# Chapter 1

# Introduction

Nowadays every aspect of life is characterized by the need to make choices. Each choice is conditioned by a variable number of parameters. In many applicative domains (recommendation systems, medical analysis tools, real time game engines, speech recognizers...), this number could be very high. The possibility to manage in an efficient way such a number of parameters is guaranteed by a significant number of mathematical techniques increasingly performing and refined.

Every decisional problem can be translated into a *mathematical model* that represents the essence of the problem itself. A crucial step in formulating a model is the construction of the *objective function*. This requires developing a quantitative measure of performance relative to each of the decision maker's ultimate objectives that were identified while the problem was being defined. [HL01]

However we know that it is not always possible to define the objective function in advance. It is possible that objective function is unknown or, at least, partially unknown at the time the problem is dealt with. In these cases the optimization process of the objective function takes the name of *black-box optimization*.

Mathematically, we are considering the problem of finding a global maximizer (or minimizer) of an unknown objective function $f$ :

$$x^* = \arg\max_{x \in \mathcal{X}} f(x) \tag{1.1}$$

where $\mathcal{X}$ is some design space of interest. In global optimization, $\mathcal{X}$ is often a compact subset of $\mathbb{R}^d$ [SSW+16].

Bayesian optimization is one of the best known black-box optimization techniques. It is a powerful tool for the joint optimization of design choices that is gaining great popularity in recent years. It is a sequential model-based approach to solving problem [AS08]. We prescribe a prior beliefs over the possible objective functions and then sequentially refine this model as data are observed via Bayesian posterior updating. The Bayesian Posterior represents our updated beliefs -given data- on the likely objective function we are optimizing [SSW+16].

In this thesis I propose an innovative approach to black-box optimization based on the Reinforcement Learning (RL) technique.

Reinforcement Learning (RL) is the problem faced by a learner that must behaviour through trial-and-error interactions with a dynamic environment. It can be considered the problem of mapping situations to actions in order to maximize a numerical reward signal [KLM96].

In the first part of this work I will compare the state of the art performances of the Bayesian model with those obtained through the implementation of a Reinforcement Learning (RL) agent in the same order.

In the last part of the thesis I will explain how RL and Bayesian optimization can be joined to emulate human brain learning process.

# Chapter 2

# Background

## 2.1 Markov Decision Process

Markov Decision Processes (MDPs) are a classical formalization of sequential decision making under uncertainty, where actions influence not just immediate responses, but also subsequent situations. Hence an MDP can be described as a controlled *Markov chain*[1], where the control is given at each step by the chosen action. In this chapter we will present the structure of an MDP and many techniques to solve its.

**Markov Decision Process** Markov decision processes are defined as controlled stochastic processes satisfying the *Markov property*[2] and assigning reward values to state transitions [Put94]. Formally, they are described by the 5-tuple ($S$, $A$, $T$, $p$, $r$) where:

- $S$ is the state space in which the process' evolution takes place;
- $A$ is the set of all possible actions which control the state dynamics;
- $T$ is the set of time steps where decisions need to be made;
- $p()$ denotes the state transition probability function;
- $r()$ provides the reward function defined on state transitions.

[1] A sequence of random variables $X_0, X_1, ...$ with values in a countable set $S$ is a *Markov chain* if at any time $n$, the future states (or values) $X_{n+1}, X_{n+2}, ...$ depend on the history $X_0, ..., X_n$ only through the present state $X_n$ [Kon09] .

[2] A stochastic process has the *Markov property* if the probabilistic behaviour of the chain in the future depends only on its present value and discards its past behaviour.

**States** The set of environmental states $S$ is defined as the finite set $\{s_1, ..., s_N\}$ where the size of the state space is $N$, i.e. $|S| = N$ [WvO12]. Each state $s \in S$ is a vector of attributes (*state variables*) that describes the current configuration of the system [NFK06]. More specifically, a state variable is the minimally dimensioned function of history that is necessary and sufficient to compute the "state of knowledge" [Pow07].

**Actions** The set of actions $A$ is defined as the finite set $\{a_1, ..., a_K\}$ where the size of the action space is $K$, i.e. $|A| = K$. Actions can be used to control the system state. The set of actions that can be applied in some particular state $s \in S$, is denoted $A(s)$, where $A(s) \subseteq A$. In more structured representations the fact that some actions are not applicable in some states, is modelled by a precondition function : $S \times A \rightarrow true, false$, stating whether action $a \in A$ is applicable in state $s \in S$ [WvO12].

**Time Steps** The set of time steps $T$ is defined as the finite set $\{t_1, ..., t_M\}$ where the size of the action space is $M$, i.e. $|T| = M$.

**Transition Probability** The transition probabilities $p()$ characterize the state dynamics of the system, i.e. indicate which states are likely to appear after the current state. For a given action $a$, $p(s'|s, a)$ represents the probability for the system to transit to state $s'$ after undertaking action $a$ in state $s$. This $p()$ function is usually represented in matrix form where we write $P_a$ the $|S|$ x $|S|$ matrix containing elements $\forall s, s', P_{\text{a, s, s'}} = p(s'|s, a)$. Since each line of these matrices sums to one, the $P_a$ are said to be stochastic matrices. The $p()$ probability distributions over the next state $s'$ follow the fundamental property which gives their name to Markov decision processes. If we write $h_t = (s_0, a_0, ..., s_{\text{t-1}}, a_{\text{a-1}}, s_t)$ the history of states and actions until time step $t$, then the probability of reaching state $s_{\text{t+1}}$ consecutively to action $a_t$ is only a function of $a_t$ and $s_t$, and not of the entire history $h_t$ [SB10]. We can resume this concept through the following equation :

$$\forall h_t, a_t, s_{\text{t+1}} \quad P(s_{\text{t+1}}|h_t, a_t) = P(s_{\text{t+1}}|s_t, a_t) = p(s_{\text{t+1}}|s_t, a_t) \tag{2.1}$$

**Reward Function** The reward function specifies rewards for being in a state, or doing some action in a state. The state reward function is defined as $R : S \rightarrow \mathbb{R}$, and it specifies the reward obtained in states. The reward function is an important part of the MDP that specifies implicitly

the *goal* of learning. Thus, the reward function is used to give direction in which way the system, i.e. the MDP, should be controlled [WvO12]. It is critical that the rewards we set up truly indicate what we want accomplished. If we reward the achievement of subgoals , then the agent might find a way to achieve them without achieving the real goal. Reward signal is our way of communicating the agent *what* we want to achieve, not *how* it achieved [SB18].

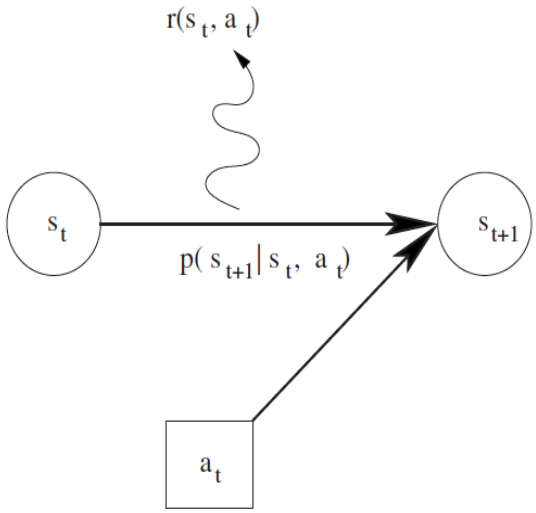


Figure 2.1: Markov decision process [SB10].

Markov decision processes allow us to model the state evolution dynamics of a stochastic system when this system is controlled by an agent choosing and applying the actions $a_t$ at every time step $t$. The procedure of choosing such actions is called an action policy, or strategy, and is written as $\pi$ [SB10] .

**Policy** Formally, given an MDP $\langle S, A, p(), r() \rangle$, a policy is a computable function that outputs for each state $s \in S$ an action $a \in A(s)$ [WvO12] A policy can decide deterministically upon the action to apply or can define a probability deistribution over the possible applicable actions. Then, a policy can be based on the whole history $h_t$ (history-dependent policy) or can only consider the current state $s_t$. Thus, we can obtain four main families policies, as shown in table 2.1.

For a deterministic policy, $\pi_t(s_t)$ or $\pi_t(h_t)$ defines the chosen action $a_t$.

| Policy $\pi_t$ | Deterministic | Stochastic |
|---|---|---|
| Markov | $s_t \rightarrow a_t$ | $a_t, s_t \rightarrow [0,1]$ |
| History-dependent | $h_t \rightarrow a_t$ | $h_t, s_t \rightarrow [0,1]$ |

Table 2.1: Different policy families for MDPs [SB10]

For a stochastic policy, $\pi_t(a, s_t)$ or $\pi_t(a, h_t)$ represents the probability of selecting $a \in A$ for $a_t$ [SB10]. The sets so defined are included in each other, from the most general case of stochastic, history-dependent policies, to the very specific case of deterministic, Markov policies, as shown in figure 2.2.

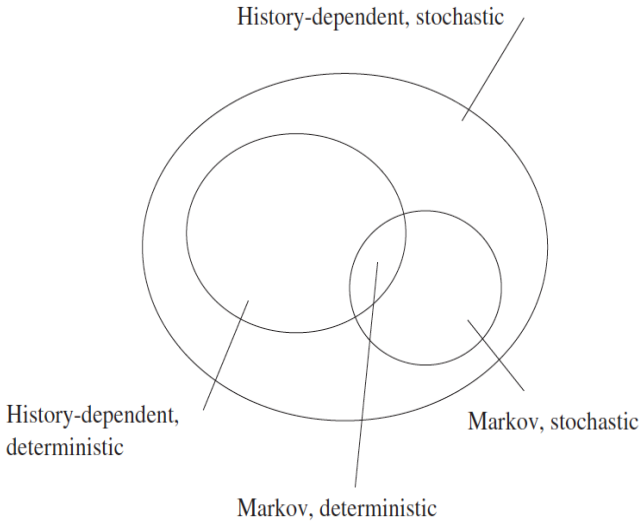


Figure 2.2: Relationship between the different sets of policies [SB10].

Application of a policy to an MDP is done in the following way. As shown in figure 2.3 each time an agent performs an action $a_\mathrm{t}$ in a state $s_\mathrm{t}$, it receives a real-valued reward $t_\mathrm{t}$ that indicates the immediate value of this state-action transition. This produces a sequence of states $s_\mathrm{i}$, actions $a_\mathrm{i}$, and immediate rewards $r_\mathrm{i}$ as shown in the figure. The agent's task is to learn a control policy, $\{\pi : S \longrightarrow A\}$, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay [Mit97].

As already said the goal of learning in an MDP is to gather rewards. There are several ways of taking into account the future in how to behave now. There are basically three models of optimality in the MDP, which are

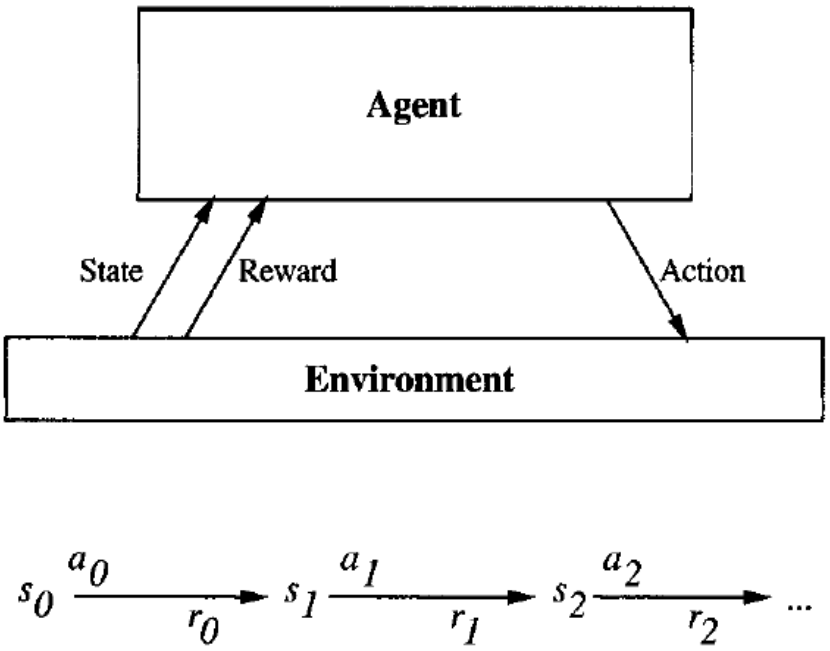


Figure 2.3: Policy application schema [SB18].

sufficient to cover most of the approaches in the literature :

$$E[\sum_{t=0}^{h} r_t] \qquad E[\sum_{t=0}^{\infty} \gamma^t r_t] \qquad \lim_{h\to\infty} E[1/h \sum_{t=0}^{h} r_t]$$

Figure 2.4: Optimality : **a)** finite horizon, **b)** discounted, infinite horizon, **c)** average reward

**Discount Factor** The *finite horizon* model simply takes a finite horizon of length $h$ and states that the agent should optimize its expected reward over this horizon. In the *infinite-horizon model*, the long-run reward is taken into account, but the rewards that are received in the future are discounted according to how far away in time they will be received. A *discount factor* $\gamma$, with $0 \leq \gamma \leq 1$ is used for this. Note that in this discounted case,

rewards obtained later are discounted more than rewards obtained earlier. Additionally, the discount factor, ensures that, even with infinite horizon, the sum of the rewards obtained is finite. In episodic tasks, i.e. in tasks where the horizon is finite, the discount factor is not needed or can equivalently be set to 1. If $\gamma = 0$ the agent is said to be myopic, which means that it is only concerned about immediate rewards. The last optimality model is *average-reward* model, maximizing the long-run *average-reward*. Sometimes this is called the *gain optimal* policy and in the limit, it is equal to the infinite-horizon discounted model [WvO12]

**Bellman Equation** The concept of *value function* is the link between optimality criteria and policies. Most learning algorithms for MDPs compute optimal policies by learning value functions. The value of a state $s$ under policy $\pi$, denoted $V^\pi(s)$ is the expected return when starting in $s$ and following $\pi$ thereafter. Using the infinite-horizon, discounted model :

$$V^\pi(s) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k t_{\mathrm{t+k}} | s_t = s\} \tag{2.2}$$

A similar state-action value function : $Q : S \text{ x } A \to \mathbb{R}$ can be defined as the expected return starting from state $s$. taking action $a$ and thereafter following policy $\pi$ :

$$Q^\pi(s,a) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{\mathrm{t+k}} | s_t = s, a_t = a\} \tag{2.3}$$

For any policy $\pi$ and any state $s$ the expression 2.3 can recursively be defined in terms of a so-called *Bellman Equation* :

$$\begin{aligned} V^\pi(s) &= E_\pi\{r_t + \gamma_{\mathrm{t+1}} + \gamma^2 r_{\mathrm{t+2}} + ... | s_t = t\} \\ &= E_\pi\{r_t + \gamma V^\pi(s_{\mathrm{t+1}}) | s_t = s\} \\ &\sum_{s'} T(s, \pi(s), s')(R(s,a,s') + \gamma V^\pi(s')) \end{aligned} \tag{2.4}$$

It denotes that the expected value of state is defined in terms of immediate reward and values of possible next state weighted by thei transition probabilities, and additionally a discount factor. Note that multiple policies can have the same value function, but for a given policy $\pi$, $V^\pi$ is unique. The goal for any MDP is to find a best policy, i.e. the policy that receives the most reward. This means maximizing the value function of equation 2.3 for

all states $s \in S$. An optimal policy, denoted $\pi^*$, is such that $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $s \in S$ and all policies $\pi$ :

$$V^*(s) = \max_{a \in A} \sum_{s'} T(s, \pi(s), s')(R(s, a, s') + \gamma V^{\pi}(s')) \tag{2.5}$$

This expression is called the *Bellman optimality equation*. It states that the value of a state under an optimal policy must be equal to the expected return for the best action in a state [WvO12].

## 2.2 Solving MDP

Now that we have defined MDPs, policies, optimality criteria and value functions, it is time to consider the question of how to compute optimal policies. As already said, solving an MDP means computing an optimal policy $\pi^*$. Several dimensions exists along which algorithms have been developed for this purpose. The most important distinction is that between *model-based* and *model-free* algorithms [WvO12].

| | **Model-based algorithms** | **Model-free algorithms** |
|---|---|---|
| **General name** | DP | RL |
| **Basic assumption** | A model of the MDP is known beforehand, and can be used to compute value functions and policies using the Bellman equation. | They rely on interaction with the environment. Because a model of the MDP is not known, the agent has to explore the MDP to obtain information. |

Table 2.2: Main differences between model-based and model-free algorithms.

**Dynamic Programming** The term DP refers to a class of algorithms that is able to compute optimal policies in the presence of a perfect model of the environment [WvO12]. The method of dynamic programming systematically records solutions for all sub-problems of increasing lengths. According to Bellman optimality principle, all optimal sub-policies of an optimal policy are optimal sub-policies. Using this programming paradigm the optimal policy is defined through the step-by-step definition of optimal sub-policies. The assumption that a model is available will be hard to ensure for many applications, however DP algorithms are very relevant because they define fundamental computational mechanism which are also used when no model is available.

## 2.3 The nature of Reinforcement Learning

Reinforcement Learning (RL) is the problem faced by a learner that must behavior through trial-and-error interactions with a dynamic environment. It can be considered a problem of mapping situations to actions in order to maximize a numerical reward signal [KLM96].

In RL the learner must select an action to take in each time step: every choice done by the agent changes the environment in an unknown fashion and receives a reward which value is based on the consequences. The objective of the learner is to choose a sequence of actions based on observations of the current environment that maximizes cumulative reward or minimizes cumulative cost over all time steps [LM16].

RL is different both from *supervised learning* and from *unsupervised learning*. It is not a sample of learning from a training set of labelled examples provided by a knowledgeable external supervisor (*supervised learning*) and it is not a sample of searching and finding structure hidden in a collection of unlabelled data (*unsupervised learning*). More specifically we can say that RL can be distinguished from other forms of learning based on the following characteristics :

- Reinforcement Learning deals with temporal sequences. In contrast with non-supervised learning problems where the order in which the examples are presented is not relevant, the choice of an action at a given time step will have consequences on the examples that are received at a subsequent time steps [SB10].
- In contrast with supervised learning, the environment does not tell the agent what would be the best possible action. Instead, the agent may just receive a scalar reward representing the value of its action and it must *explore* the possible alternative actions to determine whether its action was the best or not [SB10].

According to what just said, one of the challenges that arise in RL, and not in other kind of learning, is the trade-off between *exploration* and *exploitation*. To obtain an higher reward, an RL agent must prefer actions that it has tried in the past and found to be effective in producing reward. In order to discover such actions, it has to try actions that it has not selected before. The agent *exploits* what it has already experienced in order to

obtain reward, but it has also to *explore* in order to eventually make better action selections in the future [SB18]. In other words *exploitation* consists of doing again actions which have proven fruitful in the past, whereas *exploration* consists of trying new actions, looking for a larger cumulated reward, but eventually leading to a worse performance. Dealing with the exploration/exploitation trade-off consists of determining how the agent should explore to get as fast as possible a policy that is optimal or close enough to the optimum [SB10].

## 2.4 Elements of reinforcement learning

In RL the learner and decision maker is called the *agent*. The thing it interacts with, comprising everything outside the agent, is called the *environment*. It is described by some set of possible *states* ($S$). The agent can perform any of a set of possible *actions* ($A$). Each state $s \in S$ is a vector of attributes (*state variables*) that describes the current configuration of the system [NFK06]. More specifically, a state variable is the minimally dimensioned function of history that is necessary and sufficient to compute the "state of knowledge" [Pow07].

Except for the *agent* and for the *environment* one can identify four other main sub-elements:

- the *policy* : defines the learning agent's way of behaving at a given time (a mapping from states to actions).
- the *reward signal* : defines the goal in a RL problem. On each time step, the environment sends to the RL agent a single number called the *reward*. The agent's sole objective is to maximize the total reward it receives over the run.
- the *value function* : specifies the total amount of reward an agent can expect to accumulate over future, starting from a specific state.
- the *model* : allows inferences to be made about how the environment will behave.

As shown in figure 2.5 each time an agent performs an action $a_{\mathrm{t}}$ in a state $s_{\mathrm{t}}$, it receives a real-valued reward $t_{\mathrm{t}}$ that indicates the immediate value of this state-action transition. This produces a sequence of states $s_{\mathrm{i}}$, actions $a_{\mathrm{i}}$, and immediate rewards $r_{\mathrm{i}}$ as shown in the figure. The agent's task is to learn a control policy, $\{\pi : S \longrightarrow A\}$, that maximizes the expected

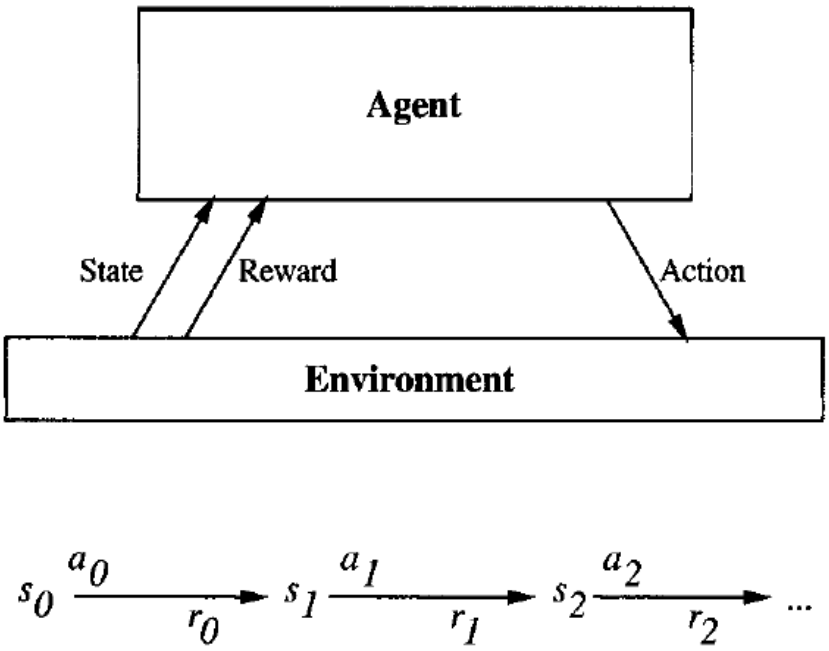


Figure 2.5: RL working schema.

sum of these rewards, with future rewards discounted exponentially by their delay [Mit97].

According to how just said, the most important feature distinguishing RL from other types of learning is that it uses training information that *evaluates* the actions taken rather than *instructs* by giving correct actions. It is clearly an *on-line learning* technique (data arrives sequentially and the model is continually updated as new data becomes available).

## 2.5 Finite Markov Decision Processes

The environment in RL problems can be well described as *Markov decision processes* (MDPs). MDPs are a classical formalization of sequential decision making under uncertainty, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those

future rewards. From a similar perspective we can define MDPs as a general mathematical formalism for representing shortest path problems in stochastic environments.

A MDP process relies on the notation of state at time $t$, $S_t$, the action selected on time $t$, $A_\mathrm{t}$, the corresponding reward $R_\mathrm{t}$. The value of an arbitrary action $a$, denoted $q_*(a)$ is the expected reward given that $a$ is selected : $q_*(a) = \mathbb{E}[R_\mathrm{t}|A_\mathrm{t} = a]$. We denote the estimated value of an action $a$ at time step $t$ as $Q_\mathrm{t}(a)$.

Solving an MDP consists of controlling the agent in order to reach an optimal behaviour, i.e. to maximize its overall revenue. Because action effects are stochastic and, thus, can result in different possible states at the next stage of the decision process, the optimal control strategy cannot necessarily be represented as a single sequence of actions. Due to the uncertainty in actions' results, applying a given policy can result in different sequences of states/actions [SB10].

### 2.5.1 The Agent-Environment Interface

In MDPs the agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, 4, ....$ At each time step $t$, the agent receives some representation of the environment's *state*, $S_\mathrm{t} \in S$, and on that basis selects an *action*, $A_\mathrm{t} \in A(s)$. One time step later, in part as a consequence of its action, the agent receives a numerical *reward*, $R_{\mathrm{t}+1} \in R \subset \mathbb{R}$, and finds itself in a new state, $S_{\mathrm{t}+1}$. The MDP and agent together thereby give rise to a sequence of *trajectory* that begins like this :

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, ... \tag{2.6}$$

In a *finite* MDP, the set of states, actions and rewards $(S, A, R)$ all have a finite number of elements. In this case, the random variables $R_\mathrm{t}$ and $S_\mathrm{t}$ have well defined discrete probability distributions dependent only on the preceding state and actions. That is, for particular values of these random variables, $s' \in S$ and $r \in R$, there is a probability of those values occurring at time $t$, given particular values of the preceding state and action :

$$p(s', r|s, a) = Pr\{S_\mathrm{t} = s', R_\mathrm{t} = r|S_{\mathrm{t}\text{-}1} = s, A_{\mathrm{t}\text{-}1} = a\} \tag{2.7}$$

for all $s'$, $s \in S$, $r \in R$, and $a \in A(s)$. The function $p : S$ x $R$ x $S$ x $A \longrightarrow [0, 1]$ is an ordinary deterministic function of four arguments [SB18].

The '|' in the middle of it just reminds us that $p$ specifies a probability distribution for each choice of $s$ and $a$ :

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) = 1, \forall s \in S, a \in A(s). \tag{2.8}$$

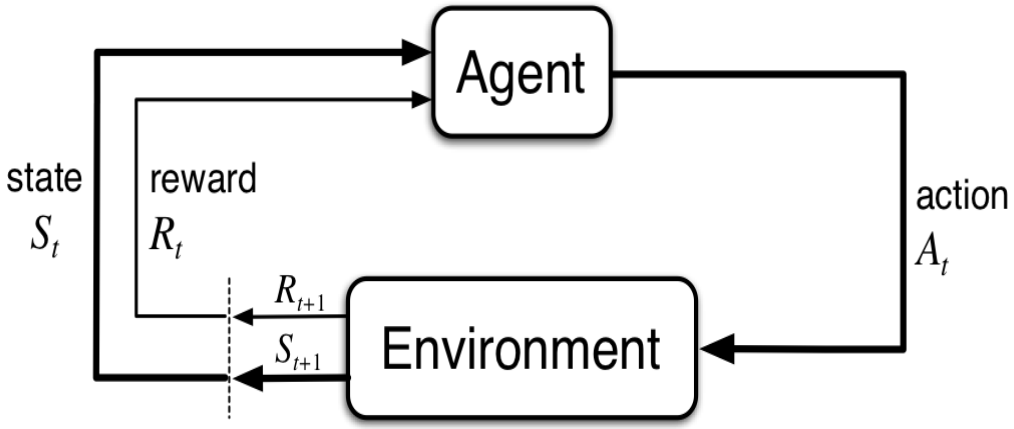


Figure 2.6: The agent-environment interaction in a MDP.

This type of decision process is called a Markov process because it obeys the *Markov property*. In systems that obey this property, the future of the system is independent of its past, given the current state. In other words, if we know the current state, knowing additional information about previous states and reinforcements does not improve our ability to predict future states or reinforcements [Mai09].

### 2.5.2 Goals, Rewards, Returns and Episodes

As previous said, in RL, the goal of the agent is to maximize the expected value of the cumulative sum of a received scalar signal called *reward*. It is critical that the rewards we set up truly indicate what we want accomplished. If we reward the achievement of subgoals , then the agent might find a way to achieve them without achieving the real goal. Reward signal is our way of communicating the agent *what* we want to achieve, not *how* it achieved.

Let's suppose that the sequence of rewards received after time step $t$ is denoted $R_{t+1}$, $R_{t+2}$, $R_{t+3}$, $R_{t+4}$, .... We expect the agent maximizes the *expected return*, where the return, denoted $G_t$, is defined as some specific function of the reward sequence. In the simplest case the return is the sum of the rewards :

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + ... + R_T, \tag{2.9}$$

where $T$ is a final time step. This approach make sense in applications in which there is a natural notion of final time step, that is, when the agent-environment interaction breaks naturally into subsequences, which we call *episodes* $(E)$ [SB18]. To be more specific, in this case, the breaking into subsequences is also visible in episodes and we call them *epochs* $(e)$.

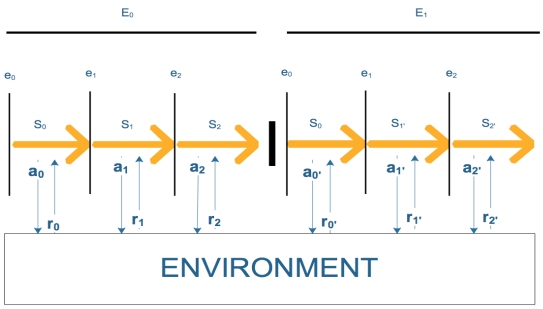


Figure 2.7: Schema of an agent-environment interaction broken into two episodes $(E)$, each one broken into three epochs $(e)$.

Each episode ends in a special state called *terminal state*, followed by a reset to a starting state or to a sample from a standard distribution of a standard of starting states.

In many cases the agent environment interaction does not break naturally into identifiable episodes, but goes on continually without limit; we call this *continuing tasks*. The return formulation is problematic for continuing tasks because the final time step would be $T = \infty$, and the return, which is what we are trying to maximize, could itself be infinite [SB18].

An additional concept that we need is that of *discounting*. According to approach previously explained, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it choses $A_t$ to maximize the expected *discounted return* :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.10}$$

where $\gamma$ is a parameter, $0 \leqslant \gamma \leqslant 1$, called the *discount rate*. This value determines the present value of future rewards. If $\gamma < 1$, the infinite sum in (1.5) has a finite value as long as the reward sequence $R_k$ is bounded. If $\gamma = 0$, the agent is "myopic" in being concerned only with maximizing immediate rewards: its objective in this case is to learn how to choose $A_t$ so as to maximize only $R_{t+1}$. As $\gamma$ approaches 1, the return objective takes future rewards into account more strongly [SB18].

### 2.5.3 Policies and Value Functions

Almost all reinforcement learning algorithms involve estimating *value functions* that estimate *how good* it is for an agent to be in a given state (or how good it is to perform a given action in a given state). More formally we can say that a *policy* is a mapping from states to probabilities of selecting each possible action. If the agent is following policy $\pi$ at time $t$, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$.

The *value* of a state $s$ under a policy $\pi$, denoted $v_\pi(s)$, is the expected return when starting in $s$ and following $\pi$ thereafter. For MDPs, we can define $v_\pi$ formally by

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t\,:\,k\,:\,1}|S_t = s], \forall s \in S \tag{2.11}$$

where $\mathbb{E}[\cdot]$ denotes the expected value of a random variable given that the agent follows policy $\pi$, and $t$ is any time step. We call the function $v_\pi$ the *state-value function* for policy $\pi$. Similarly, we define the value of taking action $a$ in state $s$ under a policy $\pi$, denoted $q_\pi(s, a)$, as the expected return starting from $s$, taking the action $a$, and thereafter following policy $\pi$ :

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t\,+\,k\,+\,1}|S_t = s, A_t = a] \tag{2.12}$$

We call $q_\pi$ the *action-value function* for policy $\pi$. The value functions $v_\pi$ and $q_\pi$ can be estimated from experience.

A fundamental property of value functions used throughout RL is that they satisfy recursive relationships. For any policy $\pi$ and any state $s$, the following consistency condition holds between the value of $s$ and the value of its possible successor states:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{\mathrm{t+1}} + \gamma G_{\mathrm{t+1}}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{\mathrm{t+1}}|S_{\mathrm{t+1}} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S \end{aligned} \tag{2.13}$$

where it is implicit that the actions, $a$, are taken from the set $A(s)$, that the next states, $s'$, are taken from the set $S$, and that the rewards, $r$, are taken from the set $R$.

Equation (1.8) is the *Bellman equation* for $v_\pi$. It expresses a relationship between the value of a state and the values of its successors states [SB18].

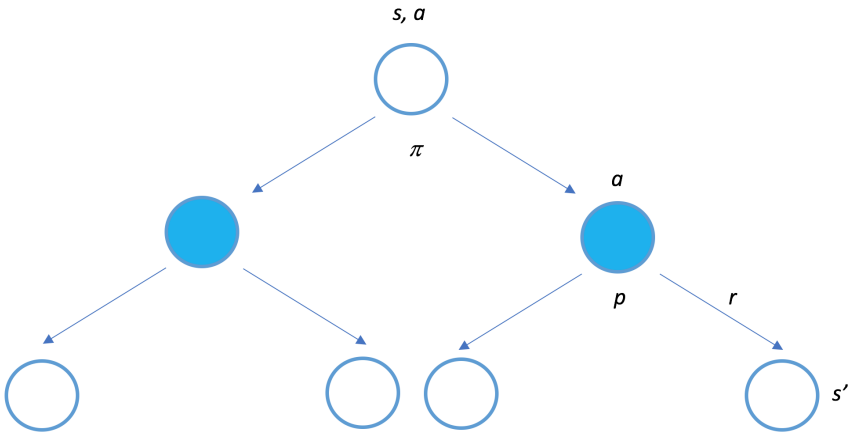


Figure 2.8: Think of looking ahead from a state to its possible successor states. Each open circle represents a state and each solid circle represents a state-action pair. Starting from state $s$, the root node at the top, the agent could take any of some set of actions - two are shown in the diagram - based on its policy $\pi$. From each of these, the environment could respond with one of several next states, $s'$, along with a reward, $r$, depending on its dynamics given by the function $p$. The Bellman equation (1.8) averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the discounted value of the expected next state, plus the reward expected along the way.

A policy $\pi$ is defined to be better than or equal to a policy $\pi'$ if its expected return is greater than or equal to that of $\pi'$ for all states. There is always a at least one policy that is better than or equal to all other policies. This is the so called *optimal policy*. We have to underline that it is possible that exist more than one optimal policy. We denote all of them by $\pi_*$. They share the same state-value function, called the *optimal state-value function*, denoted $v_*$, and defined as

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in S \tag{2.14}$$

Optimal policies also share the same *optimal action-value function*, denoted $q_*$ and defined as

$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall s \in S, a \in A(s) \tag{2.15}$$

For the state-action pair $(s, a)$, this function gives the expected return taking action $a$ in state $s$ and thereafter following an optimal policy. Thus, we can write $q_*$ in terms of $v_*$ as follows [SB18] :

$$q_*(s, a) = \mathbb{E}[R_{t\ +\ 1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \tag{2.16}$$

# Bibliography


[AS08] Ryan P. Adams and Oliver Stegle. Gaussian process product models for nonparametric nonstationarity. In *ICML*, 2008.

[HL01] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, New York, NY, USA, seventh edition, 2001.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.

[Kon09] Takis Konstantopoulos. Markov chains and random walks. 2009.

[LM16] Ke Li and Jitendra Malik. Learning to optimize. *CoRR*, abs/1606.01885, 2016.

[Mai09] Tiago V. Maia. Reinforcement learning, conditioning, and the brain: Successes and challenges. *Cognitive, Affective & Behavioral Neuroscience*, pages 343–364, Dec 2009.

[Mit97] Tom M. Mitchell. *Machine Learning*. 1997.

[NFK06] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 673–680, New York, NY, USA, 2006. ACM.

[Pow07] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007.

[Put94] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[SB10] Olivier Sigaud and Olivier Buffet. *Markov Decision Processes in Artificial Intelligence*. Wiley-IEEE Press, 2010.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning : An Introduction*. 2018.

[SSW+16] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[WvO12] M. Wiering and M. van Otterlo. *Reinforcement Learning: State-of-the-Art*. Adaptation, Learning, and Optimization. Springer Berlin Heidelberg, 2012.