Markov Decision Processes

Machine Learning

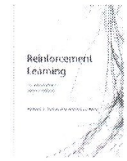
Daniele Loiacono

POLITECNICO MILANO 1863

Outline and References

- Outline
  - Agent-Environment Interface
  - Markov Decision Process
  - Policy
  - Value Functions
  - Optimality

- References
  - Reinforcement Learning: An Introduction [RL Chapter 3]
  - Fundamentals of Reinforcement Learning (Coursera)



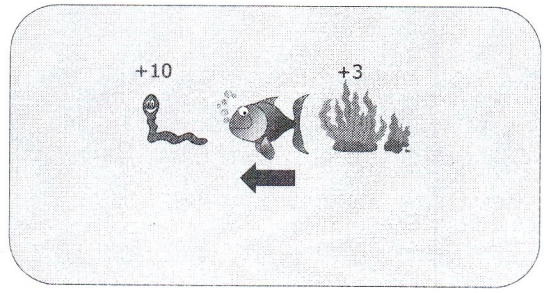
Agent-Environment Interface

Sequential Decision Making

- In sequential decision making...
  - ... we take a **sequence** of decisions (or **actions**) to reach the **goal**
  - ... the optimal actions are **context-dependent**
  - ... we generally do not have **examples** of correct actions
  - ... actions may have **long-term** consequences
  - ... **short-term consequences** of optimal actions might seem negative

What we have, instead, is a reward guidance that tells if we're doing good or not. This doesn't necessarily be connected to the very last action. (we may get rewarded only at the end of a sequence of actions, for example)

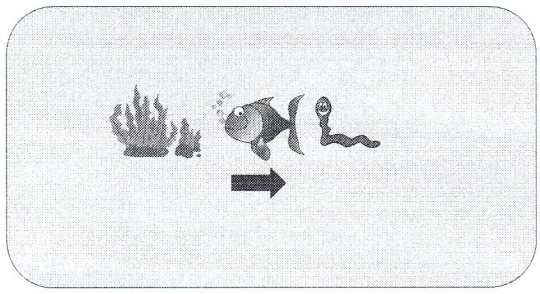
Sequential Decision Making



We are the fish and we want to it. We discover that eating the worm is rewarded +10 while eating the grass is rewarded +3. So, if the actions are ←, →, in this context we choose ←.

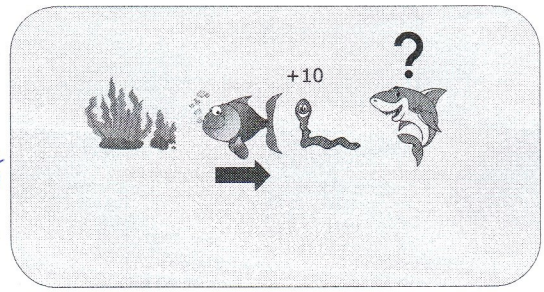
Sequential Decision Making

Obviously, if we change the context we have to adapt the actions to what we learned. In this context we go → because we learned that the warm is better.

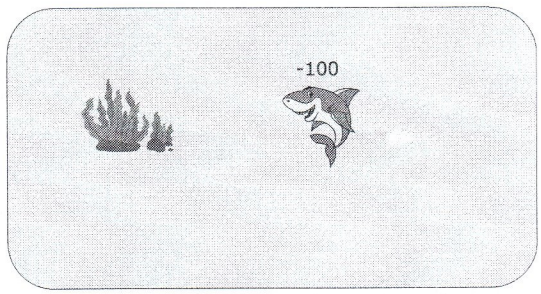


Sequential Decision Making

However we don't have to consider a single decision, we have to consider sequences of decisions. If here we look at the decisions one at the time we follow the warm to get +10 instead of +3 but then we end up facing a shark which "reward" is −100.

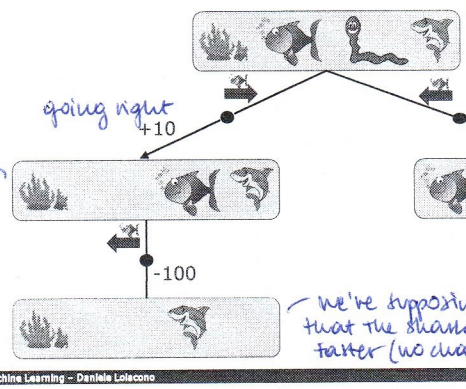


Sequential Decision Making



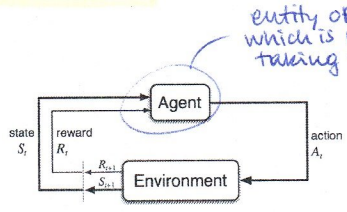
* we call our problem **ENVIROMENT** to stress that we cannot properly set our vbles and control everything

Sequential Decision Making



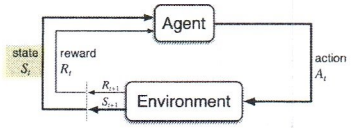
What seems to be a very good choice on the short term (turning right), because the initial reward is bigger, turns out to be very bad. The **TOTAL REWARD** (:= G) in the "going left" path is +3, instead, in the "going right" path is −90.

The Agent-Environment Interface

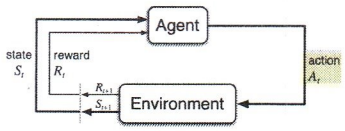


The agent perceives a state S_t (the current situation of the problem). Then the agent takes an action A_t (the action is chosen by the agent) which changes the environment. From this we get a new state, S_{t+1}, and a reward signal, R_{t+1}. The reward accounts only for the very last action that the agent took. (is not something that provides a feedback on how good is the action w.r.t. the ultimate goal)

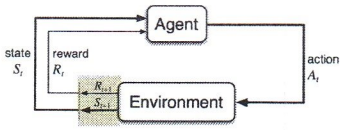
The Agent-Environment Interface



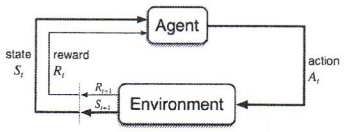
The Agent-Environment Interface



The Agent-Environment Interface



The Agent-Environment Interface



Agent and environment interact at discrete time steps: $t = 0, 1, 2, K$

Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t: $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}$

$$\ldots \; S_t \xrightarrow[A_t]{} R_{t+1}, S_{t+1} \xrightarrow[A_{t+1}]{} R_{t+2}, S_{t+2} \xrightarrow[A_{t+2}]{} R_{t+3}, S_{t+3} \xrightarrow[A_{t+3}]{} \ldots$$

State space: all the possible values (configurations) that the state can assume

action space: all the possible actions that the agent can take as a function of the CURRENT STATE (the actions available may depend on how the environment changed)

reward space: all the possible rewards that an action can generates (actually, all the actions)

Markov Decision Process

Markov Decision Process: One-Step Dynamics

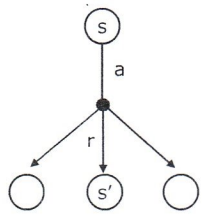
- **Markov Property:** future state (s') and reward (r) only depends on current state (s) and action (a)
  - It is not a limiting assumption, if see it as a property of state

- In a Markov Decision Process (MDP), the one-step dynamic can be described as:

$$p(s', r|s, a)$$

one step dynamic of the model

  - $p\colon \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,1]$
  - $\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$



Note that this is a distribution, there might **not** be deterministicity

Finite Markov Decision Processes

- When holds the Markov Property and the state and action sets are finite, the problem is a **finite Markov Decision Process**
- To define a finite MDP, you need to define:
  - **state and action sets**
  - one-step dynamics:

$$p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$$

probability of observing a state **s** and an action **a** leading into a new state **s'** and getting a reward **r**

  - we can also derive the next state distribution and expected reward as:

$$p(s'|s, a) \doteq \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

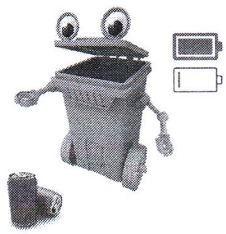
distribution of the future states based on the action that the agent takes and the current state

$$r(s, a) \doteq \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

expected reward for taking the action **a** in the state **s**

An Example Finite MDP: Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected



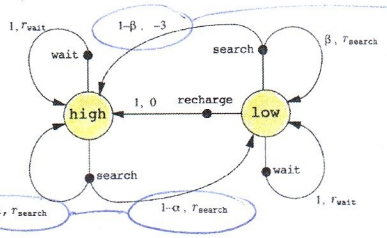
The only variables that define the states are: battery level high / low

An Example Finite MDP: Recycling Robot

Complete modelling:

$\mathcal{S} = \{\text{high}, \text{low}\}$
$\mathcal{A}(\text{high}) = \{\text{search}, \text{wait}\}$
$\mathcal{A}(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$

r_{search} = expected no. of cans while searching
r_{wait} = expected no. of cans while waiting
$r_{\text{search}} > r_{\text{wait}}$



if we are in low battery mode and we still perform the search, with probability 1-β we completely run out of battery, this brings an extremely bad reward: -3 (someone has to rescue the robot)

with probability 1-α the level of the battery becomes low (α and 1-α are the: $p(s'|s, a)$)

Return

- Agent should not choose actions on the basis of **immediate reward**
- In fact, **long-term** consequences are more important than **short-term** reward
- So, we need to take into account the **sequence of future rewards**
  - We define **return**, G_t, as a function of the sequence of future rewards

$$G_t \doteq f(R_{t+1} + R_{t+2} + R_{t+3} + \dots)$$

"the function of", actually we either use simple sum or weighted sum of the future rewards

  - To succeed, the agent will have to maximize the expected return $\mathbb{E}[G_t]$
- Different definition of return are possible:
  - Total reward
  - Discounted reward
  - Average reward
  - ...

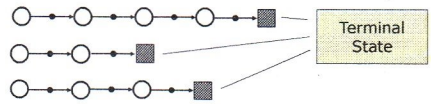
Episodic Task

- In **episodic task** the agent-environment interaction naturally breaks into chunks called **episodes**

for example: the chess game will eventually finish, it won't go on forever.
The previous case of the robot, instead, is an example of NON-EPISODIC task. There is no point where "we've done".

Episodic Task

- In **episodic task** the agent-environment interaction naturally breaks into chunks called **episodes**



- It is possible to maximize the expected **total reward**:

$$\mathbb{E}[G_t] = \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T]$$

Final time step

Continuing Tasks

- In **continuing task** the agent-environment interaction goes on **continually** and there are no terminal state
- The total reward is a sum over an **infinite sequence** and might not be finite:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_{t+k} + \ldots \stackrel{?}{=} \infty$$

- To solve this issue we can **discount** the future rewards by a factor γ $(0 < \gamma < 1)$:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{k-1} R_{t+k} + \cdots < \infty$$

!

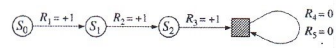
In this way we're giving more weight to the rewards that are close to us

- Thus, the expected reward to maximize will be defined as:

$$\mathbb{E}[G_t] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] \leq R_{max} \frac{1}{1-\gamma}$$

Unify Notation for Returns

- In episodic tasks, we number from zero the time steps for each episode
- We can design terminal state as **absorbing states** that always produce zero reward:



this state produces rewards that are zeros and brings deterministically the agent in the same state (kind of black hole).
This allows a single notation for both episodic tasks and continuous tasks.

- We can use the same definition of expected reward for episodic and continuing tasks:

$$\mathbb{E}[G_t] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

  - where $\gamma = 1$ can be used if an absorbing state is always reached
  - if $\gamma = 0$, agent would only care about immediate reward
  - As $\gamma \to 1$, agent would take future rewards into account more strongly

Goal and Reward

- A goal should specify **what** we want to achieve, not **how** we want to achieve it
- **The Reward Hypotesis:** *That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward).*
- Examples of reward design
  - goal-reward representation: 1 for goal, 0 otherwise
  - action-penalty representation: -1 for not goal, 0 once goal reached
- Challenges to reward hypothesis
  - How to represent risk-sensitive behavior?
  - How to capture diversity in behavior?

The reward is just a mean that we use in MDP to formalize our objective for the agent s.t. it's possible for the agent to learn the optimal strategy to reach the goal.

Policy

we need to formalize what is a strategy in MDP

What is a policy?

- A policy, at any given point in time, **decides** which action the agent selects
- A policy fully defines the **behavior** of an agent
- Policies can be:
  - Markovian / Non Markovian
  - Deterministic / Stochastic
  - Stationary / Non Stationary (changing or not during time)

Deterministic Policy

: deterministic function that maps each state in the state space into one (and only one) action

- In the simplest case the policy can be modeled as a function ($\pi: \mathcal{S} \rightarrow \mathcal{A}$):

$$\pi(s) = a$$

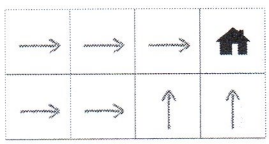
It has an input **s** and gives an output **a**. There is no stochasticity or distributions or ...

- Accordingly, the policy maps each state into an action
- This type of policy can be conveniently represented using a table

State	Action
s_0	a_1
s_1	a_0
s_2	a_0

- This is an example of deterministic policy in a gridworld environment

The state is where we are. In each state we have an action.



Stochastic Policy

- A more general approach is to model policy as function that maps each state to a probability distribution over the actions:

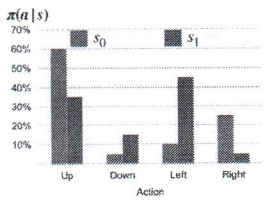
$$\pi(a|s)$$

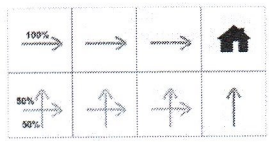
  - $\sum_{a \in \mathcal{A}(s)} \pi(a|s) = 1$
  - $\pi(a|s) \geq 0$

How?

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = a^* \\ 0 & \text{elsewhere} \end{cases}$$

- A stochastic policy can be used to represent also a deterministic policy
- This is an example of stochastic policy in a gridworld environment

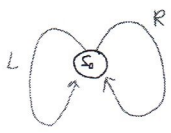




Markovian/Non Markovian Policy

MARKOV

The idea is always that the distribution of the actions depends only on the current states and not on what happened in the past



π_1: choose 50% R and 50% L

π_2: alternate R and L

- Are π_1 and π_2 both markovian?
- No! π_2 does not depend only from current state, so it is not a valid policy for us!
- However, we can overcome this limitation by extending the state definition (e.g., in this case including previous action)

Value Functions

Once we introduced the policies we are to find a method with which we evaluate how good a policy is. In order to do so, we introduce **value functions**

Value Functions

- Given a policy π we can compute the **state-value function** as:

$$V_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right]$$

  - it represents the expected return from a given state s, following policy π

It doesn't make sense to say how good is a policy if we do not take into account the state from which this policy is applied. How good is a policy is how good a policy can do from a given state.

- We can also compute the **action-value function** as:

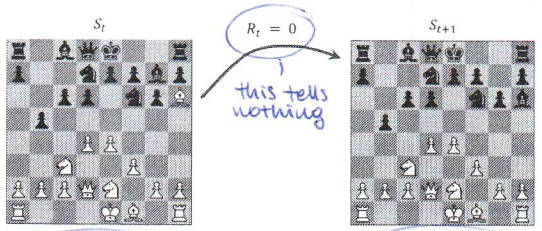
$$Q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a\right]$$

  - it represents the expected return from a given state s, when a given action a is selected and then policy π is followed

This is the way to make explicit the contribution of an action. Why is this interesting? We will use this when we'll try to change the policy to improve it. If we have a policy and we evaluate it we can try to improve it by starting from some state and trying other actions. $Q\pi(s,a)$ is convenient because tells how would be the return if instead of doing the action of the policy we took another action.

Why do we compute value functions?

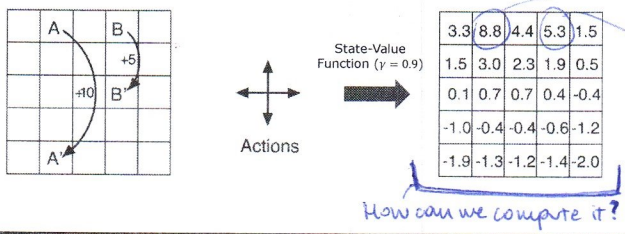
Because sometimes (often) the awards don't give informations (during the problem)



- Let assume the reward is +1 at if white wins and 0 elsewhere
- In this setting, $V_\pi(s)$ represents the probability of winning for white in s

State-Value Function: an example

- Let consider the following gridworld evironment:
  - Actions: `north`, `south`, `east`, `west` (deterministic dynamics)
  - Reward: -1 for bumping into the wall, positive from state A and B, 0 otherwise
  - Policy: random movement (25% north, 25% south, 25% east, 25% west)



How can we compute it?

this is less than the immediate reward (+10)
this is more than the immediate reward (+5)
Why?
Because A' is near the wall while B' is not. In the long term A' will also go against the wall, B' will reach the wall less probably

Bellman Expectation Equation

We use **RECURSION**

- The state-value function can again be **decomposed** into immediate reward plus discounted value of successor state:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s] \\ &= \sum_{a\in\mathcal{A}}\pi(a|s)\left(r(s,a) + \gamma\sum_{s'\in\mathcal{S}}p(s'|s,a)V_\pi(s')\right) \end{aligned}$$

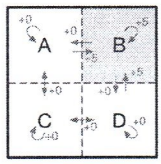
- The action-value function can be similarly decomposed:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= r(s,a) + \gamma\sum_{s'\in\mathcal{S}}p(s'|s,a)V_\pi(s') \\ &= r(s,a) + \gamma\sum_{s'\in\mathcal{S}}p(s'|s,a)\sum_{a'\in\mathcal{A}}\pi(a'|s')Q_\pi(s',a') \end{aligned}$$

If we stop here we don't get a recursive function

Why Bellman Equations?

- Let see how Bellman equations can be used in practice with an example



$$V_\pi(A) \doteq \mathbb{E}_\pi[G_t \mid S_t = A]$$
$$V_\pi(B) \doteq \mathbb{E}_\pi[G_t \mid S_t = B]$$
$$V_\pi(C) \doteq \mathbb{E}_\pi[G_t \mid S_t = C]$$
$$V_\pi(D) \doteq \mathbb{E}_\pi[G_t \mid S_t = D]$$

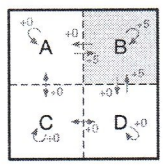
?

$\pi(a|s)$ 25% 25% 25% 25% = random policy

$\gamma = 0.7$

Why Bellman Equations?

- Let see how Bellman equations can be used in practice with an example



$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]$$
$$= \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right)$$

$$V_\pi(A) = \frac{1}{4}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(A)$$
$$V_\pi(B) = \frac{1}{2}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D)$$
$$V_\pi(C) = \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D) + \frac{1}{2}0.7V_\pi(C)$$
$$V_\pi(D) = \frac{1}{4}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(D)$$

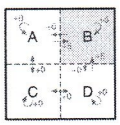
$V_\pi(A) = 4.2$
$V_\pi(B) = 6.1$
$V_\pi(C) = 2.2$
$V_\pi(D) = 4.2$

$\pi(a|s)$ 25% 25% 25% 25%

$\gamma = 0.7$

In A we have 4 possible actions: we can go right with prob. 1/4, we get 5 as immediate reward and $0.7\,V_\pi(B)$, we can go down with probability 1/4, we get 0 immediate reward and $0.7\,V_\pi(C)$ we can go up with prob. 1/4 and get $0.7\,V_\pi(A)$ or left with prob. 1/4 and get $0.7\,V_\pi(A)$. The last can be summed in $1/2 \cdot 0.7\,V_\pi(A)$.

Limitations of Bellman Equations



→ Solving 4 Linear Equations → V_π

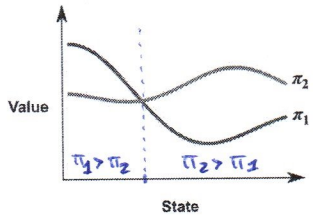
→ Solving ~~4~~ Linear Equations → V_π

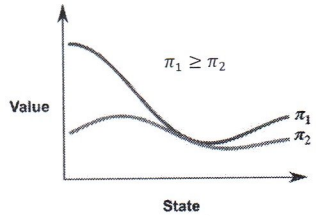
Optimality

Comparing two policies

We consider the value function:

Here we see that π_1 and π_2 are not one better than the other: in some states π_1 is better, in others π_2 is.





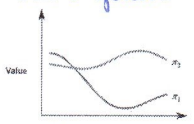
- We say that $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s), \quad \forall s \in \mathcal{S}$

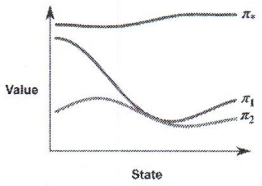
Optimal Policy

"deterministic" is to stress because if we consider also stochastic policies we'll have as possible policies

- For any Markov Decision Process, there exists always at least one **optimal deterministic policy** π^* that is better or equal to all the others ($\pi^* \geq \pi, \ \forall \pi$)
- **Sketch of proof**

Consider a generic situation:



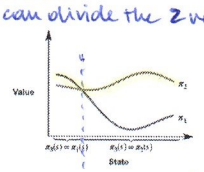


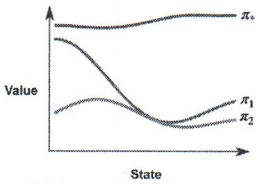
neither π_1 or π_2 are strictly the best policy

Optimal Policy

- For any Markov Decision Process, there exists always at least one **optimal deterministic policy** π^* that is better or equal to all the others ($\pi^* \geq \pi, \ \forall \pi$)
- **Sketch of proof**

We can divide the 2 regions:

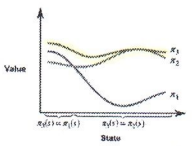


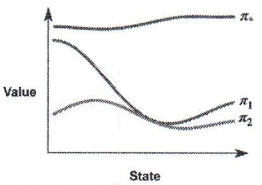


for each region we behave like the best π_i.
In this way we obtain at least ——.

Optimal Policy

- For any Markov Decision Process, there exists always at least one **optimal deterministic policy** π^* that is better or equal to all the others ($\pi^* \geq \pi, \ \forall \pi$)
- **Sketch of proof**





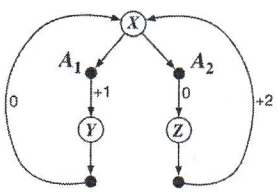
Often, when we combine two/more policies, we achieve a new policy π_3 which is strictly better.

IF this works for 2 policies, then we can apply it iteratively and find a final optimal policy π^* (better or equal to all the others)

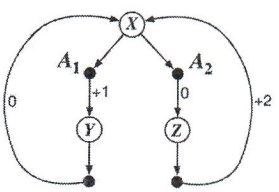
Optimal Policy: an example

3 states: X, Y, Z; in X we have 2 actions: A_1, A_2; in Y, Z we have just 1 action

The numbers represent (deterministic) rewards.



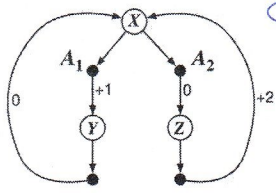
Optimal Policy: an example



$\pi_1(X) = A_1 \quad \pi_2(X) = A_2$

Suppose that we have 2 policies to compare. ⟹ let's compute the value function
(Because of how many actions we can choose, there ∃ ONLY two policies)

Optimal Policy: an example



$\pi_1(X) = A_1 \quad \pi_2(X) = A_2$

Computation of the value function:

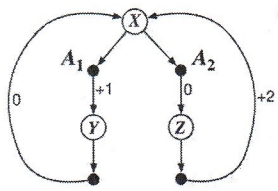
$\gamma = 0$ — this means immediate reward

$v_{\pi_1}(X) = 1$

$v_{\pi_2}(X) = 0$

$\Rightarrow \pi_1 > \pi_2$

Optimal Policy: an example



$\pi_1(X) = A_1 \quad \pi_2(X) = A_2$

$\gamma = 0$

$v_{\pi_1}(X) = 1$

$v_{\pi_2}(X) = 0$

$1 + \gamma \cdot 0 + \gamma^2 \cdot 1 + \gamma^3 \cdot 0 + \dots$

$\gamma = 0.9$

$v_{\pi_1}(X) = \sum_{k=0}^{\infty} (0.9)^{2k} = \frac{1}{1-0.9^2} \approx 5.3$

$v_{\pi_2}(X) = \sum_{k=0}^{\infty} (0.9)^{2k+1} * 2 = \frac{0.9}{1-0.9^2} * 2 \approx 9.5$

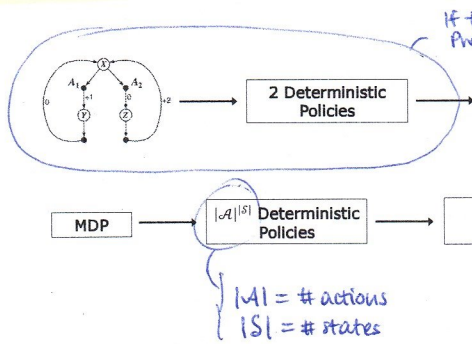
$\Rightarrow \pi_2 > \pi_1$

$1 \cdot 0 + \gamma \cdot 2 + \gamma^2 \cdot 0 + \gamma^3 \cdot 2 + \dots$

This is to prove that the optimality definition changes based on γ.

Limitations of solving MDP to find optimal policy

Defined formally which is the optimality of a policy (and the fact that γ modify the optimality), we want now move to how can we compute the optimal policy.



Optimal Value Function

- We said that that $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s), \quad \forall s \in \mathcal{S}$
- As a consequence we can compute optimal state-value function and optimal action-value function as:

unique

$$V^*(s) \doteq \max_\pi V_\pi(s) \quad \forall s \in \mathcal{S}$$

$$Q^*(s,a) \doteq \max_\pi Q_\pi(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

They're simply the value function and the action value function of the optimal policy π^*

Note: we can have n optimal policies (different among them), since the previous property (of optimality) says "at least one", however they'll all have the same value function (and action value function)

Bellman Optimality Equations

- Bellman Optimality Equation for V*

$$V^*(s) = \sum_{a \in \mathcal{A}} \pi^*(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right)$$

$$= \max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

we cannot use this equation because $\pi^*(a|s)$ is unknown (previously we didn't have this problem because we needed to evaluate a given policy π)

- Bellman Optimality Equation for Q*

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi^*(a'|s') Q^*(s',a')$$

$$= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a'} Q^*(s',a')$$

- Why do we care?

They are recursive and they don't depend on unknown variables!

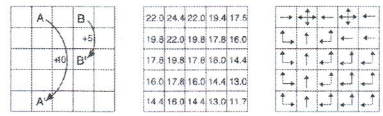
! Thanks to the fact that π^* needs to be optimal, for any state we can procede greedly: we give up on a distribution among actions and we always choose the action that provides the highest return $(r(s,a) + \sum p(s'|a,s) V^*(s'))$. We can procede like this because we know that there is at least 1 DETERMINISTIC policy that is OPTIMAL.

Why do we care?

- From V^* and Q^* we can easily compute the optimal policy π^*

$$\pi^*(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\} = \arg\max_a Q^*(s,a)$$

- Example:



a) gridworld $\gamma = 1$ b) v_* c) π_*

Computing the Optimal Value Function

$$V^*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

r, p, γ → Linear System Solver → V*

Computing the Optimal Value Function

$$V^*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

r, p, γ → Linear System Solver → V*

because of the max the problem becomes one which is **not** a linear system

Computing the Optimal Value Function

$$V^*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

r, p, γ → Linear System Solver → V* ✗

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right)$$

π*, r, p, γ → Linear System Solver → V* ✓

Computing the Optimal Value Function

$$V^*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

r, p, γ → Linear System Solver → V*

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right)$$

? π r, p, γ → Linear System Solver → V*

we need a different approach

Computing the Optimal Value Function

$$V^*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

r, p, γ ⟶ [Linear System Solver] ⟶ V^*

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right)$$

? π r, p, γ ⟶ [Linear System Solver] ⟶ V^*

⟹ Dynamic Programming and Reinforcement Learning Algorithms