

# Machine Intelligence

## 14. Reinforcement Learning

### Learning From Experience

Álvaro Torralba



AALBORG UNIVERSITET

Fall 2022

Thanks to Thomas D. Nielsen and Jörg Hoffmann for slide sources

## In Previous Chapters of the ML Lecture...

### Supervised Learning (Chapters 7-9)

- Learn a function from input/output examples
- Task: predict some (unobserved) **target** or **class** variable based on observed values of **(predictor) attributes**
  - **Regression**, if target is continuous
  - **Classification**, if target is discrete
- Model types e.g.: Decision trees,  $k$ -nearest neighbors, Neural networks, Naive Bayes,...

### Unsupervised Learning (Chapter 10)

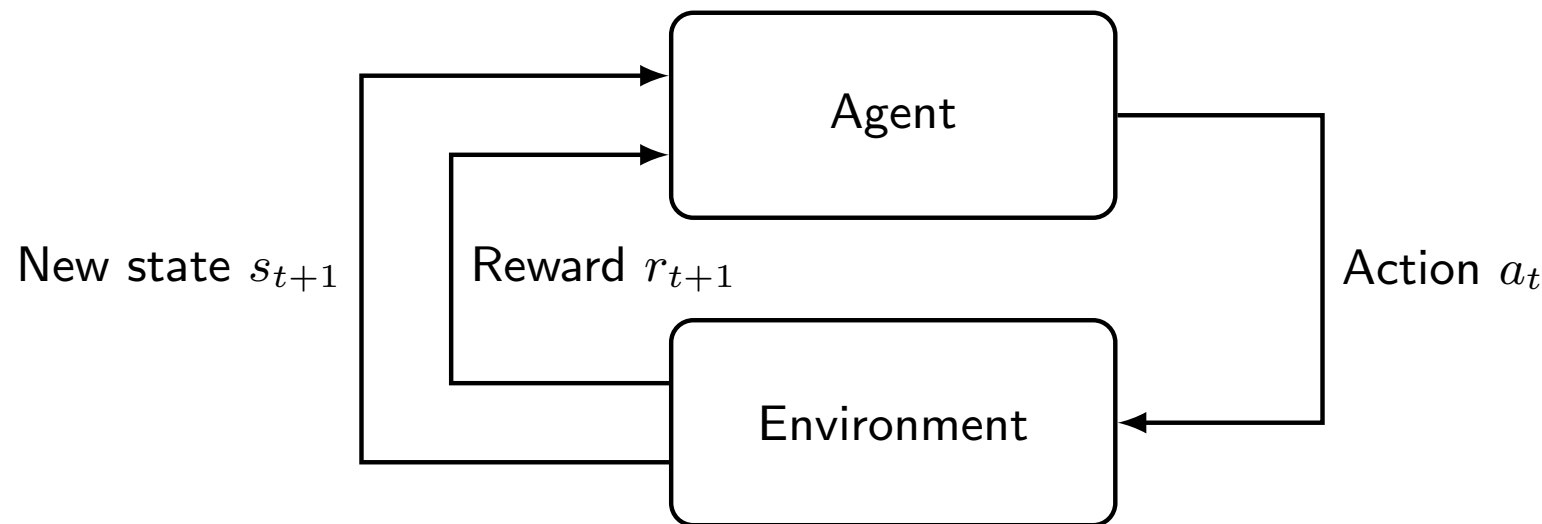
- Learn patterns in the input data
- Task: **Clustering**: identify coherent subgroups in data
- Model types e.g.:  $k$ -means, hierarchical clustering, Self-organizing maps, probabilistic clustering,...

### Reinforcement learning (This Chapter)

- Task: Learn how to take actions in an environment aiming to maximize a cumulative reward

# Reinforcement Learning

**Reinforcement Learning:** Learn when acting in an environment



- Agent interacts with an environment
- Every time it executes an action, the agent gets from the environment:
  - Observations: here, we assume it observes the full next state
  - **Reward** signal: that the agent wants to maximize
- The agent needs to learn in an **online** fashion, as its acting in the environment

## Recap from Chapter 13

**Definition (Markov Decision Process).** An **MDP** is a 6-tuple  $\Theta = (S, A, r, P, I, S^T)$  where:

- $S$  is a finite set of **states**.
- $A$  is a finite set of **actions**.
- $r : S \times A \times S \mapsto \mathbb{R}$  is the **reward function**.
- $P : S \times A \times S \mapsto [0, 1]$  is the **transition probability function**, representing  $P(s' \mid s, a)$ , the probability that by applying action  $a$  in state  $s$ , we end in state  $s'$ .
- $I \in S$  is the **initial state**.
- $S^T \subseteq S$  is the set of **terminal states**. The set of terminal states could be empty.

So, what is the difference?

**Probabilistic Planning:** Given a known MDP, how to compute the optimal policy.

**Reinforcement Learning:** What if transition probabilities (and rewards) are unknown a priori?

→ **We need to interact with the environment in order to learn!**

# Model-free versus Model-based Reinforcement Learning

## Question

So, what should we learn?

# Model-free Reinforcement Learning

We do not need to learn how the environment works. It may be enough to learn what to do in each state.

In the previous lecture, we considered:

**Value function**  $V^*(s)$ : expected utility starting in  $s$  and afterwards acting optimally.

Here, we consider instead:

**Q-value**  $Q^*(s, a)$ : expected utility starting in  $s$ , applying action  $a$ , and afterwards acting optimally.

## Aside: Online Mean Estimation

**Difficulty:** Compared to supervised learning, we received experiences one by one. One option is to accumulate all previous experiences, construct a database and learn from such dataset

However:

- Is that how you learn?
- Is it really necessary to memorize all previous experiences?

Let's consider a different problem for a moment. We want to compute the average value ( $\hat{x}$ ) of a list of numbers  $x_1, x_2, x_3, \dots$ , but we receive numbers one by one.

- $x_1$ : 10  $\rightarrow$  average = 10
- $x_2$ : 20  $\rightarrow$  average = 15
- $x_3$ : 9  $\rightarrow$  average = 13
- $\dots \rightarrow$  average = 11
- $x_{10}$ : 30  $\rightarrow$  average =

$$\hat{x}_n = \hat{x}_{n-1} + \frac{x_n - \hat{x}_{n-1}}{n}$$

We update our belief based on each observation (and we do not need to explicitly store all the individual observations).  $\rightarrow$  We can do the same in RL

# Q-learning

- Learn an approximate model of  $Q^*(s, a)$  directly.
- Update current estimate  $Q$  of  $Q^*$  after each experience  $\langle s_t, a_t, r_t, s_{t+1} \rangle$

## Q-learning update

After  $\langle s, a, r, s' \rangle$  update  $Q(s, a)$ :

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$$

- We do not need to know the transition probabilities or the reward function, because we get  $s'$  and  $r$  directly by sampling the environment.
- $\alpha$  is the **learning rate**. It is a parameter in  $[0, 1]$  that controls how much we weight our previous experiences and the new one.

## Question!

Does this take into account the probabilities in the transition function?



# Tabular Q-Learning Algorithm

**Input:** States  $S$ , Actions  $A$ , Discount factor  $\gamma$ , Learning rate  $\alpha$

```




1 Initialize  $Q(s, a)$  arbitrarily (e.g. initialize to 0 for all  $s, a$ );
2  $s \leftarrow$  Initial state;
3 repeat
4    $a \leftarrow$  Select action ;
5    $s', r \leftarrow$  execute  $a$  in  $s$  and observe state and reward ;
6    $\text{expected\_value} \leftarrow r + \gamma \cdot \max_{a' \in A} Q(s', a')$ ;
7    $Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot \text{expected\_value}$  ;
8    $s \leftarrow s'$  ;
9   if  $s$  is terminal then
10     $s \leftarrow$  start new episode (from initial state or some arbitrary state;
11 until until convergence;
```

## Algorithm 1: Q-learning

- As the agent interacts with the environment by executing action, it learns the  $Q$ -value function
- This allows the agent to select better actions in the future (we discuss how actions are selected in the next slides)

## Q-learning: Example!

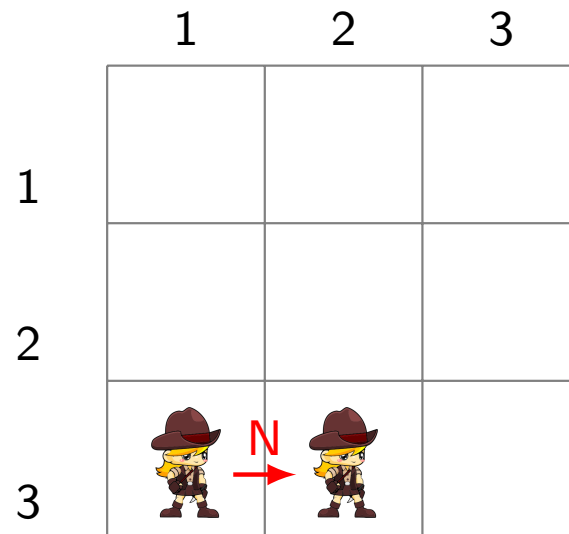
$$\alpha = 0.5, \gamma = 0.9$$

	1	2	3
1			
2			
3			

A 3x3 grid of triangles. The center triangle (row 2, column 2) contains the value -25. The triangle at row 2, column 1 contains the value 0 in red. All other triangles contain the value 0.

# Q-learning: Example!

$$\alpha = 0.5, \gamma = 0.9$$



<div><div>2</div><div>-2</div><div>3</div></div>	<div><div>1</div><div>-5</div><div>-4</div></div>	<div><div>2</div><div>4</div><div>1</div></div>
<div><div>3</div><div>0</div><div>0</div></div>	<div><div>1</div><div>-2</div><div>-4</div></div>	<div><div>-2</div><div>3</div><div>2</div></div>
<div><div>4</div><div>2</div><div>2</div></div>	<div><div>-2</div><div>-4</div><div>-3</div></div>	<div><div>0</div><div>3</div><div>2</div></div>

## Question!

Pick an action: north, you get -1.2 of reward!  
Which cell is updated, and to which value?

# Exploration versus Exploitation

When choosing and executing action  $a$ , one has to balance two objectives:

**Exploration:** Choose  $a$  for which the estimate  $P(S' | s, a)$  is not yet accurate and/or the reward  $R(s, a)$  is not yet known.

**Exploitation:** Choose  $a$  for which  $R(s, a)$  is high, and/or  $P(S' | s, a)$  gives high probability to states  $s'$  with high  $U^c(s')$ .

## Action Selection: $\epsilon$ -greedy

How should the agent select actions?

$\epsilon$ -greedy strategy: For some  $0 < \epsilon < 1$ .

- With probability  $\epsilon$ : Pick an action randomly
- With probability  $(1 - \epsilon)$ : Pick the action with the highest  $Q$ -value  
→  $\operatorname{argmax}_a Q(s, a)$  where  $s$  is the current state

Typically the value of  $\epsilon$  is close to 0, but there is also the possibility of starting with higher values and decreasing  $\epsilon$  as the agent gains experience

# Q-Learning Properties

Q-Learning is **guaranteed to converge to the optimal value if** the following properties hold:

① **All states and actions are visited infinitely often**

→ For convergence you don't need to be acting optimally (or even close enough), but you must do enough exploration

② **Adjust the learning rate**  $\alpha$  such that  $\sum_{t=0}^{\infty} \alpha_t = \infty$  and  $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$

So,  $\alpha$  needs to be decreased over time (at some point your experience is more important than any new information you obtain from the environment) but without reaching 0 (never stop learning!)

Example:

$$1 + \frac{1}{2} + \frac{1}{3} \cdots = \infty$$

$$1 + \frac{1}{2^2} + \frac{1}{3^2} \cdots = \frac{\pi^2}{6} < \infty$$

# Beyond Tabular Representations

**Deep Reinforcement Learning:** Use a Neural Network to represent the Q-value function

Advantages:

- Allows us to use this ideas on problems with exponentially-many states!
- Allows the agent to generalize and use experiences on some states to update the Q-value on similar states.

# Evaluation of RL Agents

- How much reward they accumulate over time.
  - They should perform well as they learn (so this evaluates both how much the agent learns as well as the exploitation/exploration strategy)
- After training for a fixed amount of time, how good is the resulting policy
  - While evaluating the policy, the agent only selects actions greedily (is not focused on exploration anymore)



# Real World versus Simulator

Learning in the real world is difficult (and may be dangerous)

- When executing actions, the agent could cause harm!  
→ Are you sure you want to do random exploration in real life?
- Gaining real experience is very expensive  
→ For example, if it requires interacting with users

Potential solution: use a **simulated environment**

Drawback: If you don't know the transition probabilities or the reward function, how do you construct such a simulator?

→ **If you have the model of the environment, you should also consider (probabilistic) planning algorithms**

# Model-based Reinforcement Learning

Can learn transition probabilities from example traces:

$$t_1 : s_{0,1} a_{0,1} s_{1,1} a_{1,1} s_{2,1} a_{2,1} s_{3,1} \dots$$

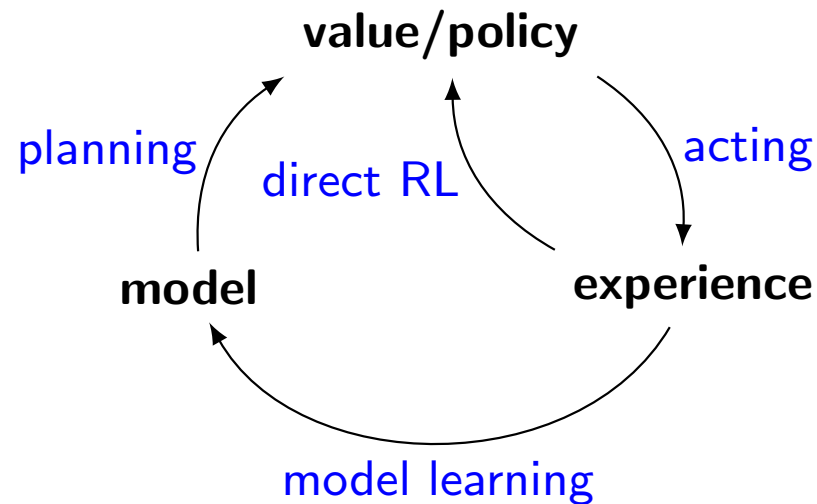
$$t_2 : s_{0,2} a_{0,2} s_{1,2} a_{1,2} s_{2,2} a_{2,2} s_{3,2} \dots$$

...

- Known are state and action sets  $S, A$ .
- Maintain current model  $P^c, U^c$  for transition probabilities and value function
- Do:
  - observe current state  $s$
  - choose and execute action  $a$
  - observe new state  $s'$  and reward  $R(s, a)$
  - update estimate for  $P(s' | s, a)$  and  $U(s)$

At every step, we can apply planning in order to act optimally according to our current belief over the transition probabilities and value function!

# Summary: MDPs and RL



## Known MDP: Offline solution

**Goal**

Compute  $V^*$ ,  $Q^*$ ,  $\pi^*$   
Evaluate a fixed policy  $\pi$

**Technique**

Value/policy iteration  
Policy evaluation

## Unknown MDP: Model-based

**Goal**

Compute  $V^*$ ,  $Q^*$ ,  $\pi^*$   
Evaluate a fixed policy  $\pi$

**Technique**

VI/PI on approx. MDP  
PE on approx. MDP

## Unknown MDP: Model-free

**Goal**

Compute  $V^*$ ,  $Q^*$ ,  $\pi^*$   
Evaluate a fixed policy  $\pi$

**Technique**

Q-Learning  
TD-Learning

# Conclusions

- **Reinforcement Learning** is how an agent learns how to act by acting in an environment and receiving a certain **reward** signal.
- **Q-learning** is an RL algorithm that learns the Q-value function.
- Q-learning always converges to the optimal value function if there is enough exploration, and the learning rate is adjusted in the right way.

## Further Material

- If you are interested in the topic, you can watch the 6-lecture videos of Pieter Abbeel on Reinforcement Learning, where they cover parts of what we saw in Chapters 13 and 14: <https://www.youtube.com/watch?v=2GwBez0D20A>

# The exam

- January 6th, 2022, 10:00 to 14:00.
- Written exam with internal censors.
- Answers should be written in English.
- We will use the IT-Flex Platform. Software that you can use:
  - PDF viewers (in particular, you can download and check during the exam all material from the lecture)
    - However, don't rely on doing this, the exam requires time and if you have to check for most of the questions, there won't be enough time for everything
  - Calculator
  - Drawing software to write down the solutions of the exercises
- Software that you cannot use:
  - Software like Weka, or Hugin is not needed
  - Excel or other programming tools

## What the course has covered

### The course has covered the following issues:

- Introduction
- Search-based methods
- Constrained satisfaction problems
- Logic-based knowledge representation
- Reasoning under uncertainty.
- Bayesian networks
- Inference in Bayesian networks
- Machine learning
- Planning
- Reinforcement Learning

This corresponds to the following literature:

- The slides from the course.  
→ **Main study material. All 14 chapters are relevant for the exam. Parts that are not relevant are marked as “Further material” or “(for reference)”.**  
**All the rest is relevant for the exam.**
- David L. Poole and Alan K. Mackworth, Artificial Intelligence: Foundations of computational agents (Second edition): Preface, Ch. 1, 3-3.6, 3.7-3.7.1, 3.7.3, 3.8.2 - 3.8.3, 4-4.7.3, 5-5.2, 7-7.5 (except 7.4.2), 7.7, 8-8.4.1, 8.6-8.6.5, 9-9.4 (except 9.1.3), 9.5-9.5.2, 10.1.2, 10.2, 11-11.4, 12.1-5, A.3
- Finn V. Jensen and Thomas D. Nielsen, Bayesian networks and decision graphs: Sections shared in Moodle

## Exam Questions

- You may expect written exercises of similar nature to the ones you solved in the exercise sessions.  
→ Of “similar nature” does not mean that the exercises will be like a template, you need to understand the concepts and be able to apply it in different ways
- You can find previous exams on Moodle. This can be used as additional exercises. However, note that they are from previous editions of the course, so don't assume that the new exam will look exactly like those.