deviates toward left & right)

- Lets write state transition probability when robot is at location (3,1) & we ask the robot to move to (3,2)

$$P_{(3,1),N}((3,2)) = 0.8$$

move north

$$P_{(3,1),N}((4,1)) = 0.1$$

$$P_{(3,1),N}((2,1)) = 0.1$$

$$P_{(3,1),N}((3,3)) = 0$$

⋮

etc

$$P_{\underbrace{(3,1)}_{s},\underbrace{N}_{a}}(\underbrace{(4,1)}_{s'}) = 0.1$$

If robot is at state (3,1) and takes action "N", then the probability that robot will ~~be~~ transition into state (4,1) is 0.1 or 10%.

* Reward (R):

We have $R((4,3)) = +1$

$R((4,2)) = -1$

and $R(s) = -0.02$ (~~slight~~ small negative reward for all other states)

Assumption

World ends when robot reaches $R((4,3))$. (Zero cost absorbing state)

How MDPs work

At state S_0

Choose a_0

Get to S_1 by randomly drawing s_1 : $S_1 \sim P_{s_0 a_0}$

Choose a_1

Get to $s_2 \sim P_{s_1 a_1}$

⋮

$$S_0 \xrightarrow{a_0} S_1 \sim P_{s_0 a_0} \xrightarrow{a_1} S_2 \sim P_{s_1 a_1} \rightarrow \cdots \boxed{\text{end}}$$

After a while, robot would have visited sequence of state

$s_0 \quad s_1 \quad s_2 \quad \ldots$

And in order to evaluate how well robot did, we will apply reward function to states & sum them

$$R(s_0) + R(s_1) + R(s_2) \ldots$$

We will sum them up with discount factor, giving total payoff

$$\text{Total Payoff} = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

$0 \leq \gamma < 1$ (i.e reward of future state have less weight than their past)

The goal of learning algorithm is to choose actions overtime $(a_0, a_1, \dots)$ to maximize the expected value of total payoff:

$$E\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots\right]$$

More Concretely, RL will compute a policy π.

$$\text{Policy } \pi : \underset{\substack{\uparrow \\ \text{state}}}{S} \mapsto \underset{\substack{\uparrow \\ \text{action}}}{A}$$

An Example of Policy π

Optimal Policy for Robot Navigation

3	→	→	→	+1
2	↑	▨	↑	-1
1	↑	←	←	←
	1	2	3	4

→ If you are are in state (4,1), then take "left" action i.e $\pi((4,1)) = W$

$A: \{W, N, E, S\}$

* This policy will maximize the expected value of total payoff

Lets define the following:

$$V^{\pi}, V^*, \pi^*$$

π^* seems to be optimal policy, butt it is not. It will be optimal as a consequence of our definition.

For any policy π:

For any π, define value function $V^{\pi} : S \mapsto \mathbb{R}$

s.t $V^\pi(s)$ is expected total payoff starting in state s, and execute π.

$$V^\pi(s) = E\left[R(s_0) + \gamma R(s_1) + \cdots \;\middle|\; \pi, s_0 = s\right]$$

This is sloppy notation as π is not a random variable and we shouldn't be conditioning on it.

As a concrete example, lets consider (bad) policy π, and its value function V^π

π

3	→	→	→	+1
2	↓	▨	→	-1
1	→	→	↑	↑
	1	2	3	4

V^π

3	.52	.73	.77	+1
2	-0.9	▨	-.82	-1
1	-.88	-.87	-.85	-1
	1	2	3	4

$$V^\pi(s) = E\Big[\underbrace{R(s_0)}_{\text{Immediate reward}} + \gamma \underbrace{\underbrace{\big(R(s_1) + \gamma R(s_2) + \cdots\big)}_{\text{future rewards}}}_{V^\pi(s_1)} \;\Big|\; \pi, s = s_0\Big].$$

$s_0 \rightarrow s$
$s_1 \rightarrow s'$

$$V^\pi(s) = R(s) + \gamma \sum_{s'} \underbrace{P_{s\pi(s)}(s')}_{P_{sa}(s')} V^\pi(s') \quad \Big\{ \text{we are exactly policy } \pi$$

$\hookrightarrow$ Bellman's Equation

$$V^{\pi}(s) = R(s) + \gamma \cdots\cdots V^{\pi}(\overset{\uparrow}{s'})$$

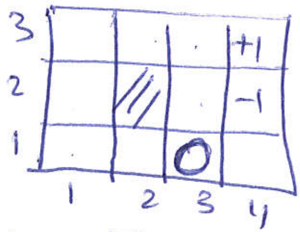
s' is random

$P_{sa}(s')$ where $a = \pi(s)$

$P_{s\pi(s)}(s')$

* Bellman's Equation allows us to solve $V^{\pi}(s)$. Given a policy π, how to compute $V^{\pi}(s)$. Answer is Bellman's equation.

Lets make Bellman's Eq more concrete. Lets consider (3,1) state: & lets assume we have a specific policy π s.t $\pi((3,1)) = N$



$$V^{\pi}((3,1)) = \underbrace{R((3,1)) + \gamma\left[0.8V^{\pi}((3,2)) + 0.1V^{\pi}((2,1)) + 0.1V^{\pi}((4,1))\right]}_{\text{Bellman's equation for state (3,1) with policy } \pi}$$

Similarly, for all 11-states, I can write $V^{\pi}(s')$.

i.e $V^{\pi}((1,1)) = \cdots$

$V^{\pi}((2,1)) = \cdots$

$\vdots$

$V^{\pi}((4,3)) = \cdots$

This means we will have 11-variable, $[V^{\pi}(1,1) \cdots V^{\pi}((4,3))]$ and 11-constraints of linear system.

Optimal value function :

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Bellman's Eq/s: (Bellman's eq/ for best value function)

$$V^*(s) = R(s) + \max_{a} \gamma \sum_{s'} P_{sa}(s') V^*(s')$$

$$\pi^*(s) = \arg\max_{a} \sum_{s'} P_{sa}(s') V^*(s')$$

The consequence of above equations will be that π^* is the optimal policy.

VALUE Iteration

This is an algorithm to compute optimal policy using Bellman's Equation.

Initialize $V(s)=0 \quad \forall s$

repeat until convergence { For every s, update

$$V(s) := R(s) + \max_{a} \gamma \sum_{s'} P_{sa}(s') V(s')$$

This will make $V(s) \longmapsto V^*(s)$

There could be two methods of updating $V(s)$

i) Synchronous Update:

$$V(s) := \underbrace{R(s) + \max_{a} \gamma \sum_{s'} P_{sa}(s') V(s')}_{\text{CALC RHS for all states}}$$

then $V := B(V)$

ii) Asynchronous Update:
Use the new value of $V(s)$ in update equation

VALUE Iteration example

3	s_8	s_9	s_{10}	+1 s_{11}
2	s_5	▨	s_6	-1 s_7
1	s_1	s_2	s_3	s_4
	1	2	3	4

Init $V(s)=0$

$V(1,1)=0.0$
$V(2,1)=0$
$V(3,1)=0$
$V(4,1)=0$
$V(4,2)=0$
$V(1,2)=0$
$V(3,2)=0$
$V(1,3)=0$
$V(2,3)=0$
$V(3,3)=0$
$V(4,3)=0$

OR $V(s_1)=V(s_2)=\dots=V(s_{11})=0.0$

SYNCHRONOUS UPDATE

$$\begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_{11}) \end{bmatrix} = \begin{bmatrix} R(s_1)+\max_a \gamma \left[P_{s_1a}(s_5)V(s_5)+\dots\right] \\ \vdots \end{bmatrix}$$

Async Update

$$\boxed{V(s_1)} = R(s_1)+\max_a \gamma \left[P_{s_1a}(s_5)V(s_5)+ P_{s_1a}(s_2)V(s_2)\right]$$

$$V(s_2) = R(s_2)+\max_a \gamma \left[P_{s_2a}(s_3)V(s_3) + P_{s_2a}(s_1)\boxed{V(s_1)}\right]$$

$V(s_3) =$ ⋮

$V(s_{11}) =$

To use this algorithm, lets use $\gamma = 0.99$

3				+1
2		///		-1
1				
	1	2	3	4

Lets run value iteration on MDP above. The numbers we get for V^* are as follows.

$V^* =$

3	.86	.90	.93	+1
2	.82	///	.69	-1
1	.78	.75	.71	.49
	1	2	3	4

If you plug this V^* into the formula

$$\pi^*(s) = \arg\max_a \sum_{s'} P_{sa}(s') V^*(s')$$

$\pi^* =$

3	→	→	→	+1
2	↑	///	↑	-1
1	←	←	←	←
	1	2	3	4

To summarize, we run value iteration to get V^* (as shown above: table of numbers) and then I compute optimal policy using $\pi^*(s)$ formula.

In π^*, lets take a look at state (3,1). why do we select action "W" A: {W, N, E, S}.

(58:40)
L#15

At (3,1)

Moving west : $\sum_{s'} P_{sa}(s') V^*(s')$

$$= 0.8 \times 0.75 + 0.1 \times 0.69 + 0.1 \times 71 = 0.740$$

Moving North $= \sum_{s'} P_{sa}(s') V^*(s')$

$$= (0.8 \times 0.69) + (0.1 \times 0.75) + (0.1 \times 0.49) = 0.676$$

If Robot is at (3,1), then the expected payoff of moving west (0.74) is higher than moving north (0.676) That's why $\boxed{\pi((3,1)) = W}$

At (3,1):

Moving W: state (2,1) $\sum_{s'} P_{sa}(s') V^*(s')$

$= \underset{\text{state: }(2,1)}{(0.8 \times 0.75)} + \underset{\text{state: }(3,2)}{(0.1 \times 0.69)}$

$+ \underset{\text{state: }(3,1)}{(0.1 \times 0.71)}$

*Moving North (3,2)

$= \sum_{s} P_{sa}(s') V^*(s')$

$= \underset{s:(3,2)}{(0.8 \times 0.69)} + \underset{s:(2,1)}{(0.1 \times 0.75)} + \underset{s:(4,1)}{(0.1 \times 0.49)}$

Another algorithm to find optimal policy for MDP is "Policy Iteration".

Policy Iteration

Initialize π randomly

Repeat:

1) Let $V := V^{\pi}$ (Solve Bellman's Eq) (solve system of linear equations 11 unknowns & 11 constants)
2) Let $\pi(s) := \arg\max_{a} \sum_{s'} P_{sa}(s') V(s')$

$V \mapsto V^*$ & $\pi \mapsto \pi^*$

Step(1) in Policy iteration is expensive. If you have N state, then you have to solve a linear system of N ~~unknow~~ variables with N constraints.

If your MDP is very large (≈10 Million states), then its better to use value iteration than policy iteration as policy iteration would need to solve system of around ≈10 million operations

* Now you can use policy iteration or value iteration to ~~solve~~ compute optimal policy for a given MDP.

Now, what if you don't know state-transition probabilities (P_{sa})?

→ MDP: S, A, P_{sa}, γ, R (we will estimate $P_{sa}(s')$ from data)

$$P_{sa}(s') = \frac{\#\text{ times took action "a" in } s,\text{ got to } s'}{\#\text{ times took action "a" in } S}$$

$$P_{sa}(s') = \left(\text{or } \frac{1}{|S|} \text{ if } \frac{0}{0}\right)$$

Putting it together

```
Repeat: {
    * Take actions using $\pi$ to get experience in MDP
    * Update estimates of $P_{sa}$
    * Solve Bellman's eq using VI to get $V$
    * Update $\pi(s) = \arg\max_a \sum_{s'} P_{sa}(s') V(s')$
}
```

Solving MDPs for those problems where P_{sa} is not available