Reinforcement Learning
Lecture 2

Georg Martius

Distributed Intelligence / Autonomous Learning Group, Uni Tübingen, Germany

October 22, 2024

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Organization

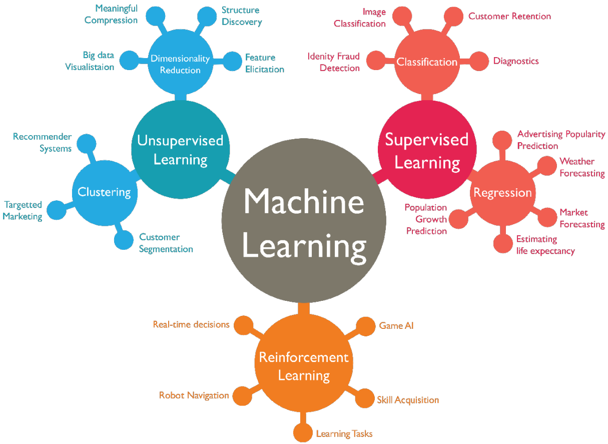
Tutor sessions:

- Hörsaal N11 (next door) (change in room in 3 weeks)
- Hörsaal N03 (Hörsaalzentrum Morgenstelle) (change in room in 3 weeks)
- Hörsaal TTR2 (Obere Viehweide, Maria-von-Linden-Str. 6, Tübingen AI center)

Lectures:

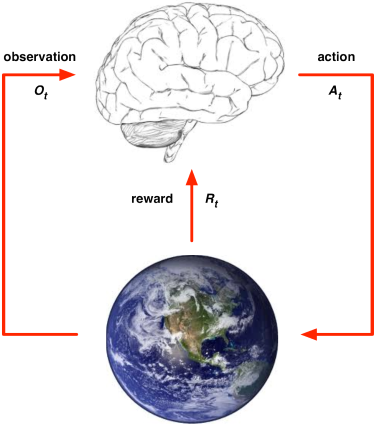
- Today's lecture will go over-time (would like to cover until Value-Iteration for homeworks)
- Next week: I am on a scientific Conference (EWRL). Dr. René Geist will substitute me

Three different classes of tasks:



RL is special, because: interaction with a system, generation of own data, sequential

Agent and Environment (Reminder)



The Reinforcement Learning Problem (Reminder)

Behave such that maximal expected (discounted) future return is achieved

- Behave: find Policy that determines actions
- Optimal w.r.t. expected return: Value function
- Maybe Model the environment

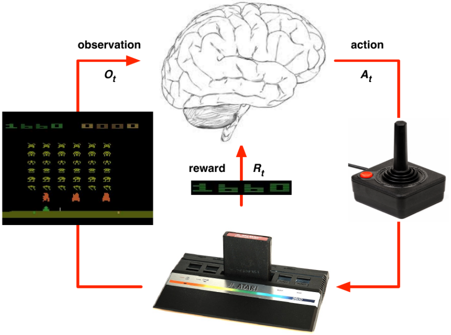
Learning and Planning

Two fundamental problems in sequential decision making

- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
  - a.k.a. learning by doing, trail and error learning
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy
  - a.k.a. deliberation, reasoning, introspection, pondering, thought, search

[Slide adapted from David Silver]

Atari Example: Reinforcement Learning

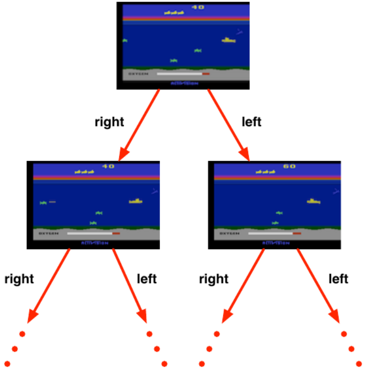


- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

[Slide adapted from David Silver]

Atari Example: Planning

- Rules of the game are known
- Can query emulator
  perfect model inside agent's brain
- If I take action a from state s:
  - what would the next state be?
  - what would the score be?
- Plan ahead to find optimal policy
  e.g. tree search



[Slide adapted from David Silver]

Relationship between Planning and Reinforcement Learning

- Planning: on-the-fly computation of best action
  - typically short-horizon optimization only
- RL: learns (for a long time) to find the best policy
  - solves the global optimzation problem
  - amortizes previous interactions into a policy ➡ fast at runtime
- Is planning and RL mutually exclusive?
  - No, but traditionally treated by different communities
  - can be combined (model-based RL, AlphaGo, etc)

Prediction and Control

Subproblems within Reinforcement Learning:

- Prediction: evaluate the future

  Given a policy: How good is the agent?
- Control: optimize the future

  Find the best policy with respect to current knowledge

Markov Decision Processes

Formally describe environments for reinforcement learning

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, \ldots$ with the Markov property.

Reminder: Markov property

A state S_t is Markov if and only if

$$P(S_{t+1} \mid S_t) = P(S_{t+1} \mid S_1, \ldots, S_t)$$

Definition (Markov Process/ Markov Chain)

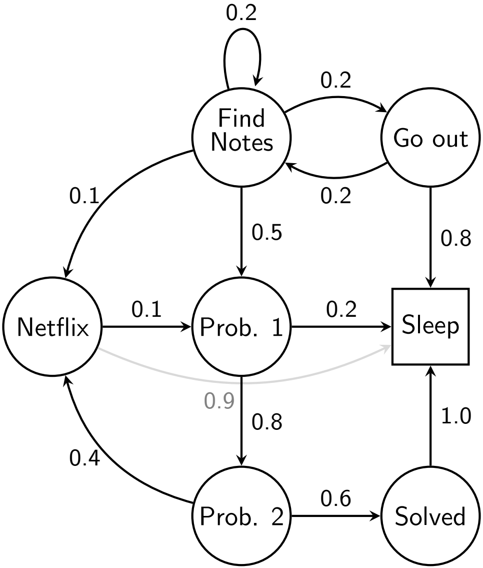
A *Markov Process* (or *Markov Chain*) is a tuple $(\mathcal{S}, \mathcal{P})$

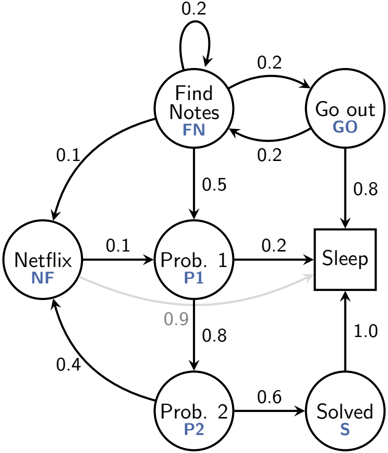
- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'} = P(S_{t+1} = s' \mid S_t = s)$$

Example: Student Markov Chain

Let's consider the evening of a student in this class ;-)





Sample episodes for starting from s_1 =FN

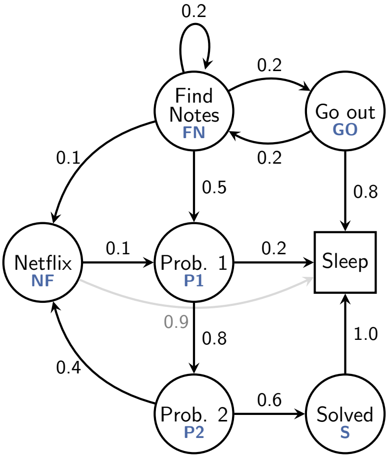
$$s_1, s_2, \ldots, s_T$$

- FN P1 P2 S Sleep
- FN GO FN FN P1 Sleep
- FN P1 P2 NF P1 P2 S Sleep
- FN GO FN P1 P2 NF P1 P2 NF P1 Sleep

Example: Student Markov Chain Episodes

Transition Matrix:

$$\mathcal{P}_{ss'} = P(S_{t+1} = s' \mid S_t = s)$$



Transition Matrix:

$$\mathcal{P} = \begin{array}{c} \\ \text{FN} \\ \text{P1} \\ \text{P2} \\ \text{S} \\ \text{GO} \\ \text{NF} \\ \text{Sleep} \end{array} \begin{array}{c} \begin{array}{ccccccc} \text{FN} & \text{P1} & \text{P2} & \text{S} & \text{GO} & \text{NF} & \text{Sleep} \end{array} \\ \begin{bmatrix} 0.2 & 0.5 & & & 0.2 & 0.1 & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & & 0.4 & \\ & & & & & & 1.0 \\ 0.2 & & & & & & 0.8 \\ & 0.1 & & & & & 0.9 \\ & & & & & & 1 \end{bmatrix} \end{array}$$

Markov Reward Process

A Markov reward process is a Markov chain with values.

Definition (MRP)

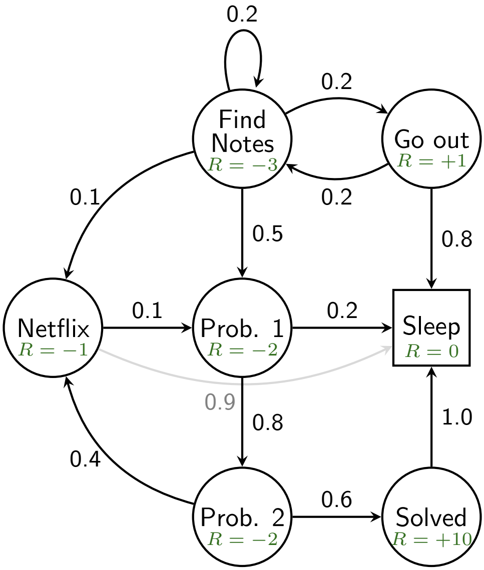
A *Markov Reward Process* is a tuple $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix,

$$P(S_{t+1} \mid S_t) = P(S_{t+1} \mid S_1, \ldots, S_t)$$

- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Example: Student MRP



Return

Definition

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The discount $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k+1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Discussion on discounting

Return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why is discouting often used?

- Mathematically convenient to discount rewards (keeps returns finite)
- A way to model the uncertainty about the future (since the model may not be exact)
- Animal/human behaviour shows preference for immediate rewards

In some cases *undiscounted* Markov reward processes (i.e. $\gamma = 1$), are considered, e.g. if all sequences terminate.

Value Function

The value function describes the value of a state (in the stationary state)

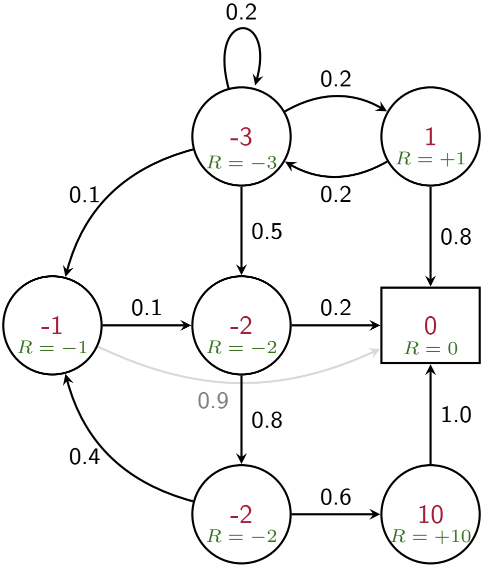
Definition

The state *value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

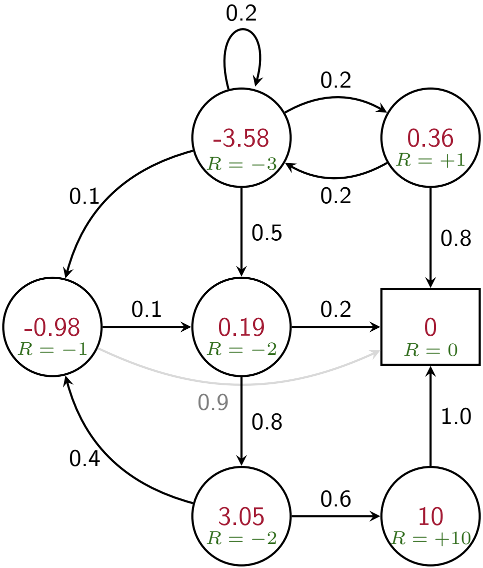
Example: Value Function for Student MRP

Extremely myopic $\gamma = 0$: $v(s)$



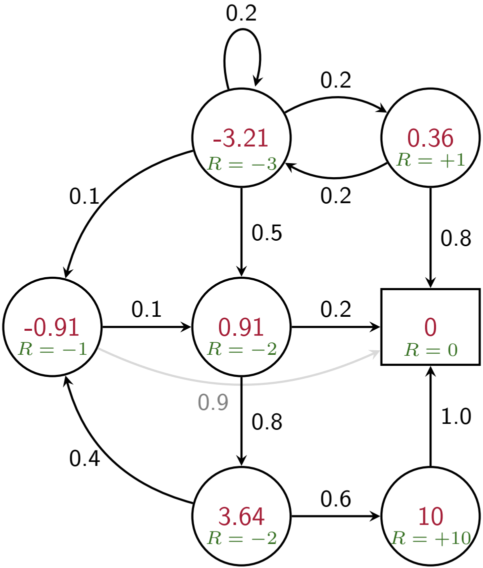
Example: Value Function for Student MRP

Now more farsighted $\gamma = 0.9$: $v(s)$



Example: Value Function for Student MRP

Now fully farsighted $\gamma = 1.0$: $v(s)$



Bellman Equation (MRP) I

Idea: Make value computation recursive by tearing apart contributions from:

- immediate reward
- and from discounted future rewards

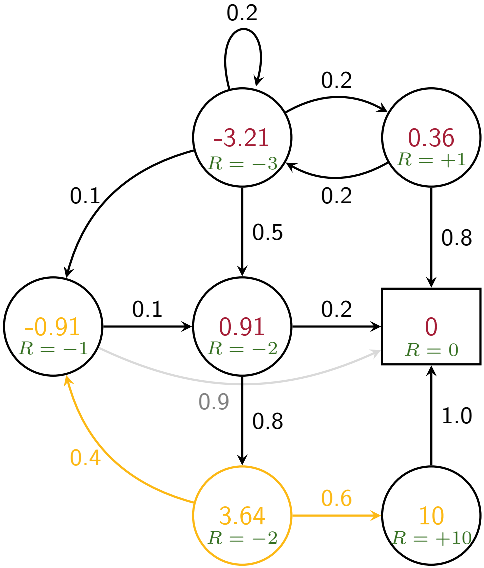
$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

Mh... need Expectation over S_{t+1}
Use transition matrix to get probabilities of succeeding state:

$$
v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')
$$

Example: Bellman Equation for Student MRP

$\gamma = 1.0$: $v(P2)$



$$v(P2) = 3.64 = -2 + 0.4 \cdot (-0.91) + 0.6 \cdot 10$$

Bellman Equation (MRP) II

Bellman equations in matrix form:

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where $v \in \mathbb{R}^{|S|}$ and $\mathcal{R}$ are vectors

The Bellman equation can be solved explicitly (in closed form):

$$v = (\mathbb{I} - \gamma \mathcal{P})^{-1} \mathcal{R}$$

- computational complexity is $O(|S|^3)$

A Markov reward process has no agent, there is no influence on the system.
An MRP with an active agent forms a Markov Decision Process.

- Agent takes decision by executing actions
- State is Markovian

Definition (MDP)

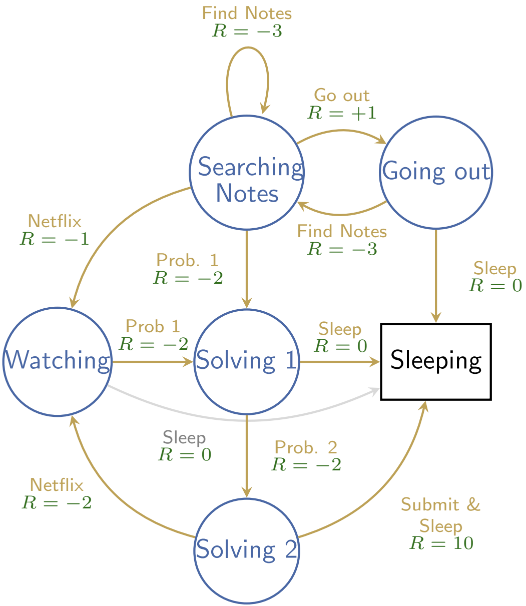
A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'}^{a} = P(S_{t+1} \mid S_t, A_t = a)$$

- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

Example: Student MDP



How to model decision taking?

The agent has an action function called policy.

Definition

A policy π is a distribution over actions given states,

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

- Since it is a Markov process the policy only depends on the current state
- Implication: policies are stationary (independent of time)

An MDP with a given policy turns into a MRP:

$$\mathcal{P}^{\pi}_{ss'} = \sum_{a \in \mathcal{A}} \pi(a \mid s) \mathcal{P}^{a}_{ss'}$$

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a \mid s) \mathcal{R}^{a}_{s}$$

Modelling expected returns in MDP

How good is each state when we follow the policy π?

Definition

The state-value function $v_\pi(s)$ of an MDP is the expected return when starting from state s and following policy π.

$$v_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$$

Should we change the policy?
How much does choosing a different action change the value?

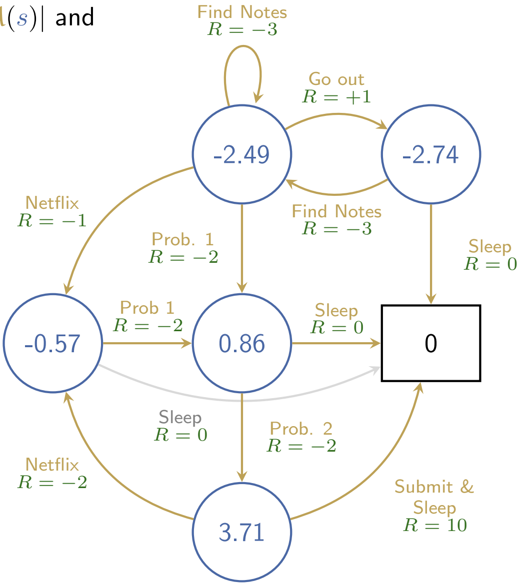
Definition

The action-value function $q_\pi(s, a)$ of an MDP is the expected return when starting from state s, taking action a, and then following policy π.

$$q_\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$$

Example: State-Value function for Student MDP

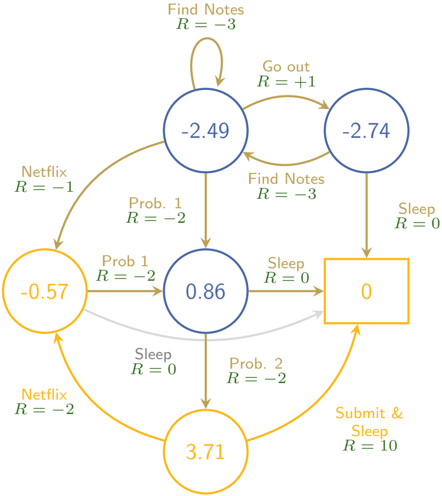
$v_\pi(s)$ for uniform policy
$\pi(a \mid s) = 1/|\mathcal{A}(s)|$ and
$\gamma = 1$



Recall:
Bellman Equation: decompose expected reward into immediate reward plus discounted value of successor state:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

$$v_\pi(P2) = 3.71$$
$$= 0.5 \cdot (-2 - 0.57) + 0.5 \cdot (10 + 0)$$



$\pi(a \mid s) = 1/|\mathcal{A}(s)|$ and $\gamma = 1$

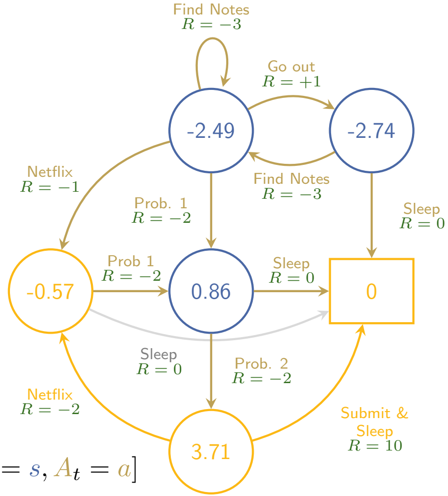
Bellman Expectation Equation

Recall:
Bellman Equation: decompose expected reward into immediate reward plus discounted value of successor state:

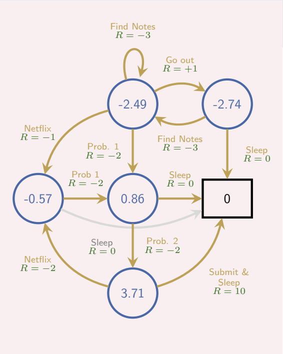
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

The action-value function can be similarly decomposed,

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$



$\pi(a \mid s) = 1/|\mathcal{A}(s)|$ and $\gamma = 1$



Value function can be derived from q_π:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s, a)$$

...and q can be computed from transition model

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Substituting q in v:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Explicit solution for v_π

Since a policy induces an MRP v_π can be directly computed (as before)

$$v = (\mathbb{I} - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

But do we want v_π?

We want to find the optimal policy and its value function!

Optimal Value Function

Definition

The optimal state-value function $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

Definition

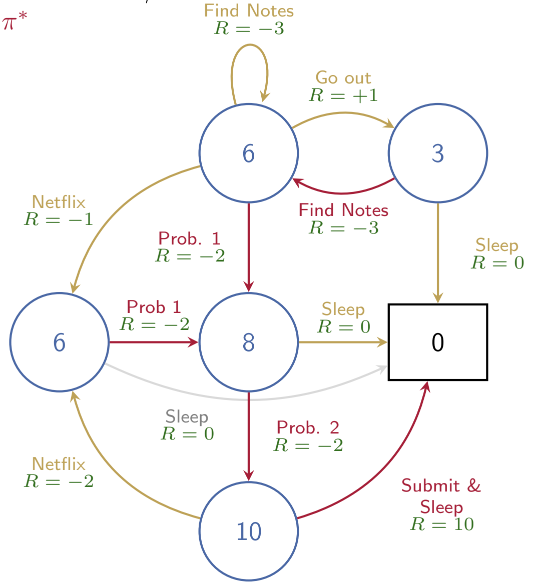
The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

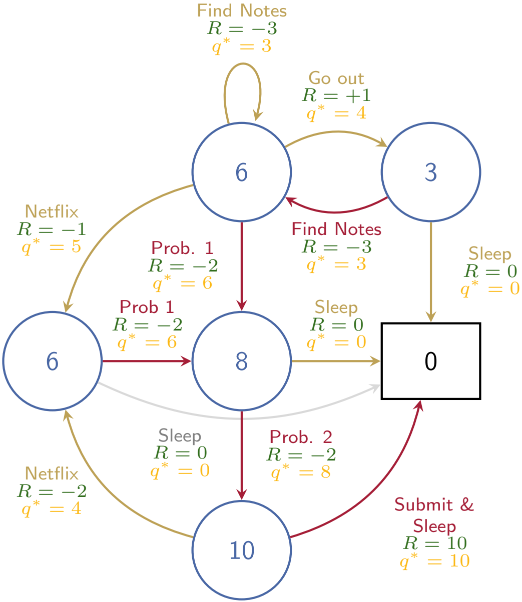
What does it mean?

- v_* specifies the best possible performance in an MDP
- Knowing v_* solves the MDP (how? we will see...)

Optimal value function for $\gamma = 1$
optimal policy π^*



Example: Optimal State Function q_* in Student MDP



To solve an MDP, we need to obtain the optimal policy.
Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal state-value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

Finding an Optimal Policy

Given the optimal action-value function q_*:
How do we get the optimal policy?

Discuss with your peers ...

The optimal policy is given by maximizing q_*

$$\pi_*(a \mid s) = [\![a = \arg\max_{a \in \mathcal{A}} q_*(s, a)]\!]$$

$[\![\cdot]\!]$ is Iverson bracket: 1 if *true*, otherwise 0.

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy (greedy)

Bellman Equation for optimal value functions

Also for the optimal value functions we can use Bellmans optimality equations: Exercise!

Remember:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, q_\pi(s, a)$$

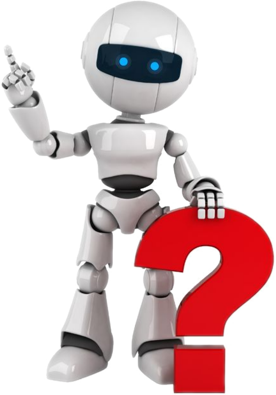
$$\pi_*(a \mid s) = [\![a = \arg\max_{a \in \mathcal{A}} q_*(s, a)]\!]$$

Solving the Bellman Optimality Equation

Bellman Optimality Equation is non-linear (because of max operation)

- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - SARSA

Questions?



[Image: globalrobots.com]

Break

Dynamic Programming

Dynamic Programming?

Dynamic: sequential or temporal component of the problem

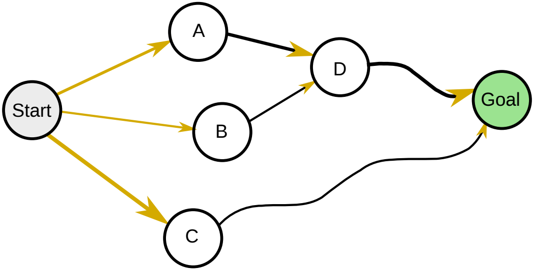
Programming: optimizing a “program”, i.e. a policy
name like in linear programming
(mathematical programming = optimization)

- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

When can we use Dynamic Programming?

When problems have two properties:

- Optimal substructure
  - Principle of optimality applies
  - Optimal solution can be decomposed into subproblems
- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused



When can we use Dynamic Programming?

When problems have two properties:

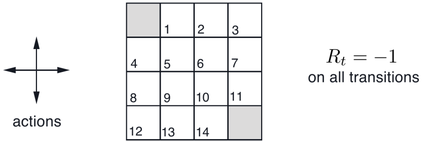
- Optimal substructure
  - Principle of optimality applies
  - Optimal solution can be decomposed into subproblems
- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused

Markov decision processes? ➡ Satisfy both properties!

- Bellman equation gives recursive decomposition
- Value function stores and reuses solutions

Dynamic programming assumes full knowledge of the MDP!

Example MDP:



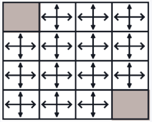
- Undiscounted episodic MDP ($\gamma = 1$)
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached

Can be used for planning in an MDP.

Input: MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ **and policy** π

For prediction

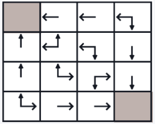
Yields value function v_π



0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

For control

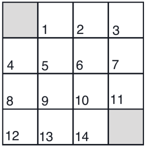
Yields optimal value function v_* and π_*



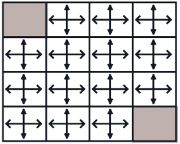
Prediction: Policy Evaluation

Problem: evaluate policy (find its value function)

actions



$R_t = -1$
on all transitions



0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Solution: Dynamic Programming to compute value function

Iterative algorithm to obtain: $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_\pi$

Iterative Policy Evaluation

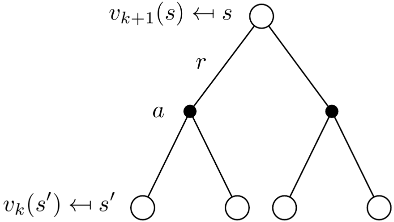
- At each iteration $k+1$
- For all states $s \in \mathcal{S}$
- Update $v_{k+1}(s)$ from $v_k(s')$ for all s'
- where s' is a successor state of s

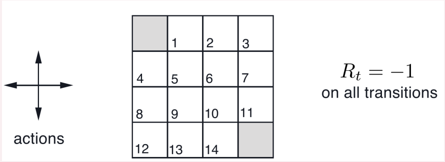
Bellmann Expectation Equation:

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\boldsymbol{v}_{k+1}(s) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \boldsymbol{v}^k$$

"Backup" diagram:





Iterative Policy Evaluation: Small Gridworld

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Improve the Policy!

Policy Iteration

Given a policy π

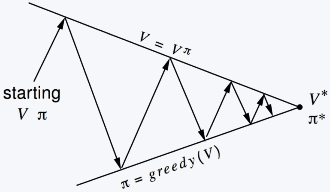
1. Evaluate the policy π (Policy Evaluation)

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots \mid S_t = s]$$

2. Improve the policy by acting greedily with respect to v_π (Policy Improvement)

$$\pi' = \text{greedy}(v_\pi)$$

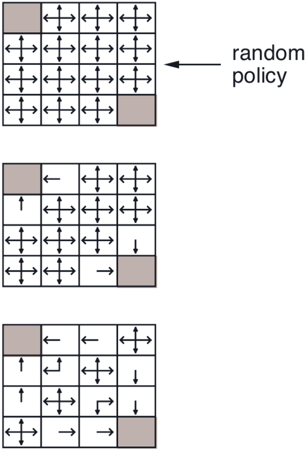
3. iterate until policy does not change



- Policy iteration always converges to π^*

v_k for the Random Policy

Greedy Policy w.r.t. v_k



$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

Policy Iteration in Small Gridworld (one iteration)

$k = 3$

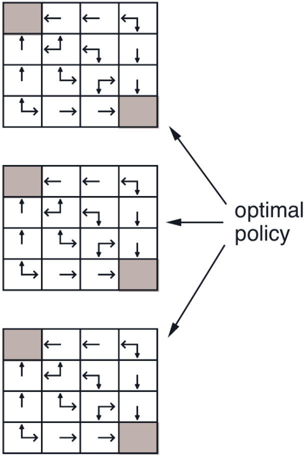
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



- In Small Gridworld: Policy improvement converges after one iteration ($\pi' = \pi^*$)

Policy Iteration: Convergence Proof

For self study

- Consider a deterministic policy, $a = \pi(s)$
- Policy *improvement* step by acting greedily

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s, a)$$

- This improves the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function: $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s) \end{aligned}$$

Policy Iteration: Convergence Proof (cont.)

For self study

- If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

- Then the Bellman optimality equation has been satisfied

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

- Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$

➡ π is an optimal policy

$\square$

Value Iteration

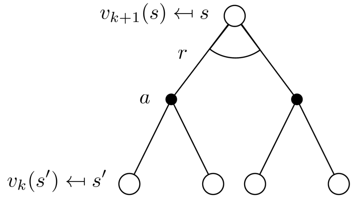
Goal: compute optimal value function and optimal policy

Algorithm: Like Policy Iteration, but with 1-step policy evaluation

1. Policy Evaluation: estimate v_π (not to congergence)
2. Policy Improvement: generate π' with $v_{\pi'} \leq v_\pi$
3. iterate until value function does not change

- Intuition: start with final rewards and work backwards
- given an optimal solution for some states, e.g. $v_*(s)$
  ➩ solution $v_*(s)$ can be found by one-step lookahead

"Backup" diagram:



Value Iteration

Improve subproblem solution $v_k\,(s')$

based on current best estimates of $v_*(s)$

Value Iteration

1. For all states $s \in \mathcal{S}$
2. Update $v_{k+1}(s)$ from $v_k\,(s')$

$$v_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k\,(s') \right)$$

Bellmann Optimality Eq.

3. if value function changes goto (1): next iteration with $k \leftarrow k+1$

- There is not explicit policy here
- Intermediate value functions might not correspond to any policy
- Convergences to v_*
- Is the same as Policy Iteration, but with 1-step Policy evaluation

Value Iteration: Example Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

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

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Sychronous Dynamic Programming Algorithms

Bellmann Expectation and Optimality Eq:

$$v_{k+1}(s) = \mathop{\mathbb{E}}_{a\sim\pi}\left(\mathcal{R}_s^a + \gamma \sum_{s'\in S} \mathcal{P}_{ss'}^a v_k(s')\right)$$

$$v_{k+1}(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s'\in S} \mathcal{P}_{ss'}^a v_k(s')\right)$$

So far we looked at synchronous Methods:
all states are updated at once.

Problem	**Bellman Equation**	**Algorithm**
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Complexity: (m actions, n states)

- Using state-value function $v(s)$: $O(mn^2)$ per iteration
- Using state-action-value function $q(s,a)$: $O(m^2n^2)$ per iteration

Next time

- Model-free Prediction and Control