Reinforcement learning (I)

AI 2025/2026

Content

Reinforcement learning

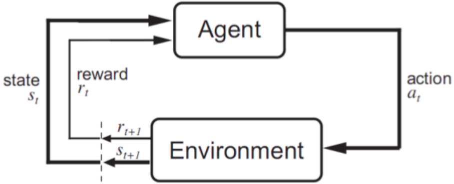
The agent must learn a behavior, without having an instructor

- The agent has a task to perform
- Performs a series of actions
- Receives *feedback* from the environment: how well he acted to accomplish the task.
  The agent receives a positive reward if the task is performed well, otherwise a negative reward.

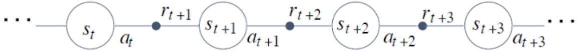
This learning method is called reinforcement learning.

The interaction model

- The agent performs actions
- The environment presents the agent situations called states and gives rewards



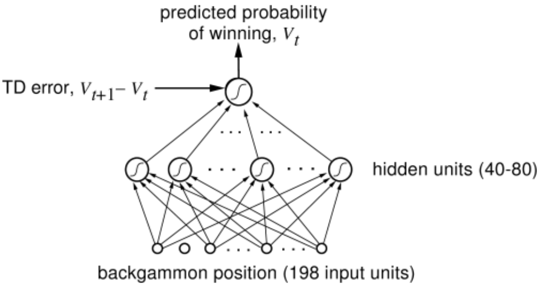
Trajectory / episode $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots)$



Reinforcement learning

- The goal: to make the agent to act in order to maximize his rewards
- The agent must identify the **sequence of actions** leading to the completion of the task
  - Data acquisition and supervised learning on data
    Training data: (S, A, R) State, Action, Reward

Applications: TD-Gammon



A neural network trained with *TD-learning*

- input: configuration
- output: an estimation of the value for that configuration

Learn from simulations (play games against himself)

Examples: robotics

AIBO: learn to walk '04 (using *Policy gradient*)

Fig. 5. The training environment for our experiments. Each Aibo times itself as it moves back and forth between a pair of landmarks (A and A', B and B', or C and C').

https://www.cs.utexas.edu/users/AustinVilla/?p=research/learned_walk

Minitaur quadrupedal robot: learn to walk (using *actor-critic deep RL*)

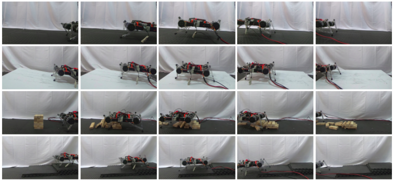


Fig. 9: We trained the Minitaur robot to walk on flat terrain (first row) in about two hours. At test time, we introduced obstacles, including a slope, wooden blocks, and steps, which were not present at training time, and the learned policy was able to generalize to the unseen situations without difficulty (other rows).

https://sites.google.com/view/minitaur-locomotion/

Deep Reinforcement Learning

AlphaGo (Google DeepMind, '15): the program learned to play Atari 2600 games by directly watching only the display and the score.

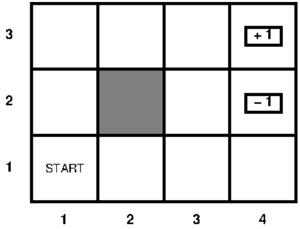
- 2016: won against a 9 dan professional GO player (prize of 1 000 000$)
- 2017: won against Ke Jie, the best GO player in the world

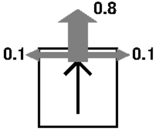
AlphaGo Zero learned to play **without input from human games** (just based on the rules of the game) and beat AlphaGo with 100-0 (Oct. '17)

AlphaZero beat AlphaGo Zero with 60-40, and after just 8 hours of training outperformed all existing **GO** and **CHESS** programs (Dec. '17)

Content

Sequential decisions

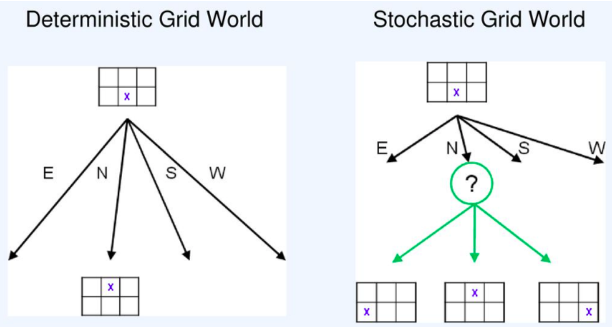




- Deterministic environment
  - (up, up, right, right, right)
- Stochastic environment
  - Transitions model $P(s'|s,a)$: probability of reaching state s' if action a is done in state s
  - The action obtains the intended effect with probability 0.8
  - The agent receives a reward: -0.04 for nonterminal states; $+$/-1 for terminal states

Deterministic vs. stochastic

Stochastic: for the sequence (up, up, right, right, right), the probability of obtaining the final state is $0.8^5 = 0.33$



Markov assumption

- The current state s_t depends on a finite history of previous states
- First order Markov process: the current state s_t depends only on the previous state s_{t-1}

$$P(s_t | s_{t-1}, \ldots, s_0) = P(s_t | s_{t-1})$$

Markov Decision Process (MDP)

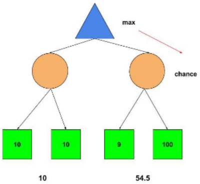
Markov Decision Process: a sequential decision problem for a stochastic environment with a Markov transition model and additive rewards

- States $s \in S$ (initial state s_0), actions $a \in A$
- Transitions model $P(s'|s, a)$
- Reward function $R(s)$

What does a solution look like? Must specify what the agent should do in each state (policy π).

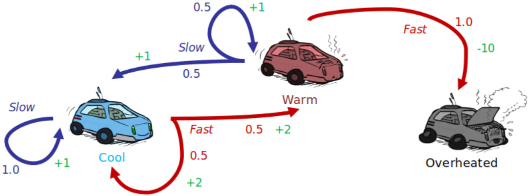
MDPs are non-deterministic search problems.

- Search algorithms. Expectimax alg: when the opponent does not play optimally.

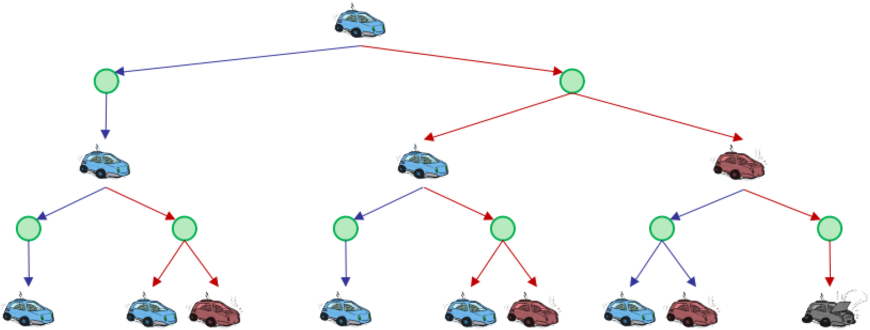


Example

- A robot car wants to travel far and quickly.
- States: Cool, Warm, Overheated; Actions: Slow, Fast; Going faster gets double reward.

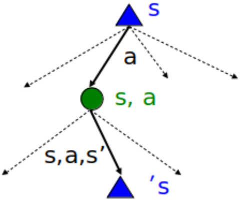


Example: search tree



MDP search trees

- Each state projects a search tree



s state, (s, a, s') transition, $P(s'|s, a)$, $R(s, a, s')$

- In search problems, the goal is to identify an optimal *sequence*.

  In MDP, the goal is to identify an optimal *policy* π^* (strategy)
  - $\pi : S \to A$
    $\pi(s)$ is the action recommended in state s

MDP

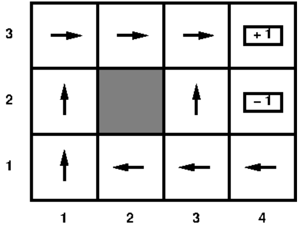
- Utility: the sum of rewards for a sequence of states.
  The reward is the immediate, short-term gain; utility is the total, long-term gain.

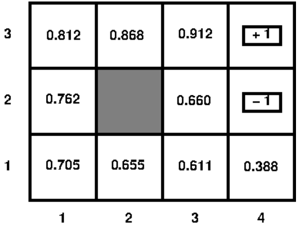
- Quality of a policy: expected utility of possible sequences of states.
  Stochastic environment: we can have a different sequence of states when executing the same policy from the initial state.

  Optimal policy π^* maximizes the expected utility.

MDP

Example: the optimal policy and the state values



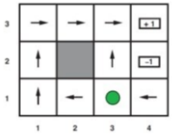


Utilities

Finite horizon

- $U_h([s_0, s_1, \ldots, s_{N+k}]) = U_h([s_0, s_1, \ldots, s_N]), \forall k > 0$
  After a fixed time N, nothing matters.
- The optimal policy is not stationary: the optimal action for a given state may change over time

Example:



- $N = 3 \rightarrow$ have to risk (up)
- $N = 100 \rightarrow$ can choose the safer solution (left)

Utilities

Infinite horizon

- There is no fixed deadline
- The optimal policy is stationary

- a. Additive rewards

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

- b. *Discounted* rewards

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

$\gamma \in [0, 1]$ *discount factor* indicates that future rewards matter less than immediate ones

Discounted rewards

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

- Prefer current rewards, not later ones
- Values of rewards decay exponentially

Example 1: $\gamma = 0.5$
$R(s_0) = 1, R(s_1) = 2, ...$
$U([1, 2, 3]) = 1 * 1 + 0.5 * 2 + 0.25 * 3$
$U([1, 2, 3]) < U([3, 2, 1])$

Example 2: Quiz

Infinite horizon - evaluation

We must ensure that the utility of a possibly infinite sequence is finite.

- **1st Approach**. If the rewards are bounded and $\gamma < 1$ then:

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = R_{max}/(1-\gamma)$$

- **2nd Approach**. If the environment contains terminal states and it's guaranteed that the agent will reach one of them (we have a *proper policy*), we can use $\gamma = 1$

Expected utility

- Each policy generates multiple sequences of states, due to the uncertainty of transitions $P(s'|s, a)$
- Let S_t be a random variable: the state the agent reaches at time t executing policy π; $S_0 = s$.
  Expected utility obtained by executing policy π starting in state s:

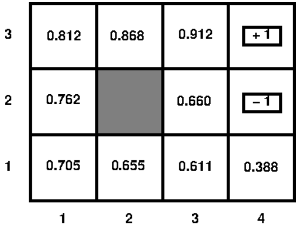
$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

($=$ the expected value of the sum of all discounted rewards obtained for all possible sequences of states)

Evaluation of a policy

- Optimal policy
$$\pi_s^* = argmax_\pi U^\pi(s)$$
- $U^{\pi^*}(s)$ the true utility of a state: the expected value of the sum of the discounted rewards if the agent executes an optimal policy

Example: Consider $\gamma = 1$ and $R(s) = -0.04$.



Near the final state, the utilities are higher because fewer steps with negative reward are required to reach that state.

Maximum Expected Utility

Maximum Expected Utility: choose the action that maximizes the expected utility of the subsequent state

$$\pi^*(s) = argmax_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

Bellman equation

$$U(s) = R(s) + \gamma max_{a \in A(s)} \sum_{s'} P(s'|s, a)U(s')$$

Bellman equation (1957): the utility of a state is the immediate reward for that state, $R(s)$, plus the maximum expected utility of the next state.

Example

The utility of state (1,1):

$$\begin{aligned} U(1,1) = -0.04 + \gamma max[0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), &\quad (Up)\\ 0.9U(1,1) + 0.1U(1,2), &\quad (Left)\\ 0.9U(1,1) + 0.1U(2,1), &\quad (Down)\\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)] &\quad (Right) \end{aligned}$$

The best action: *Up*.

Solving a Markov decision process

- n possible states
- n Bellman equations, one for each state
- n equations with n unknowns: $U(s)$
- Cannot be solved as a system of linear equations because of the *max* function

Content

I. *Value iteration*

Computes the utility of each state and identifies the optimal action in each state.

The algorithm for computing the optimal policy:

- Initializes utilities with arbitrary values
- Updates the utility of each state from the utilities of its neighbors

$$U_{i+1}(s) = R(s) + \gamma max_a \sum_{s'} P(s'|s,a) U_i(s')$$

- Repeat for each s simultaneously, until an equilibrium is reached

Value iteration: pseudocode

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
rewards $R(s)$, discount γ
ϵ, the maximum error allowed in the utility of any state
local variables: U, U', vectors of utilities for states in S, initially zero
δ, the maximum change in the utility of any state in an iteration

repeat
$U \leftarrow U'; \delta \leftarrow 0$
for each state s **in** S **do**
$$U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$$
if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \epsilon(1-\gamma)/\gamma$
return U

Value iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

From Sutton&Barto. Reinforcement Learning: an introduction

Utility function $U \leftrightarrow V$ value function

Value iteration

- Example: Racing
- Demo: https://courses.grainger.illinois.edu/cs440/fa2018/lectures/mdp-value-demo.pdf

Problems with Value iteration

- Slow: $O(S^2 A)$ per iteration
- The "max" at each state rarely changes
- The policy often converges before the values

Content

II. Policy iteration

- If we **fix the policy**, we have only one action per state
- The algorithm alternates the following steps:
  - 1. Policy evaluation: given a policy π_i, compute the utilities of the states based on the policy π_i: $U_i = U^{\pi_i}$
  - 2. Policy improvement: compute a new policy π_{i+1}, based on the utilities U_i

  Repeat the steps until the policy converges

1. Policy evaluation

The action for each state is fixed by the policy; at iteration i, the policy π_i specifies the action $\pi_i(s)$ in state s.

The simplified Bellman equations

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

- A system of n linear equations with n unknowns
- It can be solved *exactly* in $O(n^3)$ or *approximately*
- Apply *Value iteration*

$$U_{i+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

Example: $\pi_i(1,1) = Up$, $\pi_i(1,2) = Up$
Simplified Bellman equations:
$U_i(1,1) = -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)$
$U_i(1,2) = -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2)$

2. Policy improvement

- The values $U(s)$ are known
- Computes for each s, the optimal action

$$a_i^*(s) = max_a \sum_{s'} P(s'|s, a)U(s')$$

- If $a_i^*(s) \neq \pi_i(s)$, update the policy: $\pi_{i+1}(s) \leftarrow a_i^*(s)$
  Only the "promising" parts of the search space can be updated.

Policy iteration: pseudocode

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Demo: `https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html`

Bibliography

- Russel&Norvig. Artificial Intelligence: A modern Approach. Ch. 17. Making Complex Decisions
- Sutton&Barto. Reinforcement Learning. An introduction Ch 3. Finite Markov Decision Processes, Ch 4. Dynamic Programming http://incompleteideas.net/book/RLbook2020.pdf