REINFORCEMENT LEARNING

Types of Learning

Supervised learning
- Given: training data + desired outputs (labels)

Unsupervised learning
- Given: training data (without labels)

Semi-supervised learning
- Given: training data + some labels

Reinforcement learning
- Given: observations and occasional rewards as the agent performs sequential actions in an environment

No dataset in advance
You receive your data as agent performs sequential actions in environment.

How Reinforcement Learning is Different

The aim of RL is to make sequential decisions in an environment:

- Driving a car
- Cooking
- Playing a videogame
- Controlling a power plant
- Treating a trauma patient
- Displaying online ads to a user

Reward
actions good
action bad

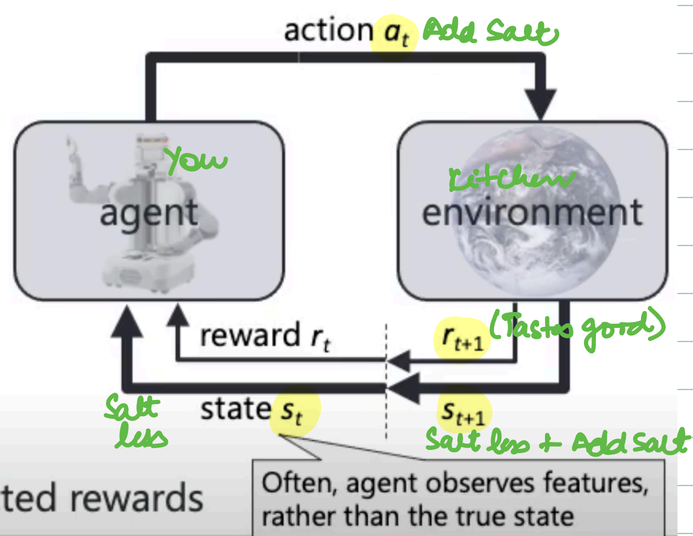
How to learn to do these things?

- RL assumes only occasional feedback, such as a tasty meal, or a car crash, or points in a video game.
- RL aims to use this feedback to learn through trial and error, as cleverly as possible. → to minimize the no. of trials.

Reinforcement Learning

Main idea:

- Agent receives observations (state $s_t \in S$) and feedback (reward r_t) from the world
- Agent takes action $a_t \in A$
- Agent receives updated state s_{t+1} and reward r_{t+1}



- Agent's goal is to maximize expected rewards

Goal of RL is to learn a policy $\pi(s): S \to A$ for acting in the environment

Policy
Mapping π
$\pi: S \to A$
maximize reward

Best Policy: optimal Policy π^*

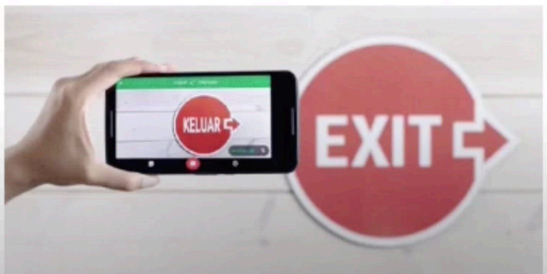
How Reinforcement Learning is Different

Characteristics of RL Problems:

- **No supervision,** only (occasional) rewards as feedback
- **Sequential decision making** ⟹ Data is generated as sequences (not i.i.d) → independent & identical distribution.
- **Training data is generated** by the learner's own behavior

When do we not need to worry about sequential decision making?

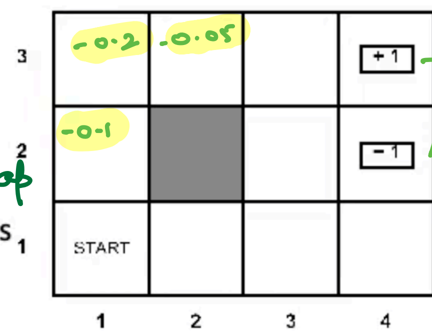
When your system is making a single isolated decision that does not affect future decision, e.g. classification, regression



Reinforcement Learning Examples:
Robotics, Autonomous Driving, Mario

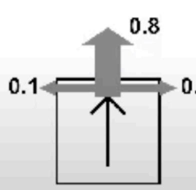
MARKOV DECISION PROCESS

Grid World



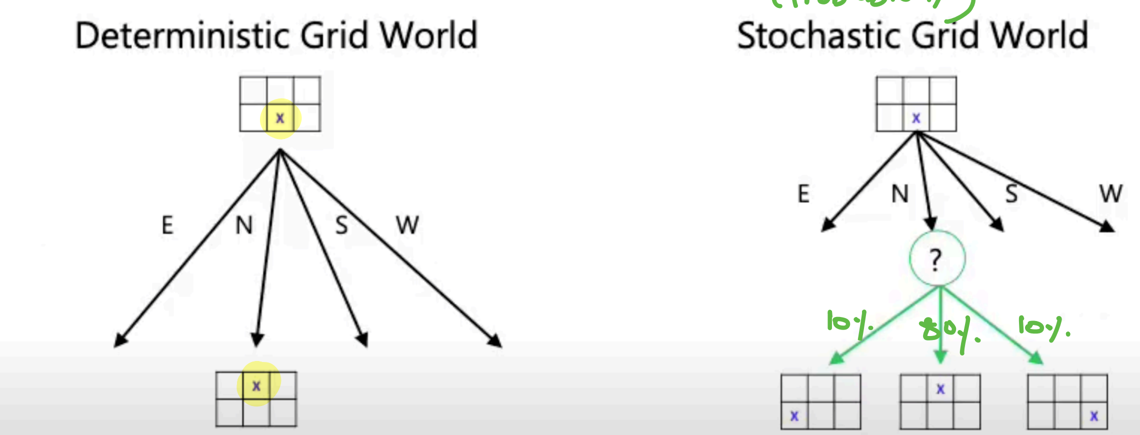
Terminal state.

- Agent operates in a grid with solid and open cells
- Each timestep, the agent receives a small negative "living" reward → penalty Aim: complete task asap
- There are two bigger magnitude rewards at terminal states that end an episode
- The agent can move North, East, South, West
  - The agent remains where it is if it tries to move into a solid cell or outside the world
  - The chosen action succeeds 80% of the time (for an open cell)
  - 10% of the time, the agent ends up 90° off
  - Another 10% of the time, the agent ends up -90° off
    - For example, an agent surrounded by open cells and moving North will end up in the northern cell 80% of the time, in the eastern cell 10% of the time, and in the western cell 10% of the time



- Goal: maximize sum of rewards (i.e., maximize expected utility)

State Transition Diagram

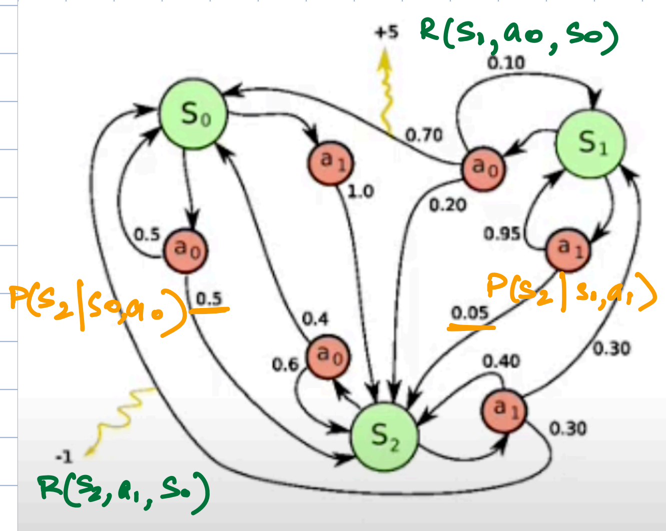


State Transition Diagram:

Markov Decision Process

An MDP (S, A, P, R) is defined by:

- Set of states $s \in S$ $\quad S = \{S_0, S_1, S_2\}$
- Set of actions $a \in A$ $\quad A = \{a_0, a_1\}$
- Transition function $P(s' \mid s, a)$
  - Probability $P(s' \mid s, a)$ that a from s leads to s'
  - Also called the "dynamics model" or just the "model"
- Reward function $R(s, a, s')$ or $R(s)$ (action, state)



we typically donot know $P(\cdot)$ and $R(\cdot)$

We donot know state transition diagram in advance, we have to learn it through trial and error. In the process of learning through trial and error we will encounter samples from the transition diagram.

why the name Markov?

The Markov Property: given the present, the future and the past are independent

- i.e., Everything you need to know about the past is included in the present state

- For MDPs, the Markov property means that:

$$P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t, \underbrace{S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots, S_0 = s_0}_{\text{Independent of past states and actions}})$$
$$= P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t)$$

Markov Assumption

Solving the MDP

To solve an MDP (S,A,P,R) means to find the optimal policy $\pi^*(s): S \to A$

→ Policy is a mapping from state to Action

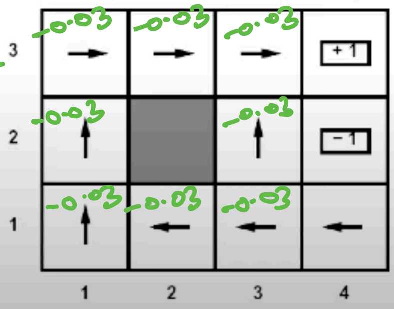
- "Optimal" $\Longrightarrow$ Following π^* maximizes total reward/utility (on average)

In RL, P and R are assumed unknown. But let's first assume that we do know the whole MDP

Example optimal policy π^*

arrow marks: actions

Living reward

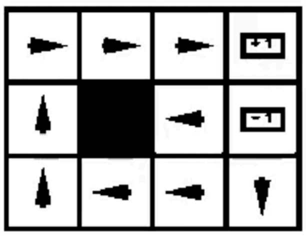


Optimal policy when
$R(s, a, s') = -0.03$
for all non-terminal states

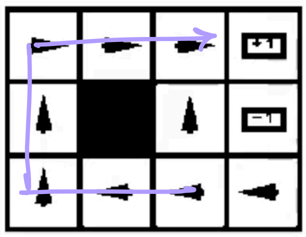
Example Optimal Policies

If your reward changes then policy will also change

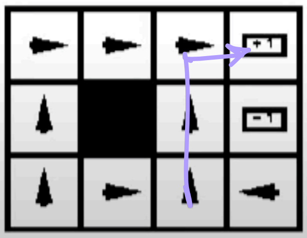
(Policy)



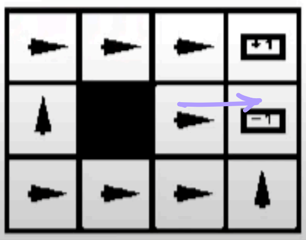
$R(s) = -0.01$



$R(s) = -0.03$



$R(s) = -0.4$

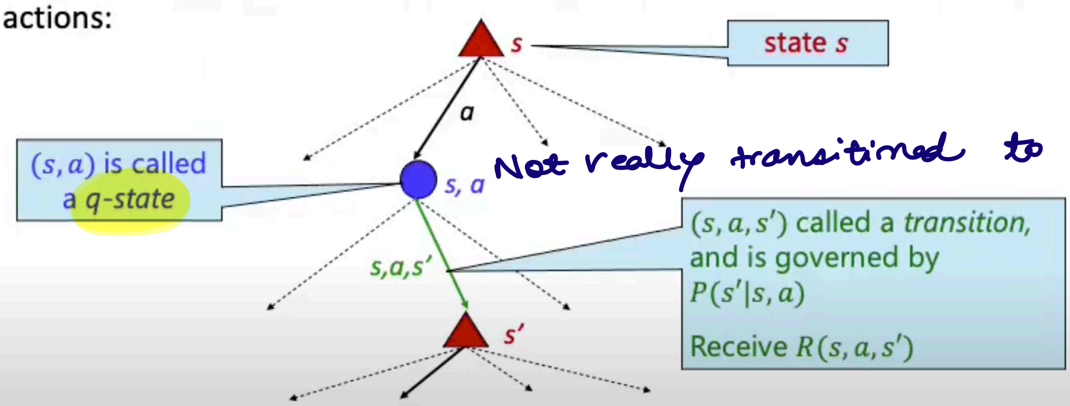


$R(s) = -2.0$

Optimal policy is sensitive to reward fxn.

MDP Search Trees

- Each MDP state has an associated tree of future outcomes from various actions:

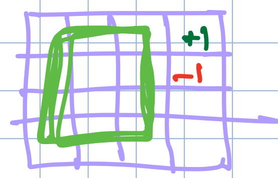


Not really transitioned to a new state.

Defining Utility

- At each step, agent chooses an action to maximize expected rewards
  - So, we must consider utility over sequences of rewards
    $[r_t, r_{t+1}, r_{t+2}, \dots, r_\infty]$ → upper bound

Problem: infinite sequences yield infinite rewards



Discounted Rewards

Idea: (uncertain) future rewards are worth exponentially less than current rewards

Future rewards are discounted by $0 < \gamma < 1$:

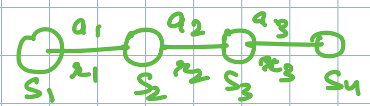
$\gamma = 1$ would correspond to sum of all rewards (called additive utility)

$$\text{Utility} \quad U([r_1, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

$\gamma = 0.9$: future rewards eq. importance

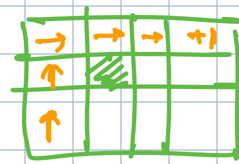
$\gamma = 0.1$: immediate steps are more important

γ controls the horizon of focus; smaller γ means a shorter horizon – more of a focus on the present



$r_1 + r_2 + r_3$

$\gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3$



$-0.3 \; -0.3 \; -0.3 \; -0.3 \; -0.3 \; +1$

Utility | Reward $= \gamma^0(-0.3) + \gamma^1(-0.3) + \gamma^2(-0.3) + \gamma^3(0.3) + \gamma^4(-0.3) + \gamma^5(1)$

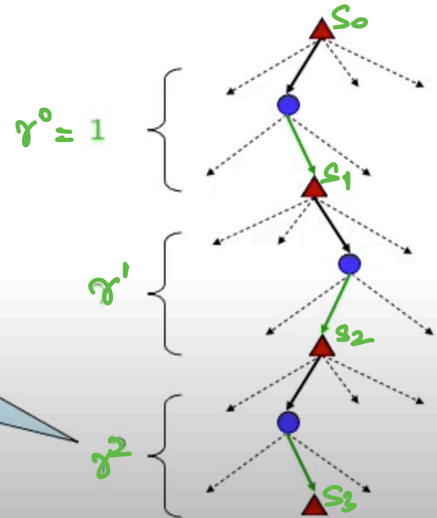
Discounted Rewards

Idea: (uncertain) future rewards are worth exponentially less than current rewards

Future rewards are discounted by $0 < \gamma < 1$:

$$U([r_1, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

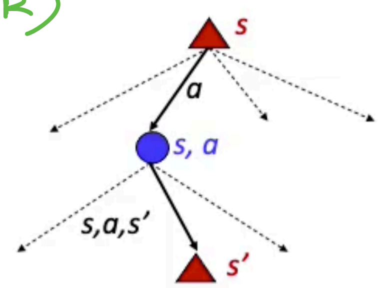
Future rewards matter less to the decision than more recent rewards



Keep diluting the contribution of future transitions

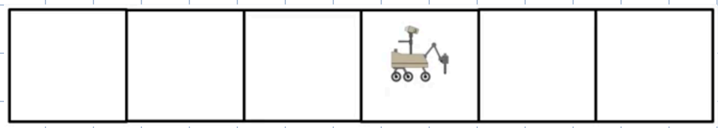
Recap: Defining MDPs

- Markov decision processes: → (S, A, P, R)
  - States S (and start state s_0)
  - Actions A
  - Transitions $P(s'|s, a)$
  - Rewards $R(s, a, s')$ (and discount γ)



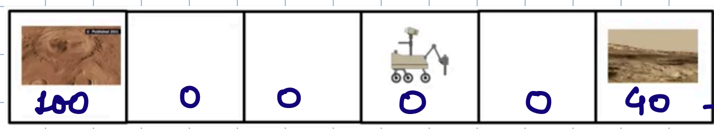
- MDP quantities so far:
  - Policy = Choice of action for each state $\pi : S \to A$
  - Utility (or return) = sum of discounted rewards

Mars Rover Example



State: 1 2 3 4 5 6

Terminal States = 1, 6
(After reaching terminal state you stop)



→ Rewards

State: 1 2 3 4 5 6

Action ← left → right

no discounting

States:

4 3 2 1 : 0 + 0 + 0 + 100 = 100
4 5 6 : 0 + 0 + 40 = 40
4 5 4 3 2 1 : 0 + 0 + 0 + 0 + 0 = 100

Rewards

RETURN (UTILITY) (Discounted Reward)

$\gamma^0 \quad \gamma^1 \quad \gamma^2 \quad \gamma^3$

4 3 2 1 : $(0.9)^0 0 + (0.9)^1 0 + (0.9)^2 0 + (0.9)^3 100$

$\gamma = 0.9$ $= 0 + 0 + 0 + 0.729 \times 100 = 72.9$

$$\boxed{\text{Return: } R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \text{ (until terminal state)}}$$

Rewards

$\gamma = 0.5$

4 3 2 1 : $0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$

: 12.5

Example of Return

Return depends on Reward
Reward depends on Action

} Return depends on action you take

100	0	0	0	0	40
1	2	3	4	5	6

Policy π : every state ← action

$\gamma = 0.5$

100	50 ←	25 ←	12.5 ←	6.25 ←	40
100	0	0	0	0	40
1	2	3	4	5	6

→ $V^{\pi}(s)$ (state value function)

You are state 2 & you take ← move how much is the utility?

$(0.5)^0 \cdot 0 + (0.5)^1 (100)$
$= 50$

$0 + (0.5)^1 0 + (0.5)^2 100$
$= 25$

$0 + (0.5)^1 0 + (0.5)^2 0 + (0.5)^3 100$
12.5

Policy π : every state → action

100	2.5 →	5 →	10 →	20 →	40
100	0	0	0	0	40
1	2	3	4	5	6

→ $V^{\pi}(s)$ (state value function)

$0 + (0.5) 0 +$
$(0.5)^2 40$

$0 + (0.5) 40$

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

Optimal Policy π^*

	50 ←	25 ←	12.5 ←	20 →	
100	0	0	0	0	40
1	2	3	4	5	6

} optimal policy: action that would give me maximum return

Policy π

$S \longrightarrow A$
State Action

$\pi(2)$: ←
$\pi(3)$: ←
$\pi(4)$: ←
$\pi(5)$: →

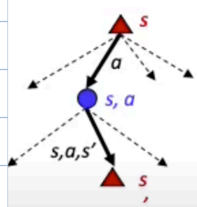
} π^*

State Value function & Action Value function

Solving MDPs: State Value Functions of Policies

↳ fxn of state and of policy that you are following

Given MDP (S, A, P, R, γ):



Value of a state s under policy π :

$V^\pi(s)$ = expected utility starting in s and acting according to π

$$V^\pi(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s\right)$$

Sequence of rewards generated by following π

$E(x) = \Sigma x P(x)$

Optimal value of a state s :

$V^*(s)$ = expected utility starting in s and acting optimally

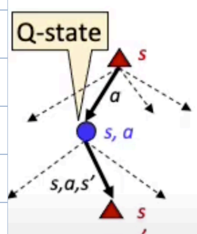
$$V^*(s) = V^{\pi^*}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s\right)$$

Rewards generated by following π^*

Solving MDPs: Action Value Functions of Policies

- It is also helpful to define action-value functions

stochastic



$E(X) = \Sigma x P(X)$

Q-value of taking action a in state s then following policy π :

$Q^\pi(s,a)$ = expected utility taking a in s and then following π

$$Q^\pi(s,a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_0 = s, A_0 = a\right)$$

Optimal Q-value of taking action a in state s : $Q^*(s,a) = Q^{\pi^*}(s,a)$

π^* can be greedily determined from Q^*: $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$ } action that maximizes $Q^*(s,a)$ will be the action output by optimal policy.

$\gamma^0 R_1 + \gamma^1 R_2 + \gamma^2 R_3 \ldots\ldots$

State Value function

Action Value function

$Q(s,a) =$ ⊙ start from state s

⊙ take action a

⊙ Behave optimally after that.

Return that you get?

$Q(2,\leftarrow)$ $Q(2,\rightarrow)$

100 100	50 12.5	25 6.25	12.5 10	6.25 20	40 40
100	0	0	0	0	40
1	2	3	4	5	6

$Q(s,a) = R(s)$

$\gamma = 0.5$

$$Q(2,\rightarrow) = R(2) + 0.5 \max_{a'} Q(3,a')$$

$$Q(2,\rightarrow) = 0 + 0.5 \max(25, 6.25)$$

$$Q(2,\rightarrow) = 0 + 0.5 \times 25 = 12.5$$

$$Q(4,\leftarrow) = R(4) + 0.5 \max_{a'} Q(3,a')$$

$$= 0 + 0.5 \max(25, 6.25)$$

$$= 12.5$$

Bellman Equation

$$\underbrace{Q(s,a)}_{\text{Action value fx}^n} = \underbrace{R(s)}_{\text{Reward at state s}} + \underbrace{\gamma \max_{a'} Q(s',a')}_{\text{Behave optimally after that}}$$

Utility Equation:

$$R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 \ldots$$

$$\underbrace{R_1}_{\text{Reward at state s}} + \gamma \underbrace{(R_2 + \gamma R_3 + \gamma^2 R_4 \ldots)}_{\max_{a'} Q(s',a')}$$

Bellman Equation

$E(X) = \sum x P(x)$

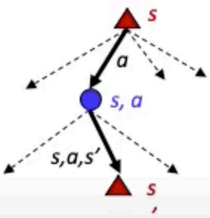
Solving MDPs: Bellman Equations

- The Bellman equations connect value functions at consecutive timesteps:

State value function

$$V^*(s) = \max_{a \in A} Q^*(s,a)$$

Optimal value of s is what we get by picking the optimal action



$$Q^*(s,a) = \sum_{s' \in S} \underbrace{P(s'|s,a)}_{\text{expected value over successor state s'}} \underbrace{[r(s,a,s') + \gamma V^*(s')]}_{\text{current reward + discounted future reward}}$$

Stochastic

Action Value function

- We can combine these together to get:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[r(s,a,s') + \gamma V^*(s')]$$

$Q(s,a) \longrightarrow$ start from state s, action a, behave optimally — Return.

S : current state

a : current action

s' : state you get after taking action a

a' : action that you take at state s'

$$Q(s,a) = \underbrace{R(s) + \gamma \max_{a'} Q(s',a')}_{\text{Deterministic}} \rightarrow V(s')$$

Bellman Eqn

In a 4x4 grid world, a robot starts at S (1, 1) and aims to reach the charging station C (4, 4). Each step incurs a reward of −2, and reaching C provides a reward of +20. The discount factor is $\gamma = 0.8$. Using the value estimates for neighboring states given in the table below, compute the expected returns for each action (up, down, left, right) from (1, 1) using the Bellman update equation. Determine the optimal action at (1, 1) and explain the impact of γ on the decision. [4] [CO6]

Table IV

State	(1,1)	(1,2)	(2,1)	(2,2)	(3,3)	(4,4)
Value	4.0	4.8	5.2	5.6	8.0	20.0

$\gamma = 0.8$

	1	2	3	4
1	Start S 4.0	4.8		
2	5.2	5.6		
3			6.0	
4				stop G 20.0

$Q(11, \rightarrow) = -2 + 0.8 * 4.8 = 1.84$

$Q(11, \downarrow) = -2 + 0.8 * 5.2 = 2.16$ ✓

$\left.\begin{array}{l} Q(11, \leftarrow) \\ Q(11, \uparrow) \end{array}\right\} = -2 + 0.8 * 4 = 1.2$

Impact of γ:

γ close to 1: emphasis on future as well

γ close to 0: emphasis to present steps.

VALUE AND POLICY ITERATION

Solving MDPs with known P and R:
Value and Policy Iteration.

—

VALUE ITERATION

Solving MDPs: Value Iteration

- Bellman Equation gives us a recursive definition of the optimal value:

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

Key Idea: solve iteratively via dynamic programming

- Start with $V_0(s) = 0$ for all states s
- Iterate the Bellman update until convergence:

$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

Example: Value Iteration

Bellman Update Rule: $V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$

Example MDP

	1	2	3	4
3	Rewards given when in terminal state			+1
2		■		−1
1	γ=0.9, living reward=0, noise=0.2			

V_0

	1	2	3	4
3	0	0	0	0
2	0	■	0	0
1	0	0	0	0

Start with $V_0(s) = 0$

V_1

	1	2	3	4
3	0	0	0	+1
2	0	■	0	-1
1	0	0	0	0

Example: Value Iteration

Bellman Update Rule: $V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$

0.72

Example MDP

	1	2	3	4
3	Rewards given when in terminal state			+1
2		■		−1
1	γ=0.9, living reward=0, noise=0.2			

Optimal action will be "right"

V_1

	1	2	3	4
3	0	0	0 →	+1
2	0	■	0	-1
1	0	0	0	0

V_2

	1	2	3	4
3	0	0	0.72	+1
2	0	■	0	-1
1	0	0	0	0

$$V_2(\langle 3,3\rangle) \leftarrow \sum_{s' \in S} P(s'|\langle 3,3\rangle, right)[r(\langle 3,3\rangle, right, s') + 0.9V_i(s')]$$

$$\leftarrow 0.8[0 + 0.9\times 1] + 0.1[0 + 0.9\times 0] + 0.1[0 + 0.9\times 0] = 0.72$$

$0.8(0+0.9\times1)+0.1(0+0.9\times0.72)+0.1(0+0.9\times0)$

best action:

a) 10% ↑, 80% →, 10% ↓

b) 80% ↑, 10% ←, 10% →

c) 10% ↑, 80% ←, 10% ↓

d) 10% ←, 10% →, 80% ↓

Example: Value Iteration

Bellman Update Rule: $V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$

Example MDP

	1	2	3	4
3	Rewards given when in terminal state			+1
2		■		−1
1	γ=0.9, living reward=0, noise=0.2			

V_2

10% ↑, 80% →, 10% ↓

	1	2	3	4
3	0	0	0.72	+1
2	0	■	0	-1
1	0	0	0	0

V_3

	1	2	3	4
3	0	0.52	0.78	+1
2	0	■	0.43	-1
1	0	0	0	0

- Information propagates outward from terminal states
- Eventually all states have correct value estimates

POLICY ITERATION

Value fxⁿ is associated with some particular policy
and
optimal value fxⁿ is associated with optimal policy.

In value iteration, we were dealing with optimal value fxⁿ but now we will compute value fxⁿ for some random policy.

$\pi : S \to A$

Policy Evaluation

- How do we calculate the V's for a fixed policy? Compute the value function corresponding to a policy
- Idea: Bellman updates for arbitrary policy:

$$V_0^{\pi}(s) = 0$$

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_i^{\pi}(s')]$$

No maximization here, we are just applying the action prescribed by the policy we are trying to evaluate.

(value iteration update rule)

$$V_{i+1}(s) \leftarrow \cancel{\max_{a \in A}} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

Policy Iteration: An Alternative to Value Iteration

Repeat steps until convergence:

1. **Policy evaluation:** keep current policy π fixed, find value function $V^{\pi_k}(\cdot)$ Finding the value fxⁿ for policy
   - Iterate simplified Bellman update until values converge:

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s,\pi_k(s))[R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s')]$$

Chooses actions according to π

2. **Policy improvement:** find the best action for $V^{\pi_k}(\cdot)$ via one-step lookahead find the best action corresponding to the value fxⁿ you currently updated.

$$\pi_{k+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_k}(s')]$$

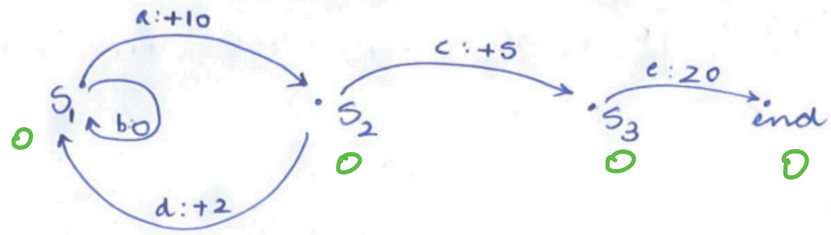
Finding the policy that would be optimal under the value fxⁿ.

If you do one iteration → Value Iteration

A car can operate in three modes: Electric Mode (S1), Hybrid Mode (S2), and Gasoline Mode (S3), each representing a state in the Markov Decision Processes (MDP). The car's system must decide which mode to switch to, aiming to optimize fuel efficiency and battery usage. Actions and their rewards are defined for each state: from S1, switch to S2 (+10) or stay in S1 (0); from S2, switch to S3 (+5) or back to S1 (+2); from S3, end the trip (+20). With a discount factor of 0.5 and initial state values of 0, perform one iteration of value iteration, calculating updated values for each state. [4] [CO1, CO3]

Q5:

a)



$\gamma = 0.5$ $V_0(S_1) = 0$ $V_0(S_2) = 0$ $V_0(S_3) = 0$

$$V_{i+1}(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V_i(s')\right]$$

transitions

$P(s'|s,a) = 1$ for $\forall$ actions

$V_1(S_1)$

action a:

$s' = S_2$ $10 + 0.5 \times 0 = 10$

action b:

$s' = S_1$ $0 + 0.5 \times 0 = 0$

$V_1(S_1) = \max(10, 0) = 10 \rightarrow$ action a.

$V_1(S_2)$

action c:

$s' = S_3$ $5 + 0.5 \times 0 = 5$

action d:

$s' = S_1$ $2 + 0.5 \times 0 = 2$

$V_1(S_2) = \max(5, 2) = 5 \rightarrow$ action c

$V_1(S_3)$

action e:

$s' =$ end $20 + 0.5 \times 0 = 20$

Temporal Difference and Q-Learning

We have seen how to solve MDP when P and R are known - by using value and Policy Iteration.

Temporal Differencing (TD)

Policy Evaluation
- Start with $V_0(s) = 0$
- Iterate until convergence: $V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s,\pi_k(s))[R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s')]$

Bellman eq^n written as update

How to extend this to when the functions $P(s'|s,a)$ and $R(s,a,s')$ are unknown and only revealed gradually through experience?

By taking action in environment we get more info about these 2 unknown quantities.

Every time you take action a from state s, you get a sample from the unknown $P(s'|s,a)$ and the corresponding reward $R(s,a,s')$

Temporal Differencing (TD)

$E(X) = \sum P(x)x$

Expectation of [ ] under P.

Policy Evaluation
- Start with $V_0(s) = 0$
- Iterate until convergence: $V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s,\pi_k(s))[R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s')]$

Idea: Treat the single sample you get as representative of the distribution, and apply an *incremental* update to reduce the "Bellman error":

Sample of V(s): $sample = R(s,\pi(s),s') + \gamma V_i^{\pi}(s')$

TD update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

α → learning rate

Doing this for each sample = computing the running average over samples

Everytime we take some action we get to see one sample.
We treat single sample as a proxy for entire distribution

TD treats every single sample you encounter as representative of distribution and you donot have to wait for large no. of interactions in environment to improve your

Consider an autonomous robot in a 4x4 warehouse grid, tasked with delivering items to a specific area. Each grid cell represents a different warehouse area. 'G' is the delivery area. The robot starts from area 1. The robot consumes energy (-0.2 reward) for each move and gains a significant energy saving (+5 reward) upon successful delivery to 'G'. The robot's route is: 1 → 2 → 3 → 4 → 8 → 12 → 11 → 15 → G. Using Temporal Difference Learning (TD(0)) with a learning rate (α) of 0.1 and a discount factor (γ) of 1, determine the updated efficiency value of area 1 after one delivery, starting from 0.

[4] [CO2]

-0.2 1	2 (-0.2)	3	4
5	6	7	8
9	10	11	12
13	14	15	G 5

Fig. 1

b)

$$V^{\pi}(S_t) = V^{\pi}(S_t) + \alpha \Big[\overbrace{R(S_t, \pi(s), S_{t+1}) + \gamma V^{\pi}(S_{t+1})}^{\text{sample}} - V^{\pi}(S_t) \Big]$$

$\alpha = 0.1 \qquad \gamma = 1 \qquad R = -0.2$

$S_1 \to S_2$: $V(1) = 0 + 0.1\,(-0.2 + 1 \times 0 - 0) = \underline{-0.02}$

$S_2 \to S_3$: $V(2) = 0 + 0.1\,(-0.2 + 1 \times 0 - 0) = \underline{-0.02}$

$S_3, S_4 \dots S_{15}$ $\}$ $V(S) = \underline{-0.02}$

$S_{15} \to G$: $V(15) = 0 + 0.1\,(5 + 1 \times 0 - 0) = \underline{0.5}$

Q Learning → Action value fx^n.

Beyond Policy Evaluation: Learning $Q^*(s,a)$

Recall Bellman equation for optimal Q*:

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

} Bellman used for Q.

The corresponding Q-value iteration equation (analogous to the state value iteration) would be:

$$Q_{i+1}(s) \leftarrow \sum_{s' \in S} P(s' | s,a) \left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a') \right]$$

Again, this requires access to $P(s'|s,a)$ and $R(s,a,s')$, of which we only have samples from experience.

Apply the TD trick to this?

Q Learning

Q-value iteration: $Q_{i+1}(s) \leftarrow \sum_{s' \in S} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$

The TD Version: Treat the single sample you get as representative of the distribution, and apply an *incremental* update to reduce the "Bellman error":

- Execute a single action a from state s and observe s' and R:

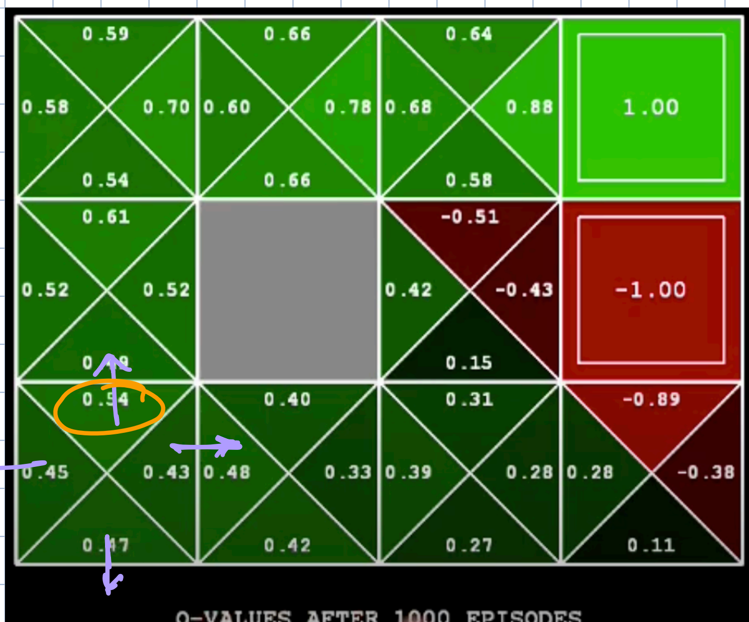
$$sample = R(s,a,s') + \gamma \max_{a'} Q^*(s',a')$$

- So, the incremental TD update is:

sample

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(\underbrace{R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)}_{\text{Bellman error}}\right)$$

This is called "Q-Learning".



Resources :

slides

https://www.youtube.com/playlist?list=PLYgyoWurxA_8ePNUuTLDtMvzyf-YW7im2
http://incompleteideas.net/book/ebook/ → Book
https://www.coursera.org/learn/unsupervised-learning-recommenders-reinforcement-learning

Mars Rover example.