Autonomous and Adaptive Systems
course notes
University of Bologna

prof. Mirco Musolesi

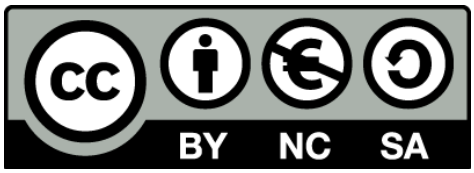
AY. 2020-2021

Alessandro Pomponio

February 2021

Material used for these notes includes:

- Prof. Musolesi's slides, available here: `https://www.mircomusolesi.org/courses/AAS20-21/AAS20-21-main/`
- Richard S. Sutton and Andrew G. Barto's "Reinforcement Learning, An Introduction" book, available here: `http://incompleteideas.net/book/RLbook2020.pdf`



1 Intelligent Systems

1.1 Computing machinery and intelligence

The course starts by reading Turing's article *"Computing machinery and intelligence"*, published on the Mind journal in October 1950 [**10.1093/mind/LIX.236.433**]. In it, Turing complains about the intrinsic ambiguity of the question "can machines think?", which relies on the definition of both "machine" and "think". To reformulate this problem by means of less ambiguous words, he introduces **the imitation game**, in which an interrogator (C) tries to guess by asking questions and receiving answers, which of the other two participants (that are in a different room and that he only knows by means of the labels X and Y) is a man (A) and which is a woman (B). A's goal is to cause C to make the wrong identification, while B's goal is to help the interrogator. To limit the number of indirect clues that the interrogator may have, the answers should be typewritten or repeated by an intermediary.

Turing then suggests replacing the previous question with a new one: *"what will happen when a machine*[1] *takes the part of A in this game? Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman?"*. In a later part of his article, he clarifies that the goal of the game is not to find out *"whether all digital computers would do well in the game nor whether the computers at present available would do well, but whether there are imaginable computers which would do well"*.

In the last chapter, Turing proposes a few solutions on how to tackle the problem of learning machines (which he argues to be a programming problem); he starts by analyzing the human brain, which he says has three components:

- The initial state of the mind (say at birth).
- The education to which it has been subjected.
- Other experience, not to be described as education, to which it has been subjected.

Instead of producing a program to simulate an adult mind, he suggests producing one that simulates a child's brain and subject it to an appropriate

[1] Later he restricts the definition of machine to digital computers

"education" to obtain an adult brain. He is very aware that *"we cannot expect to find a good child machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution [...]"*. The teaching process will have to involve punishments and rewards: *"the machine has to be so constructed that events which shortly preceded the occurrence of a punishment signal are unlikely to be repeated, whereas a reward signal increased the probability of repetition of the vents which led up to it"*. Finally, he also foresees one of the problems that is still unsolved in the machine learning field: *"an important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil's behavior"*.

1.2 Intelligent machines

After seeing a few examples of autonomous systems, like self-driving cars and agents that play videogames, it is clear that our goal is to be able to build machines that can learn from experience, trying things out on their own, without any human intervention. We first start by giving some definitions.

1.2.1 Intelligent agents

We define an **intelligent agent** as an entity that perceives its environment and takes actions that maximize the probability of achieving its goals. It is important to note that the agent does not know what set of actions will allow it to reach the goal, it just "moves" towards actions that maximize the probability of it happening. Agents may also be physically situated (we call them **robots**) or not (we refer to them as **software agents** or **bots**).

1.2.2 Adaptive agents

We define an **adaptive agent** as an entity that can respond to changes in its environment. This is possible thanks to a lack of determinism: the agent will adapt and react to the environment (which may also include other agents) and take different actions.

Learning can take place in various ways: at the end of a generation, with **natural selection** and the survival of the fittest, or during a generation,

with a method that is more similar to **reinforcement learning**.

1.2.3 Autonomous agents

We define an **autonomous agent** as an entity that only relies on its perception and acts in the world independently from its designer. A key characteristic of this type of agent is that they should be able to compensate for partial knowledge: in the beginning they may only know how to perceive the environment and how to take a certain set of actions; from a practical point of view, it makes sense to provide the agent with some knowledge of the world and the ability to learn. After sufficient experience of its environment, an intelligent agent can become effectively independent of its prior knowledge.

1.2.4 Designing agents

When designing agents, we need to take into consideration the following dimensions:

- Performance: how "good" is the agent.
- Environment: what is "around" the agent.
- Actuators: how the agent can take actions in the environment.
- Sensors: how the agent perceives the environment.

A schema of how these dimensions are linked can be found in figure 1, along with a few examples of agents in figure 2.

1.3 Characteristics of the environments

The environment in which our agent is in may be of different types:

- **Fully observable vs partially observable**: we may or may not be able to see the entire environment (e.g., there may be occlusions limiting our sight).
- **Deterministic vs stochastic**: the environment may be predictable (e.g., governed by the laws of Newtonian physics) or not (e.g., the environment may be changed by another agent).

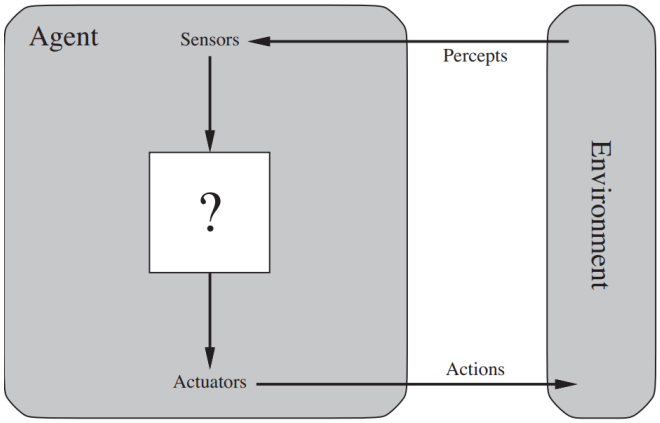


Figure 1: Schema of the interaction between an agent and the environment.

- **Episodic vs sequential**: the environment may be divided in "episodes" that have a beginning and an end (e.g., the levels of a game) or open-ended (e.g., a self-driving car that keeps driving).
- **Static vs dynamic**: the environment may or may not change over time (an action that we take now could have a different result compared to when we took it in the past, e.g., certain agents with whom we collaborated in the past, may not do so anymore; driving in dry conditions is different compared to driving in the rain or in the snow).
- **Discrete vs continuous**: the environment may be discrete or continuous (e.g., the wind speed is a continuous attribute of the environment).
- **Single agent or multi-agent**: there may be multiple agents in the environment, and we may want them to collaborate.

1.4 A categorization of intelligent agents

There are essentially four basic kinds of agent programs:

- **Simple reflex agents**.
- **Model-based reflex agents**.
- **Goal-based agents**.

	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Health patient, minimise cost, lawsuit	Patient, hospital, staff	Display questions, tests, treatments, etc.	Keyboard entry, patients' answers
Satellite image analysis system	Correct image categorisation	Downlink from satellite	Display categorisation of scenes	Colour pixel arrays
Part-picking robot	Percentage parts in correct bins	Conveyor belt with parts, bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Maximise purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, sensors
Interactive language tutor	Maximise student's test score	Set of students	Displays exercises, suggestions	Keyboard entry
Automatic display of advertisements	Click rates/ purchase conversion	Websites, online retailers, users	Display advertisements	Automatic extraction of content, clicks

Figure 2: Examples of agents and their characteristics.

- **Utility-based agents**.

The behavior of these agents can be hard-wired or it can be acquired, improved and optimized through learning.

1.4.1 Simple reflex agents

Simple reflex agents select actions on the basis of the current perceptions, ignoring the perception history. They are the most basic form of agents and are based on condition-action rules (also called simulation-action rules, productions, or if-then rules). (See figure 3)

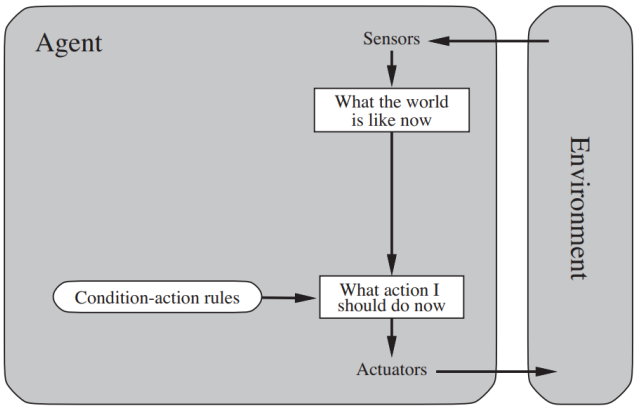


Figure 3: Schema of a simple reflex agent.

1.4.2 Model-based reflex agents

Model-based agents keep an internal state and depend on two types of knowledge:

- How the world evolves independently from the agent (e.g., the trajectory that a bullet/a stone follows when shot/thrown).
- How the actions of the agent affect the world (e.g., if I turn the wheel to the right, the car moves to the right).

The internal state is essentially used to keep track of what it is not possible to see/perceive right now. It depends on the perception history and, for this reason, it reflects at least some of the unobserved aspects of the current state. (See figure 4)

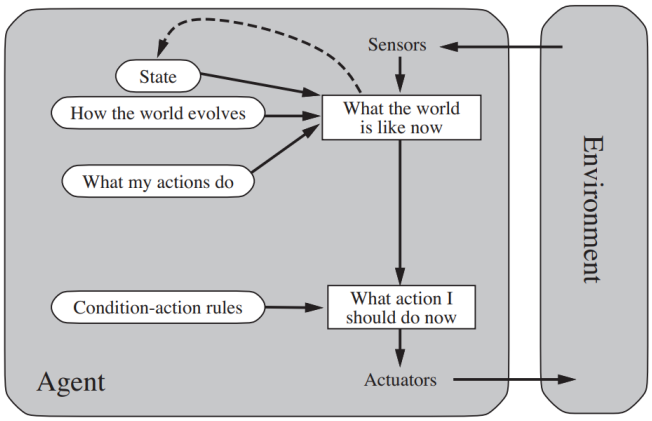


Figure 4: Schema of a model-based reflex agent.

1.4.3 (Model-based) Goal-based agents

Goal-based agents act in order to achieve their goals. If we can achieve the goal by carrying out a single action, goal-based action selection is straightforward; in the other cases, the agent needs to consider a long sequence of actions by means of search and planning. (See figure 5)

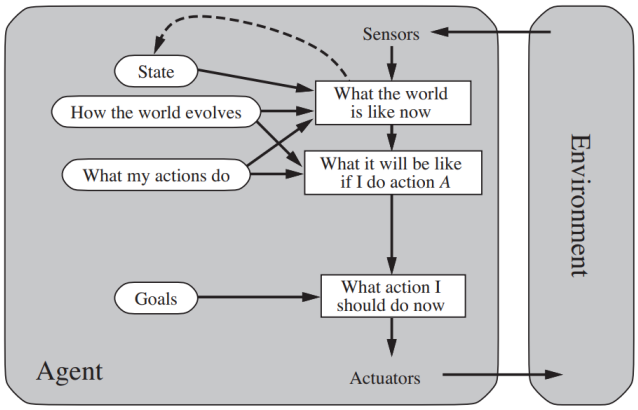


Figure 5: Schema of a goal-based agent.

1.4.4 (Model-based) Utility-based agents

Goals are not sufficient to generate "high-quality behavior" in most environments, since there are usually states that are preferrable to others. In order

to code this "preference", we use utility functions that map a state (or a sequence of states) to a real number (e.g., we want to get to a destination by following the shortest or quickest path). (See figure 6)

Note that how to model these preferences is one of the current unsolved and "hot" topics in the artificial intelligence field.

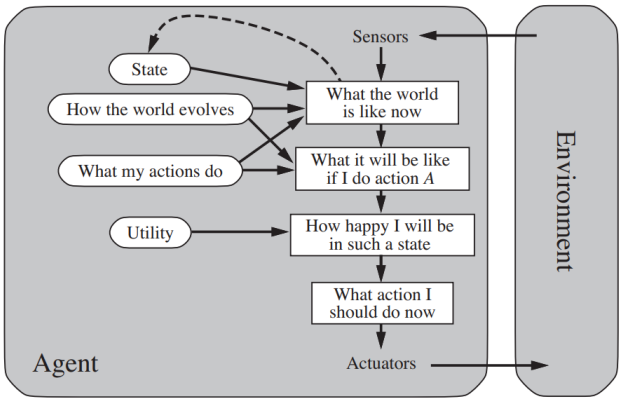


Figure 6: Schema of a utility-based agent.

To conclude this first introduction, we now have a look at learning.

1.5 Learning

As we have said before, the behavior of the agents can be pre-programmed (hard-wired, fixed) or it can be learned by means of a learning component. This component can be based on a model of the world and the gain towards a certain goal (possibly expressed in terms of the change of the value of utility functions) can be expressed through rewards. This behavior is at the basis of the type of learning that we will explore in detail in this course, called **reinforcement learning**. Following Herbert Simon's definition of autonomous and adaptive systems, we will consider *"machines that think, that learn and that create"*.

2 Introduction to Reinforcement Learning

Learning from interaction is an idea that is at the basis of nearly all theories of learning and intelligence, among which we find reinforcement learning.

Reinforcement learning is learning what to do and how to map situations to actions, so as to maximize a numerical reward (it is goal-directed learning from interaction). The learner is not told which actions to take, but instead it must discover which actions yield the most reward by trying them.

We will now introduce finite Markov decision processes, a mathematical framework that we are going to use.

2.1 Finite Markov Decision Processes

Markov Decision Processes (MDPs) are a mathematically idealized formulation of reinforcement learning for which precise theoretical statements can be made. They provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.

At aech time step, the process is in stome state s, and the decision maker may choose an action a that is available in state s. The process responds at the next time step by randomly moving into a new state s', and giving the decision maker a corresponding reward $R_a(s, s')$. The probability that the process moves into its new state s' is influenced by the chosen action. Specifically, it is given by the state transition function $P_a(s, s')$. Thus, the next state s' depends on the current state s and the decision maker's action a. But **given s and a, it is conditionally independent of all previous states and actions**; in other words, the state transitions of an MDP satisfy the Markov property *(memoryless property of a stochastic process)*.

2.2 Rewards and expected returns

Informally, the agent's goal is to maximize the total amount of rewards it receives (note how the agent should not maximize the immediate reward, but the cumulative reward). We can formalize the goal of an agent by stating the **"reward hypothesis"**: *"all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (reward)"*.

We will now try to conceptualize the idea of **cumulative rewards** more formally by means of the notion of **expected return** G_t. We first need to distinguish between two cases:

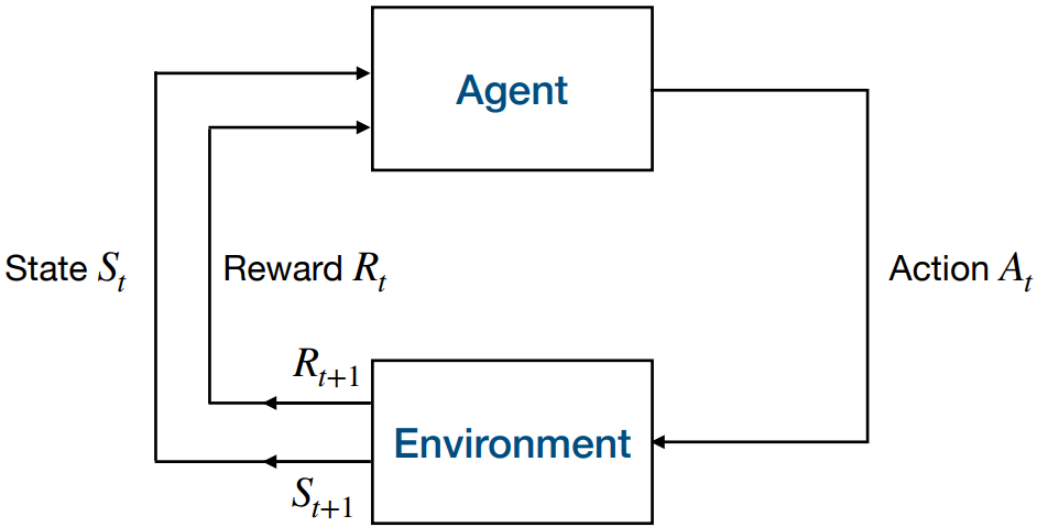


Figure 7: Schema of a Markov Decision Process.

- **Episodic tasks**, in which we can identify a final step of the sequence of rewards (i.e., in which the interaction between the agent and the environment can be broken into sub-sequences called **episodes**, such as playing a game, repeated tasks, etc.). Each episode ends in a terminal state after T steps, followed by a reset to a standard starting state or to a sample of a distribution of starting states (the next episode will be completely independent from the previous one).
- **Continuous tasks**, in which we cannot identify a final state (e.g., an ongoing monitoring of a process).

We assume that, over time, an agent receives a sequence of rewards $R_{t+1}, R_{t+2}, R_{t+3}$, and we say that:

- In the case of episodic tasks, the expected return associated to the selection of an action A_t is the sum of rewards defined as follows:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_t \tag{1}$$

- In the case of continuing tasks, the expected return associated to the

selection of an action A_t is defined as follows:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2}$$

Where γ is the **discount rate**, with $0 \leq \gamma \leq 1$. The discount rate is used to give more importance to the rewards that are closer to us in time; this is particularly useful in dynamic environments. The definition of expected return that we used for episodic tasks would be problematic for continuing tasks: the expected return at the time of termination T would be equal to ∞ in some cases, such as when the reward is equal to 1 at each time step. The discount rate determines the "present value of future rewards" (how much future rewards mean to us at the current time): a reward received k time steps in the future is worth γ^{k-1} of what it would be worth if it were received immediately.

Returns at successive time steps are related to each other as follows:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + ... \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

2.3 Policies and value functions

Almost all reinforcement learning algorithms involve estimating value functions, i.e., functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state).

A **policy** is used to model how the agent will behave based on the previous experience and the rewards (and, consequently, the expected returns) an agent received in the past. Formally, a policy is a mapping from states to probabilities of each possible action (the probability of taking a certain action in a certain state). If the agent is following the policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$.

The value function of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter (the expected return I can have in the future state, considering all the actions I might take

from there). For Markov Decision Processes, we can define the state-value function v_π for the policy π formally as:

$$v_s \doteq E_\pi \left[G_t | S_t = s\right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \quad \forall s \in \mathcal{S} \qquad (3)$$

Where $E_\pi[.]$ denotes the expected value of a random variable given that the agent follows π and t is any time step. The value of the terminal state is 0. The formula above denotes a weighted average of the expected value (it is averaged because the values depend on the probability of a certain action being taken, which is a fraction).

Similar to what we just did, we can define the action-value function, i.e., the value of taking an action a in the state s under a policy π, denoted $q_\pi(s, a)$, as the expected return starting from s, taking the action a, and following the policy π thereafter.

2.4 Choosing the rewards

When we model a real system as a reinforcement learning problem, the most difficult task is selecting the right rewards. Typically, we use negative values for actions that do not help us in reaching our goal, and positive if they do (it is also possible using 0 as a value for actions that do not help us). An alternative, is to set the values of rewards to a negative number until we reach our goal (using 0 as the value when we reach it).

When choosing the rewards, it is very important that **we should not "reward" the intermediate steps or the single actions**. This is because we do not want to "teach" the agent how to execute an intermediate step, but how to reach the final goal. If we did that, the agent would only learn how to reach that intermediate step (e.g., how to execute a sub-task).