UNIVERSITÄT OSNABRÜCK

University of Osnabrück, Institute of Cognitive Science

Machine Learning

10 – Reinforcement Learning

SS 2018

Gunther Heidemann

Introduction

So far:

- Supervised learning:
  - Task: Learn a target function. Special case: Classification.
  - “Teacher“ provides (input, output) pairs.
  - System architecture and parameters cause bias.
- Unsupervised learning:
  - Task: Find structure within data (e.g., clusters).
  - Only input is provided, no teacher (at least in the sense of labeling the input with a target output).
  - System architecture and parameters code goal implicitly.

Common to both:

- Learning system is not part of some “environment”.
- No active knowledge acquisition.

Reinforcement learning (RL):

- Agent within environment that
  - has perception (input),
  - can perform actions (output).
- Aim: Find optimal actions depending on percepts to achieve some (maybe far off) goal.

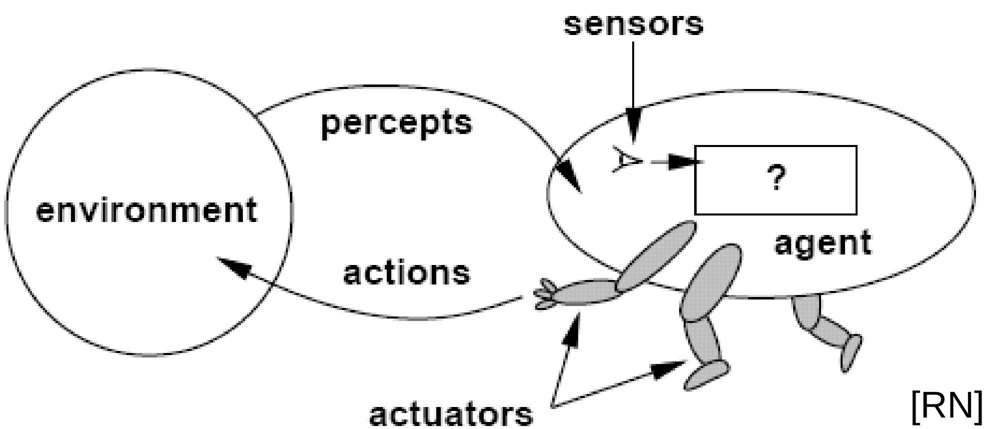
Examples:

- Mobile robot is searching for a way.
- Robot end-effector manipulates objects.
- Optimize operations in a factory.
- Play board game.

Agents

An agent is defined by “PEAS” (see courses on AI):

1. **P**erformance
2. **E**nvironment
3. **A**ctuators
4. **S**ensors



[RN]

Agents

Robot manipulates objects on a table:

1. Performance: Move object from one location to another.
2. Environment: Table with objects.
3. Actuators: Motor current.
4. Sensors: Camera, tactile sensors, force sensors.

Autonomous vehicle:

1. Performance: Trade off between reaching the goal fast and risk of causing damage.
2. Environment: Streets, cars, pedestrians, traffic signs, GPS signal.
3. Actuators: Motor, gear, steering.
4. Sensors: Laser range scanner, camera, GPS.

Agents

Gambling machine:

1. Performance: Extract money, sub-goals: Entertain, cause addiction.
2. Environment: User.
3. Actuators: Graphics, sound, money.
4. Sensors: Input given by buttons, levers etc.

Software agent, e.g., expert system:

1. Performance: Find appropriate answer after few questions.
2. Environment: User, database.
3. Actuators: Return data from database.
4. Sensors: Text input (request).

Knowledge acquisition in RL

"Weak teacher":

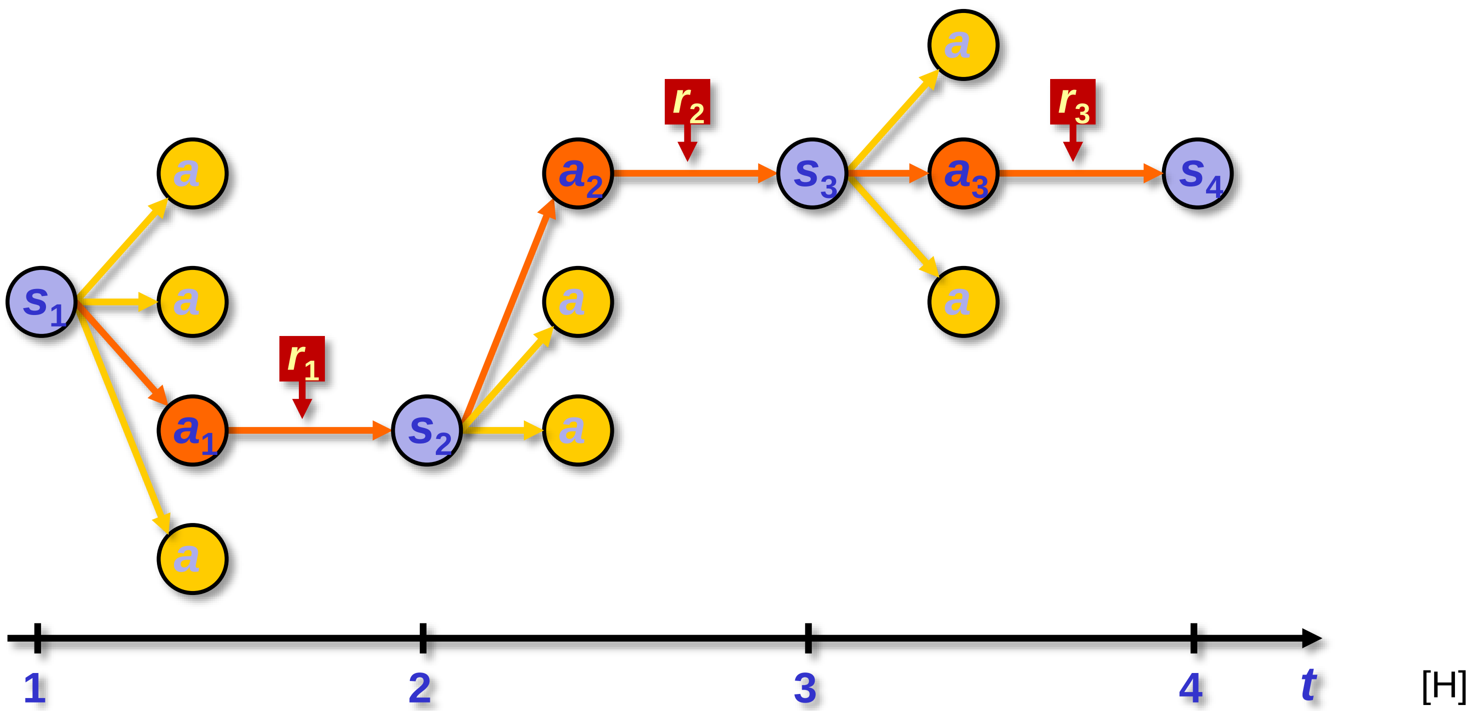
- Does not tell which action to choose in a certain situation explicitly.
- Instead, a reward is given when a certain action is performed in a certain state (negative reward = punishment).
- Reward is delayed since the action is performed first, leading to a different state.
- Aim: Maximize cumulative reward obtained over a sequence of actions.
- Rewards may be zero, i.e., no information is conveyed.
- Limit case: No rewards on the way, only at the final state. The agent has to infer a sequence of actions from the final reward only.
- Training: Try again and again to reach the goal by improving the cumulative reward (a "playground" is provided).
- Active exploration of states and actions may be useful.

RL for a Markov decision process

Restrict reinforcement learning to a Markov decision process:

- Agent performs transitions between discrete states from a finite set S. So states can not have a continuous parameterization, e.g., real valued position.
- This implies discrete time t.
- The agent can choose one action at a time from a set of discrete actions A.

RL for a Markov decision process



At time t, the agent is in state $s_t \in S$ and chooses action $a_t \in A$, which leads to receiving a reward $r_t \in \mathbb{R}$, and brings the agent into state s_{t+1}.

Training examples are of the form $((s,a), r)$, not (s,a), i.e., information is conveyed by rewarding an action a performed in state s, not by telling explicitly what action a to take in state s.

RL for a Markov decision process

Markov assumption:

- Successor state depends only on the current state and current action, not on earlier ones:

$$s_{t+1} = \delta(s_t, a_t)$$

$$\text{(not } s_{t+1} = \delta(s_t, s_{t-1}, s_{t-2}, \dots, a_t\, a_{t-1}, a_{t-2}, \dots)),$$

  where δ is the successor function.

- The same applies to the reward:

$$r_t = r(s_t, a_t)$$

$$\text{(not } r_t = r(s_t, s_{t-1}, s_{t-2}, \dots, a_t\, a_{t-1}, a_{t-2}, \dots, r_{t-1}, r_{t-2}, \dots)).$$

  where r is the reward function.

The successor function $\delta(s,a)$ and the reward function $r(s,a)$ may be

- unknown to the agent,
- non-deterministic.

Learning task

Execute actions and observe the results to learn an *action policy*

$$\pi : S \rightarrow A$$

which yields for each state the action to be executed.

To judge the success of the policy π, a function V that values the obtained rewards is needed.

Naïve approach:

$$V^{\pi}(s_t) = r_t + r_{t+1} + r_{t+2} + \ldots = \sum_{t'=t\ldots\infty} r_{t'}.$$

Problem:

If the agent does not reach a final state, the value may be infinite. Comparing different policies is not facilitated by this.

Horizon

Idea: Finite "*horizon*" after N time steps:

$$V^{\pi}(s_t) \quad = \quad \sum_{t'=t\ldots N+t} r_{t'}.$$

Problems of "hard horizon":

- Choice of N is difficult.
- Best action depends on horizon N.
- Thus, the best action depends on time (steps).
- The optimal policy is not stationary.

Better solution: "Soft horizon"

$$V^{\pi}(s_t) \quad = \quad r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{t'=t\ldots\infty} \gamma^{t'} r_{t'}.$$

with the *discount factor* $0 < \gamma < 1$ for future rewards, i.e., rewards are less important the more they are in the future (a bird in the hand is worth two in the bush).

Learning

Aim: Find the optimal policy π^* that maximizes the evaluation function

$$\forall s: \quad \pi^* = \text{argmax}_\pi V^\pi(s).$$

The corresponding maximum value V^{π^*} of the evaluation function is denoted by V^*.

Idea:

Learn $\pi^*: S \rightarrow A$.

But:

There are no training examples (s,a), only $((s,a), r)$.

So instead of learning best actions for given states, an evaluation function must be learned.

Learning

Idea:

Learn evaluation function V^*, because it tells that, e.g., a future state s yields higher cumulative rewards than a future state s': $V^*(s) > V^*(s')$.

Since the agent can not choose among states but only among actions, it must infer the optimal action a^* from V^* by a look ahead search over all actions:

$$a^*(s) = \text{argmax}_a \left[r(s,a) + \gamma V^*\big(\delta(s,a)\big)\right].$$

But:

Actions can not be chosen this way since the agent does not know

- the successor function $\delta: S \times A \rightarrow S$,
- the reward function $r: S \times A \rightarrow \mathbb{R}$.

In principle, δ and r can be learned in advance (without searching for optimal actions yet) by a complete search over the state space.

Q function

Define a new evaluation function:

$$Q(s,a) \quad = \quad r(s,a) \; + \; \gamma V^*\big(\delta(s,a)\big).$$

By learning Q instead of V^*, the agent can choose optimal actions without knowing δ:

$$a^*(s) \quad = \quad \mathrm{argmax}_a \left[r(s,a) + \gamma V^*\big(\delta(s,a)\big)\right]$$

$$= \quad \mathrm{argmax}_a \, Q(s,a).$$

Objection: Q is based on δ and r, so where is the improvement?

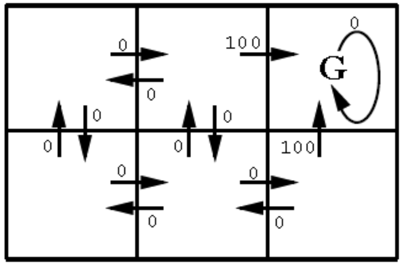
Answer:

As the agent does not know δ and r in advance, it must learn both from exploration. But it is not necessary to learn both explicitly (in separation) and completely, it's easier to learn the quantity Q instead.

Example: *r*, *Q*, *V**, π*

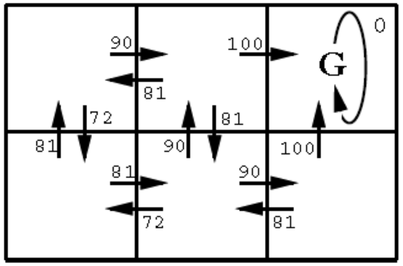
Example:

Board with 3x2 fields, each is a state. Entering goal state *G* yields reward 100, every other transition yields 0.

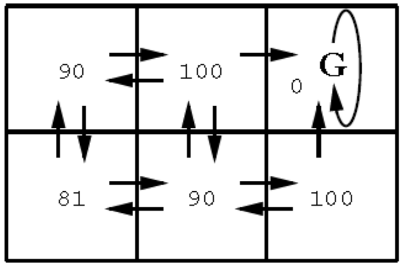


$r(s, a)$ (immediate reward) values

Q and *V** are depicted for discount factor $\gamma = 0{,}9$.

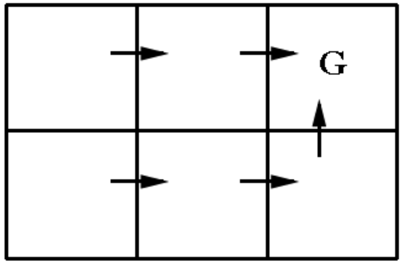


$Q(s, a)$ values



$V^*(s)$ values

There is more than one optimal policy.



One optimal policy

[M]

Training rule for Q

Estimate training values for

$$Q(s,a) \;=\; r(s,a) + \gamma V^*(\delta(s,a))$$

by iterative approximation (Watkins 1989).

Since

$$V^*(s) \;=\; \max_a Q(s,a),$$

Q can be expressed without V^* recursively:

$$Q(s,a) \;=\; r(s,a) + \gamma \max_{a'} Q(s,a').$$

The learners estimate of the true Q will be denoted by q.

The initial estimates $q(s,a)$ may be zero or random values.

The update rule for the estimates is

$$q(s,a) \;\leftarrow\; r + \gamma \max_{a'} q(s',a'),$$

where s' is the state resulting from action a in state s.

It can be proven that q converges to Q.

Q-Learning

Algorithm:

$\forall s\ \forall a$ initialize $q(s,a) \leftarrow 0$.

Observe current state s.

Repeat:

Select action a and execute it.

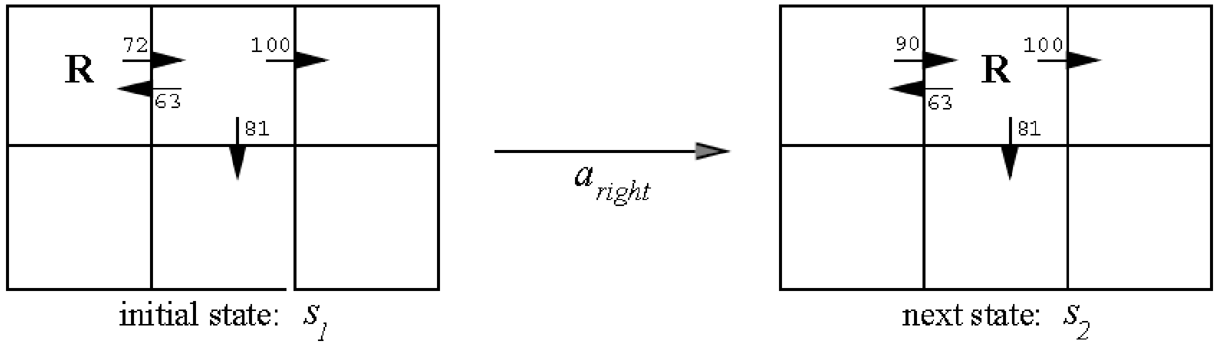
Receive reward r.

Observe new state s'.

Update $q(s,a) \leftarrow r + \gamma \max_{a'} q(s',a')$.

$s \leftarrow s'$.

Q-Learning example



[M]

Transition:

Robot R is in state s_1 (upper left). Action a_{right} leads to state s_2 (upper middle) yielding reward $r = 0$. q-estimate for this transition was 72.

Update estimate by Q-learning (γ=0,9):

$$q(s_1,a_{right}) \leftarrow r + \gamma \max_{a'} q(s_2,a')$$

$$\leftarrow 0 + 0{,}9 \max(63, 81, 100)$$

$$\leftarrow 90.$$

Q-Learning

- When the agent moves forward from *s* to *s'*, learning propagates the estimate backward from *s'* to *s*, using the received reward *r* to improve the estimation.
- Training proceeds in ***episodes***: In each episode, the agent starts at some random state and acts until the (absorbing) goal state is reached.
- Since all rewards are 0 except for transition to the goal state, the *q*-values will remain 0 until the goal state has been reached for the first time.
- Then the non-zero reward for the goal transition will be propagated to the previous state.
- Thus the goal reward will gradually spread from the goal throughout the other states over the episodes, refining *q* ever more.
- Convergence to *Q* requires non-negative rewards.

Q-Learning

- So far, no strategy to choose actions has been supplied.
- Simple strategy: Always choose action that maximizes $q(s,a)$.
- Problem: Agent will prefer good "paths" that have been found in the beginning of the training. Other regions of the state-action space may be neglected.
- Alternative: Probabilistic choice of actions, e.g., by choosing the next action according to the probability

$$P(a_i \mid s) = k^{q(s,a_i)} / \sum_j k^{q(s,a_j)}.$$

where $k>0$ is a constant that determines how strongly high q-values are favored.

- Thus, the agent can explore the state-action space.

Summary

- RL enables an agent to learn actions to reach a goal.
- Q-learning is the most popular algorithm.
- q-table may be replaced by better means, e.g., a neural network.
- Additional strategy for active exploration of state-action space necessary.
- Generalization to non-deterministic case possible.
- Numerous applications, e.g., in robotics or games.

Image sources

[M] Online material available at www.cs.cmu.edu/~tom/mlbook.html
for the textbook: Tom M. Mitchell: *Machine Learning*, McGraw-Hill

[RN] Stuart Russel & Peter Norvig: KI (Pearson)

[H] Gunther Heidemann, 2012.