REINFORCEMENT LEARNING

Types of Learning

Supervised learning

- Given: training data + desired outputs (labels)

Unsupervised learning

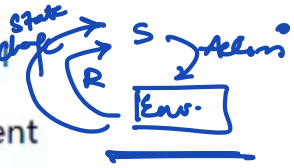
- Given: training data (without labels)

Semi-supervised learning

- Given: training data + some labels

Reinforcement learning

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



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

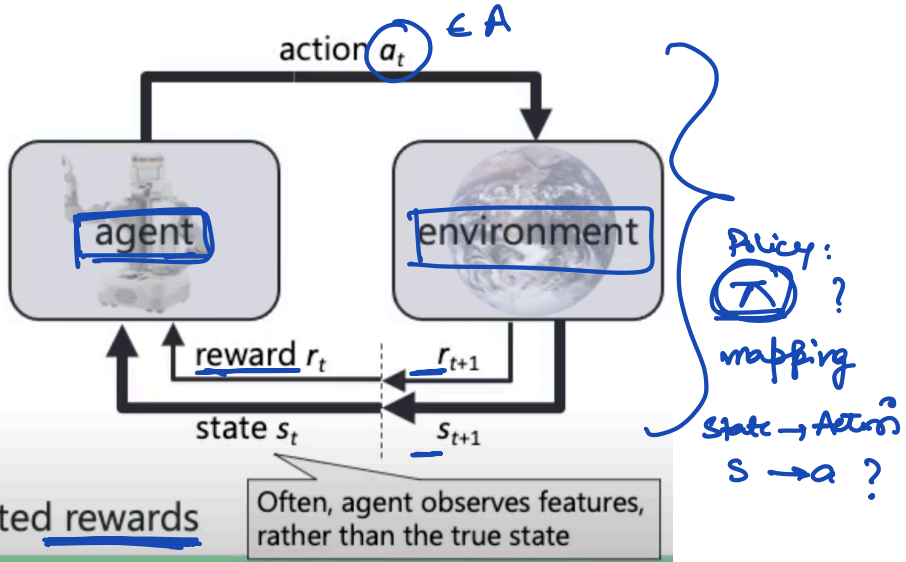
How to learn to do these things?

- 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.

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 $\pi(s): S \rightarrow A$ for acting in the environment

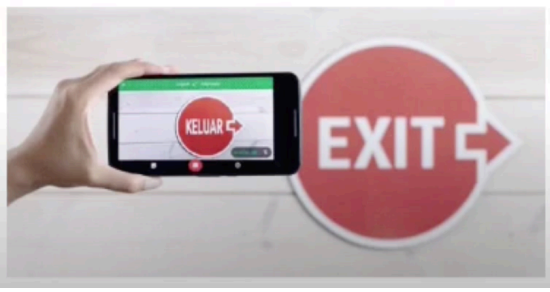
How Reinforcement Learning is Different

Characteristics of RL Problems:

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

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

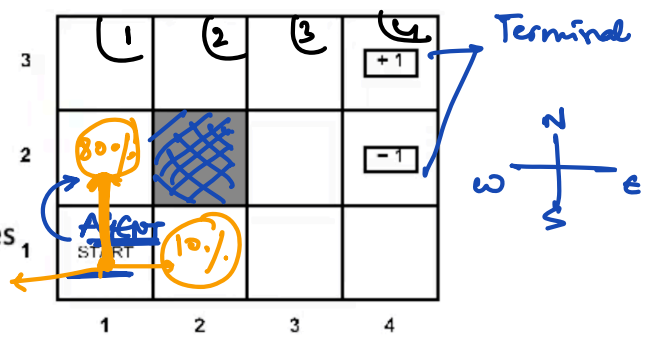
When your system is making a single isolated decision that does not affect future decision, e.g. classification, regression

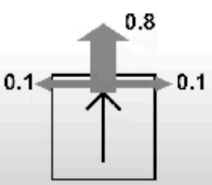


MARKOV DECISION PROCESS (MDP)

Grid World

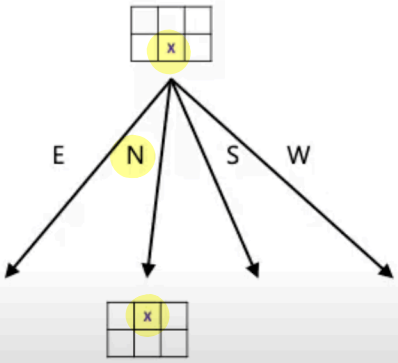
- Agent operates in a grid with solid and open cells
- Each timestep, the agent receives a small negative "living" reward
- There are two bigger magnitude rewards at terminal states that end an episode
- The agent can move North, East, South, West
  - The agent remains where it is if it tries to move 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)





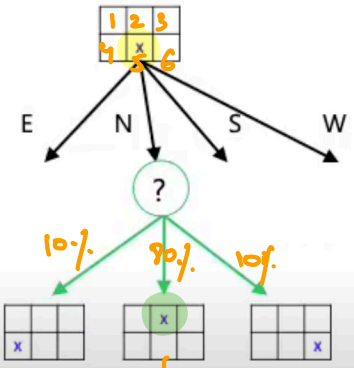
State Transition Diagram

Deterministic Grid World



(Probability)

Stochastic Grid World



Markov Decision Process

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

- Set of states $s \in S$
- Set of actions $a \in A$
- 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 $R(s)$

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

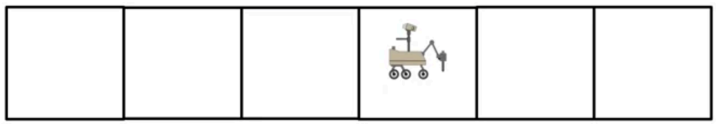
- i.e., Everything you need to know about the past is included in the present state

- For MDPs, the Markov property means that:

$$P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots, S_0 = s_0)$$
$$= P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t)$$

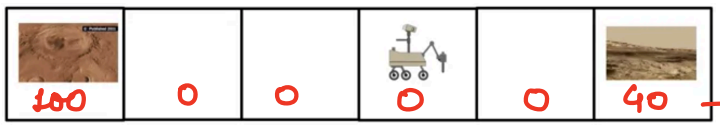
Independent of past states and actions

Mars Rover Example



State: 1 2 3 4 5 6

Terminal States: 1, 6



State: 1 2 3 4 5 6

Action ← left → right

States:
- 4 3 2 1 : 0 0 0 100
- 4 5 6 : 0 0 40
- 4 5 4 3 2 1 : 0 0 0 0 0 100

} Rewards

RETURN (UTILITY)

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

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

Return: $R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots$ (until terminal state)

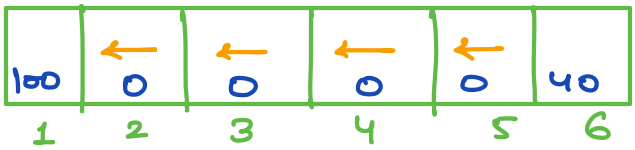
Rewards

$\gamma = 0.5$

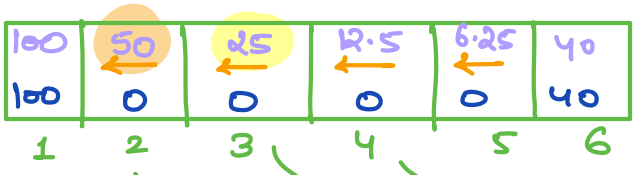
$4\ 3\ 2\ 1 : \quad 0 + (0.5)0 + (0.5)^2\,0 + (0.5)^3\,100$

$: \quad 12.5$

Example of Return



$\gamma = 0.5$

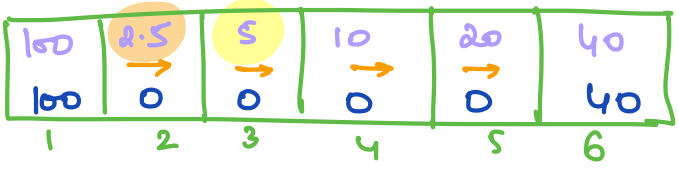


every state ← action

$0 + (0.5)(100)$

$0 + (0.5)0 + (0.5)^2 100$

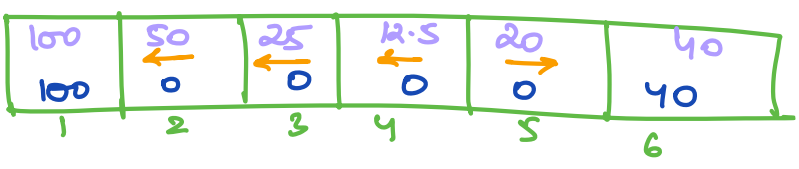
$0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$



every state → action

$0 + (0.5)40$

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



Policy: action that would give me maximum return

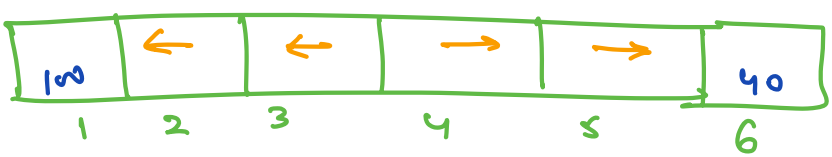
Policy π: state s → action a

$\pi(2) = \leftarrow$

$\pi(3) = \leftarrow$

$\pi(4) = \leftarrow$

$\pi(5) = \rightarrow$



Policy: Reach to the nearest terminal state in minimum moves.

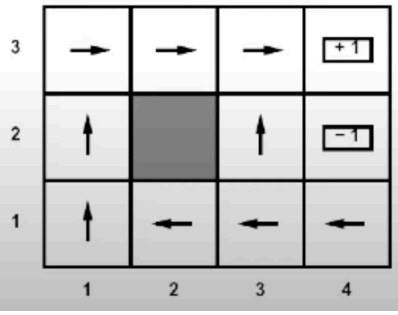
Solving the MDP

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

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

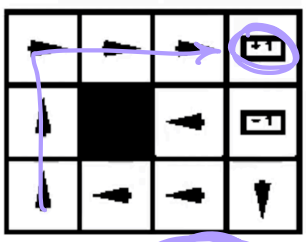
In RL, P and R are assumed unknown. But let's first assume that we do know the whole MDP

Example optimal policy π^*

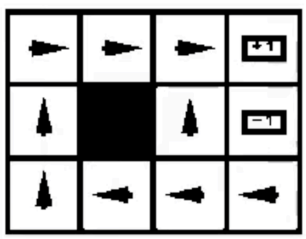


Optimal policy when
$R(s, a, s') = -0.03$
for all non-terminal states

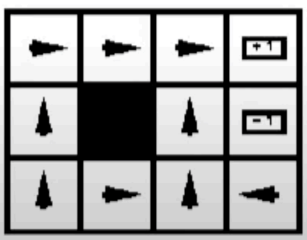
Example Optimal Policies



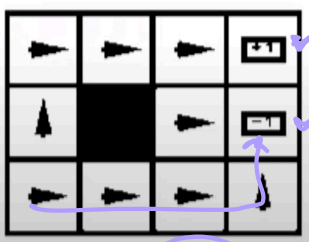
$R(s) = -0.01$



$R(s) = -0.03$

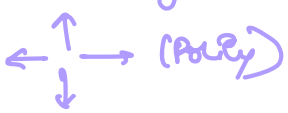


$R(s) = -0.4$



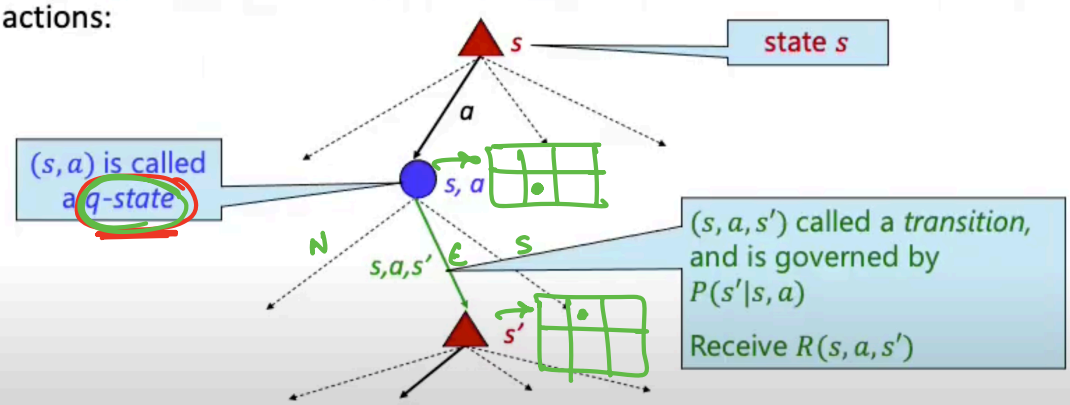
$R(s) = -2.0$

If your reward changes then policy will also change



MDP Search Trees

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



Discounted Rewards

γ → discounted reward

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)

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

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

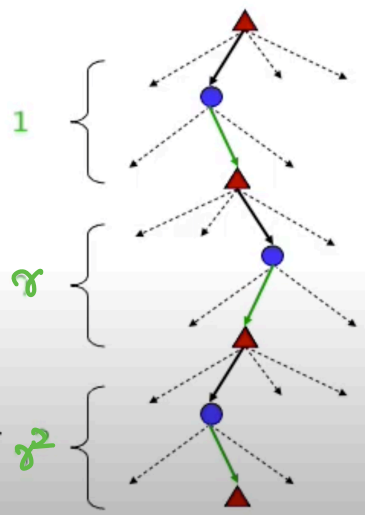
Discounted Rewards

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

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

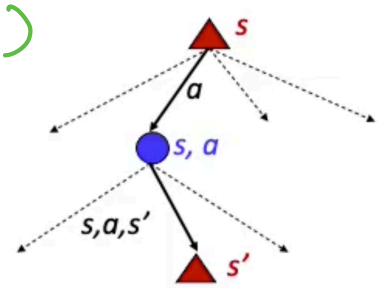
$$U([r_1, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

Future rewards matter less to the decision than more recent rewards



Recap: Defining MDPs

- Markov decision processes: → (S, A, P, R)
  - States S (and start state s_o)
  - 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
  - Utility (or return) = sum of discounted rewards

State Value function

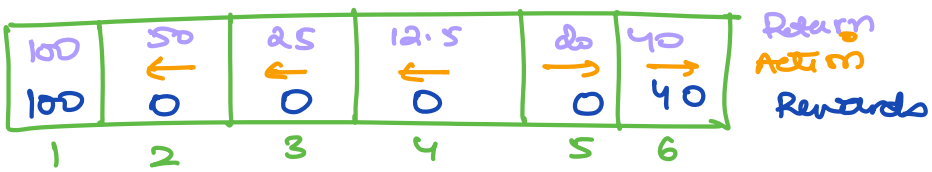
Action Value function

$Q(S,a)$ = ⊙ start from state S

⊙ take action a

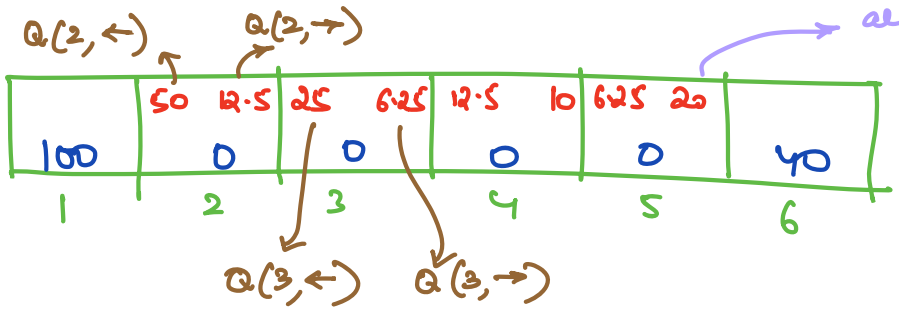
⊙ Behave optimally after that.

Return that you get?



$Q(2, \rightarrow) = 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$

$= 12.5$

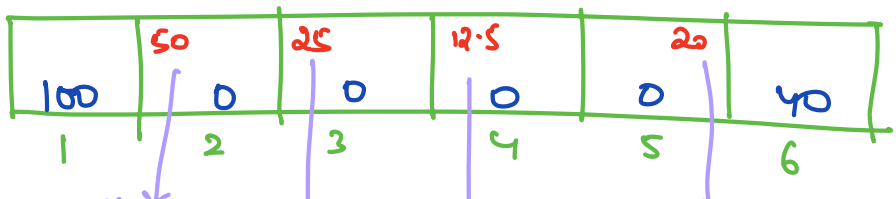


all $Q(S,\rightarrow)$ $Q(S,\leftarrow)$ Action value function

for all states

  for all actions ← →

    calculate return $Q(S,a)$



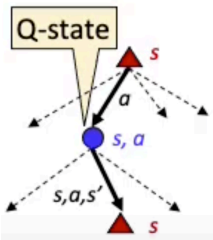
State value function

Mars Rover Example:

Deterministic

Solving MDPs: Action Value Functions of Policies

- It is also helpful to define action-value functions



stochastic

$S \rightarrow A$

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 : $Q^*(s,a) = Q^{\pi^*}(s,a)$

Stochastic

π^* can be greedily determined from Q^*: $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$

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

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

Bellman Equation

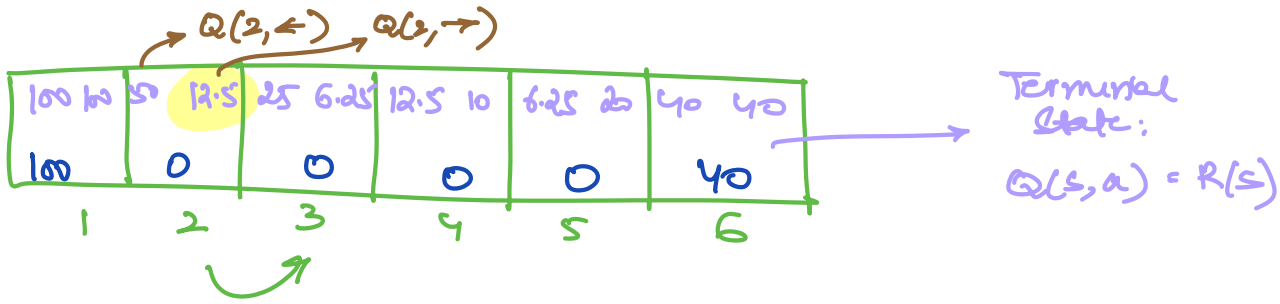
$Q(s,a) \rightarrow$ start from state s, action a, behave optimally } Return.

s : current state
a : current action

s' : state you get after taking action a
a' : action that you take at state s'

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

Deterministic



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

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

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

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

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

$= 12.5$

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

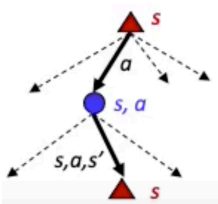
Solving MDPs: Bellman Equations

- 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



Action value function

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

expected value over successor state s'

current reward + discounted future reward

Stochastic

- We can combine these together to get:

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

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

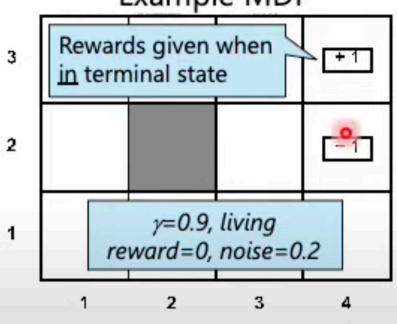
- Start with $V_0(s) = 0$ for all states s
- Iterate the Bellman update until convergence:

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

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



V_0

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

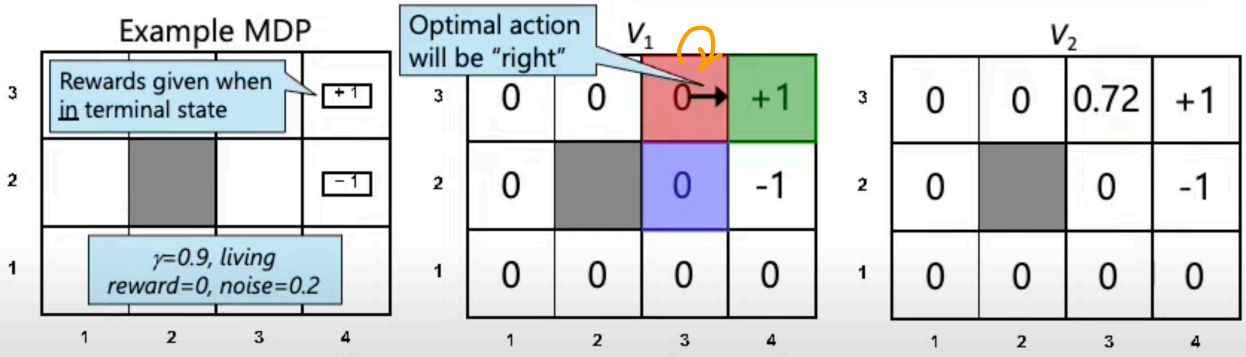
Start with $V_0(s) = 0$

V_1

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

Example: Value Iteration

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



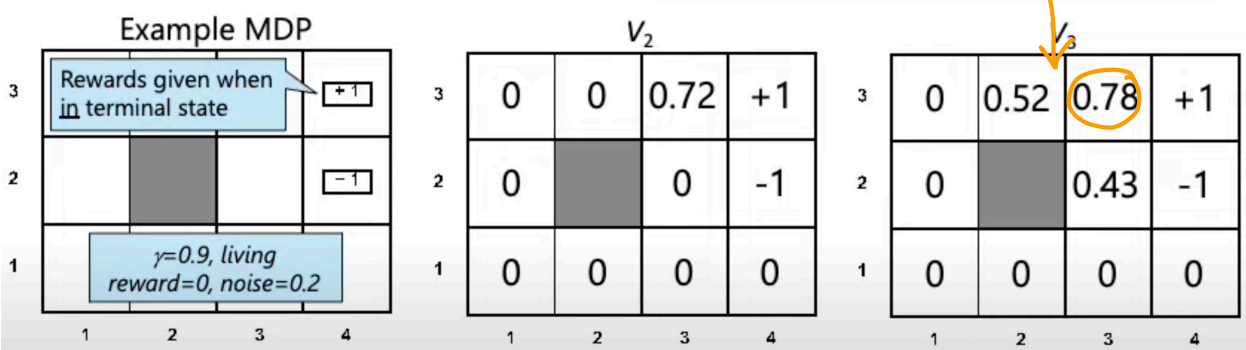
$V_2((3,3)) \leftarrow \sum_{s' \in S} P(s'|\langle 3,3\rangle, right)[r(\langle 3,3\rangle, right, s') + 0.9V_i(s')]$

$\leftarrow 0.8[0 + 0.9\times1] + 0.1[0 + 0.9\times0] + 0.1[0 + 0.9\times0] = 0.72$

" 0.1(0 + 0.9 × 0.72) "

Example: Value Iteration

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



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

POLICY ITERATION

Policy Evaluation

- How do we calculate the V's for a fixed policy?

- Idea: Bellman updates for arbitrary policy:

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

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

(value iteration update rule)

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

Policy Iteration: An Alternative to Value Iteration

Repeat steps until convergence:

1. **Policy evaluation:** keep current policy π fixed, find value function $V^{\pi_k}(\cdot)$
   - 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

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

Temporal Differencing (TD)

Policy Evaluation

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

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

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

Resources :

https://www.youtube.com/playlist?list=PLYgyoWurxA_8ePNUuTLDtMvzyf-YW7im2 (Slides)

http://incompleteideas.net/book/ebook/ (Book)

https://www.coursera.org/learn/unsupervised-learning-recommenders-reinforcement-learning

(Mars Rover Example)