1. Artificial intelligence and machine learning landscape

ARTIFICIAL INTELLIGENCE: BASIC CONCEPTS

Machine Intelligence

Artificial intelligence (AI) is the central focus of this book – it is therefore only fitting that we begin our deliberations by reflecting on the term itself.

The first word denotes something fairly easy to define: *artificial* simply means *man-made*. Naturally, any attempt to delve on specifics such as e.g. *form*, *physical location* or *architecture* of such a creation would significantly complicate things. As will be shown in the subsequent chapters of this book, at present, intelligent systems are basically computer programmes running on devices utilising silicon chips. They can operate 'autonomously', take advantage of remote intelligent services (e.g. in the Cloud Computing model) or engage with other objects to establish networks. These seemingly trivial observations are in fact anything but. These days, highly advanced operations can already be performed by quantum computers (inherently different from contemporary integrated circuitry), data can be stored in DNA codes (rather than on digital memory chips) and calculations can be performed with the use of protein structures (and one does not mean animal brains). As we can see, there is currently more than one way to potentially approach the issue of artificiality – anyone interested in the topic could do far worse than to read through Bostrom's compelling analysis of the same (2014).

A concept far more difficult to put a finger on is *intelligence*. There have been many definitions over the years, mostly in the field of psychology. It has been a long-standing dream to create structures capable of assisting or replacing people in solving various day-to-day problems. One of humans' 'competitive advantages' over animals is the ability to use tools – objects enhancing people's innate potential, e.g. strength or speed. In time, such tools evolved into machines – still, their contributions continued to be limited to physical activities. The development of higher cognitive functions, in particular the ability to formulate complex goals, adopt and implement strategies and plan actions necessary to that end, inspired people to search for methods of enhanc-

ing their abilities also in those specific areas (i.e. no longer exclusively physical but also mental). A sort of a 'prototype' in this context came in the form of mathematical theorems and algorithms which (due to their abstract nature) helped in categorising and correlating certain seemingly diverse problems and subsequently fairly easily solving them (e.g. the laws of geometry applied in the construction of the Tunnel of Eupalinos in ancient Greece). The next natural step was to 'implement' those originally abstract algorithms in physical devices: initially simple (e.g. the abacus) and then increasingly complex (e.g. Babbage's difference engine) (cf. Davis 1949).

As algorithms continued to be developed, so did the range of problems to which they could be applied, which on the one hand created a demand for increasingly advanced computing machines, and on the other generated entirely new problems. There were certain rather narrow areas (e.g. arithmetic) where machines fairly quickly proved themselves to be more capable than humans. However, as computers began to become more popular, in particular more available to a wider group of programmers, the spectrum of problems that could possibly be solved by man-made machines grew exponentially – which soon inspired people to reflect on the machines' intelligence.

Before committing to a definition of *intelligence*, let us first take a somewhat closer look at the problems one may reasonably expect intelligent machines to solve. To that end, we will use the slightly adapted classification first proposed by Russell et al. in their fundamental textbook on AI (2010).

Let us begin by *characterising the environment* wherein our machine might operate. As the given task is being performed, the same can remain unchanged (*static* as e.g. in the game of chess) or evolve – i.e. be *dynamic*.[1] Moreover, the next consecutive state of the environment may depend solely on the actions of the machine (in which case the environment is *deterministic*) or be influenced by factors independent of the agent (a *stochastic* environment). The environment can also be *discrete* (described by variables with a finite set of possible values) or *continuous*. The latter distinction is particularly important from the computing perspective: in a world of infinite choices, the system must have the capacity for generalisation. Finally, our machine (often described as the agent) may act within the given environment either *independently* or be accompanied by other 'beings' (be it machines, animals or humans) – in which case we can talk of a *multi-agent* environment.

The next distinction is related to *knowledge about the environment*. The agent may *know* the rules and laws applicable in the given context (e.g. in board games) or *not know* them (as is often the case in real life). Another aspect of knowledge about the environment is access to information about its state: the agent may either have *full knowledge* of the environment's state, only *partial knowledge* thereof or *no knowledge whatsoever* (be blind).

The final classification relates to the character of *tasks* that our agent might aim to perform. They may be either *episodic* (have a clearly defined beginning and end, like in the case of most games) or *continuous* (have no natural end point).

As follows from the above, the complexity spectrum of potential problems is very wide: from the performance of episodic tasks in single-agent, deterministic, fully observable worlds with readily available rulesets, to continuous tasks performed in stochastic environments without any knowledge of the applicable rules or ability to perceive anything.

The above classification is a good starting point when attempting to define intelligence. With full appreciation for the complexity of this issue, for the purposes of the deliberations to follow, intelligence will be defined as the *ability to effectively act under new circumstances*. A closer analysis of the proposed definition will also facilitate a first look at the concept of autonomous, self-learning systems whose business potential will be discussed in the subsequent parts of this book.

Let us begin with the word *effectively*. It assumes the existence of a certain goal function: in most cases, intelligent systems are expected to perform certain missions whose success or failure is possible to define. Measures of success are usually an integral part of the task and help the machine to monitor, on an ongoing basis, the progress of the learning process as well as the status of mission completion.

Formulating a viable goal function is a considerable challenge in itself. One of the most popular methods (discussed below in greater detail) is the trial and error approach: the agent decides to perform a given action and receives feedback from its environment containing data on the new situation and applicable reward or punishment. The reward, often described as *enhancement* – a term derived from psychology – is one of the primary drivers of the learning process. On the one hand, the imperfection of many teaching methods manifests itself in particular in the necessity of engaging in a large number of interactions between the system and its environment – i.e. many *trials* and *errors*. This can prove not only costly but also risky.

For this reason, algorithms are often trained in virtual simulators. The creators of such solutions must not only ensure they carefully recreate the given environment but also construct a precise reward system to promote various behaviours, usually by relating the state of the environment resulting from the agent's activity to mission-specific conditions. In practical terms, this can pose a considerable challenge: the system will operate on the premise of being motivated while learning, consequently any errors in the definition of the reward function will have far-reaching consequences.

A separate, more commonly discussed question is *who* should determine the goal function. If we can learn anything from human history, the story of King

Midas and the like, fulfilling one's dreams can actually have rather tragic consequences. Therefore, maybe it would be in people's best interest if AI were to decide what is and what is not good for them? In other words, at the operational level, independently map out its own way towards the target? This question has been the focus of research on so-called *inverse reinforcement learning* (RL): the purpose being to teach a system to identify goals, values and rewards applicable to a given object by observing its behaviour (cf. Heidecke 2019). Such an approach may not only impact the ways in which AI is designed but also redefine the very concept of intelligence. It may soon be the case that we will expect such systems to effectively solve problems posed to them as well as accurately identify even our unexpressed needs. Should this happen, *the future won't be what it used to be* (to paraphrase Paul Valery).

For now, however, let us go back to our definition of *intelligence*. We described it as the *ability to effectively act under new circumstances*. As we have now briefly discussed *effectiveness*, it is time to look into *action*.

Action entails not only the ability to make decisions but also access to so-called actuators: physical devices or communication interfaces facilitating interactions with the environment. In the next section focusing on the basics of RL, the concept of the so-called agent will be introduced with his role limited solely to control. Our definition of intelligence expands the concept of action by including the ability to affect the environment.

Nonetheless, interaction is not the only type of *action* necessary for the successful performance of a mission. It is equally, or possibly even more important, to be able to effectively extract data, information and knowledge from the environment, and then use it for the improvement of one's efficiency. The ability to pass on experience and knowledge, not just from generation to generation (e.g. through the process of evolution) but also within the same age group, in a scope far broader than mere genetic changes, has long constituted another element of human 'competitive advantage' over other living creatures. This allows our societies to learn at an accelerated pace, which, combined with the capacity for memory (both short term and historical) has also been the basis of our technological advancement. Given the above, another desirable skill of an intelligent system would be to have the ability to gather information and knowledge and use experiences, not only originating from oneself but also from others.

The final aspect of activity pivotal to our understanding of intelligence is the *capacity for self-learning and self-improvement*. Intelligent machines should not only effectively use their externally provided 'brains' but also relatively quickly gather new experiences and be able to independently learn from them. This is the central focus of machine learning (whose key premise will be discussed below) as well as studies conducted in the areas of so-called *meta-learning* (teaching machines to recognise the optimum strategies of

self-learning) and *self-supervised learning* (teaching machines to supervise their own learning processes). These are but a few fascinating areas of research that may strongly influence the future of the world we live in.

The final element of our definition refers to *new circumstances*. In this context, '*new*' should be understood as synonymous to the notions of complexity and uncertainty, whereas '*circumstances*' refer to the given context (environment, world) wherein the intelligent system operates. The phrase is used here to facilitate reference to the aforementioned classification of problems with which machines may have to contend.

As already stated, the spectrum of such problems is vast: from things well known, observable and deterministic, to everything that is incomprehensible, non-observable and changeable. In our definition, *new circumstances* refer to the latter extreme, bearing in mind that it may be difficult to naturally distinguish between intelligence and automation stemming from the implementation of simple algorithms. An intelligent agent ought to effectively perform its tasks under circumstances that are inherently uncertain (stochastic environments where one cannot be certain about the viability of one's actions), prone to change (with the rules of the game changing dynamically) and only partially observable ('with limited visibility'). In this context, the ability to operate in *continuous* environments seems relatively trivial (after all, this is the natural human environment: the images we perceive are not composed of large 'pixels', our movement is fluid, etc.), but in practical applications this requirement (decision making when faced with information overload) turns out to pose a very significant challenge. Our definition of *machine intelligence* is, in a way, not particularly far removed from our intuition about human intelligence. The people we consider to be intelligent are often those capable of performing effectively when faced with completely new situations – the condition of newness is therefore crucial to the distinction between *intelligence* and *teachability*.

Many of the technologies described in Chapter 2 fail to meet the requirements of such a strong definition of intelligence – nonetheless, their development is a necessary step if the same is ever to be achieved. An intelligent system should be able to receive information from its surroundings – it will therefore have to heavily rely on various sensor technologies. It must also know how to interpret the same – recognise objects and situations, identify patterns – which will require the use of algorithms facilitating image recognition or natural language processing. A machine will have to develop strategies and plans, learn, memorise and eventually also interact with its environment – not unlike us, humans. It is therefore hardly surprising that the trend of developing systems equipped with such higher abilities is referred to as *cognitive computing*.

Machine Learning and Programming: How Then Does One Build an Intelligent Machine?

Let us imagine a manager responsible for the performance of a small business unit. How would he ensure the effective operation of his department, in particular timely performance of the designated goals?

Our hypothetical manager has a few available options. The first, somewhat crude but sometimes the only viable solution, is to do all the important work himself. This minimises the risk of failure (a highly experienced expert can often complete a given task much faster than a relative novice) but is unsustainable in a long-term perspective. The subordinates will quickly get accustomed to having his support, which will prevent them from developing their own competences. Soon, exertion will have the better of our hero and he will himself become the proverbial bottleneck. Clearly, this is not the best solution.

Another option is to issue precise commands and enforce their subsequent execution. This is already a step forward: instead of doing everything himself, the manager coordinates the work of others. The risk here? Employees will quickly get accustomed to working only under the influence of external stimuli, they will not have an internal locus of control and the entire structure will collapse at the first sign of leadership deficiency.

A somewhat better idea is to manage responsibilities. Employees are no longer assigned specific tasks but rather areas of responsibility. We formulate goals and leave it to the employees to make them happen.

Are there even higher conceivable levels of self-reliance? In a way, there are: one could teach employees to formulate performance strategies and learn independently – if their competences are lacking, even to formulate their own goals consistent with the organisation's vision. This way, they could set their own targets and motivate others in the process. If all goes well, they will be able to independently solve new problems, create new rules of procedure and constantly improve their competences wherever necessary.

And how does this relate to *building intelligent machines*? Let us now imagine an engineer who aims to design an autonomous helicopter. Navigation would be a considerable challenge here: it would require adjusting the speed of the main (horizontal) rotor as well as the vertical rotor at the back. And all that not only while maintaining a certain fixed position but also while travelling in a certain direction in fast-changing (e.g. gusts of wind) conditions. So how should he approach building such a machine?

Our engineer has to choose between two options. The simplest he/she can think of would entail developing computer software that would instruct the machine telling it how to behave under given circumstances, naturally based on experiences of professional pilots. At the end of the day, this could be boiled down to rudimentary *what-if* rules, and in more refined environments to

dedicated functions capable of approximating such algorithms. What are the risks? It could soon turn out that the helicopter finds itself in a situation not supported by the algorithm – i.e. one in which it will not receive any instructions. In real-world applications this would be a sure recipe for disaster. This approach to machine building is referred to as *programming*.

So, what else can one do? Rather than programme the machine to perform tasks, he could ensure that it can actually learn how to perform them. In other words, the machine would need to be provided not so much with information on how to behave under given circumstances but rather the capacity to learn the same. This is the approach commonly referred to as *machine learning*.

To look back to our analogy of a manager responsible for a team: issuing commands can be compared to programming (the employees perform clearly defined tasks), with encouragement of independent work and self-improvement, to machine learning. An intuitive understanding of this distinction will prove very helpful when navigating the complex world of data science, machine learning and AI.

We could also use a somewhat different example to illustrate yet another aspect of the need to *teach machines how to self-learn*. Let us imagine a machine performing simple arithmetic calculations, e.g. a common calculator. In order to build it, a mathematician had to create an algorithm: procedures to follow for three input objects (for the sake of simplicity, let us assume two digits and an operator) and one output object (a number). Each digit can take ten different values, there are four possible operators (+, –, ×, /), and 82 possible results (0 to 81). A programmer then implemented this algorithm using a particular programming language, and an engineer uploaded it to the final device which is further composed of such elements as a keypad, processor or screen. Nonetheless, the key component remains the algorithm: the procedure of processing three 'small' (in terms of the number of dimensions and possible values) objects in order to produce the desired output. It is so simple in fact that mathematicians have long been able to utilise it even without machines.

Now, let us consider the process of recognising objects in photographs – for the sake of simplicity only two: a dog and a cat. In this case, there are considerably more input 'objects' to consider: so-called pixels, i.e. the most elementary components of any digital image. Each pixel is described by three numbers (separately for red, yellow and blue) with the possible values ranging from 0 (perfect black) to 225 (perfect white). Should we assume an average resolution of 800 × 600 pixels, the image would therefore be characterised by 1,440,000 numbers, each with the value range from 0 to 255 – that is considerably more than three objects with a dozen or so possible values… Consequently, we end up with an algorithm that has to process over 1.4 million 'objects' to provide an output of merely two digits, each with the range of possible values from 0 to 1: the first number to indicate the probability that the image is a cat, the

other the relative probability of the image depicting a dog. As we can see, an algorithm tasked with distinguishing between a cat and a dog will have to handle drastically more input information than a simple calculator. One could immediately ask: would a human be able to come up with a way of processing such amounts of information and distinguishing those two important animals? Obviously, our own brain is more than capable of doing that, but what we really want to know is if a programmer could implement the skill in software and an engineer get it to work on a dedicated device. It does seem that creating an algorithm of this type would be beyond human ability – but if that is the case, why not have a machine do it? This is a task exactly suited for machine learning: typically, the process yields its effects in the form of 'brains', nowadays increasingly taking the form of so-called neural networks capable of not only recognising objects in images, but also understanding spoken language and interpreting human emotions. A machine begins its 'education' with a completely fresh 'brain' and then proceeds to improve it in a variety of 'ways'. The entire process is typically described as 'machine learning' and this particular form of teaching as the 'learning algorithm'. Both will be described in greater detail in the next section.

Methods of Machine Learning

The most popular criterion used for classifying machine learning methods is their *level of autonomy*: from supervised learning, through semi-supervised, unsupervised and RL, all the way to self-supervised and meta-learning. Each of the above will be briefly characterised below, including their most important algorithms and common applications.

Let us start with *supervised learning*. In this case *supervision* means the presence of a *teacher* who supervises the learning process, usually by evaluating the correctness of the answers given. The basis for learning is provided by a so-called *teaching set* containing data characterising the problem as well as the correct answers. The task of the *teaching algorithm* is to guide the entire process in such a way that the structure trained (e.g. randomly initiated neural network or another set of rules, like a decision tree) could quickly evolve out of the random answer stage (beginning of the learning process) and up to a satisfactory level of accuracy (end of the learning process). This is not unlike teaching skills to humans: a beginner will have problems with e.g. recognising constellations in the night sky, but once trained, they will do it practically flawlessly.

As mentioned above, the starting point in this case is a *man-made dataset*. It usually comprises *objects*, their *properties* (explanatory variables, predictors) and *labels* (dependent variables). The *properties* correspond to questions and the *labels* to answers. For instance, if we want to teach our system to recognise

dog breeds from photographs, the teaching set should contain a large number (e.g. several thousand) photographs (predictors) with descriptions of the respective breeds (labels). In the course of the teaching process, we 'show' the system one photograph at a time, asking it to identify the breed. After each answer feedback is provided to indicate how correct the given answer is, which the teaching algorithm then uses to 'streamline' the trained structure. The process is then repeated until the incidence of errors is satisfyingly low.

The key element of the process is dividing the set of the correctly described objects (in our case photographs) into those that serve as the teaching base (training set) and those that will be used to test the system (test set). There is no point in testing the system's ability to provide answers that were previously taught to it. Again, a parallel to human learning could be drawn: using problems solved during lectures as test questions yields only an illusion of competence.

The most common uses of supervised learning include *discrete variable prediction* (so-called *classification*) and *continuous variable prediction* (so-called *regression*). Classification may be binary (when dealing with only two properties, e.g. SPAM or NOT-SPAM in classifying email messages) or multi-label (e.g. the dog breed classification; Figure 1.1). In regression, the system is taught how to diagnose a variable whose potential values may even be infinite (Figure 1.2). Good examples here could include apartment prices (e.g. with location and floor area serving as dependent variables) or stock price forecasts.

In a sense, the objective of variable learning is to achieve better *generalisation*. Generalisation is the foundation of abstract thought, it allows us to simplify our surrounding reality by reducing it to a limited subset of concepts. Every time I look at my reflection in the mirror, my brain receives an entirely different image: facial expressions will change, as will angles of view, lighting, elements in the background, etc. And yet, I invariably identify this set of visual stimuli as my face. Not someone else's but mine, not a leg but a face. Hence, my brain generalises: various images are equated to a single object.

A similar ordinal function is served by definitions and, more generally, many linguistic parts of speech. It is a matter of certain convention that an object slightly smaller than a person, with a small horizontal area and several (but at least three) elements attached below, will be called a table. There are literally millions of the concept's possible 'iterations', and yet we will still recognise each of them as a table. Such generalisations can certainly make one's life easier, so their utility is considerable. As discussed below, a similar ability should be available to intelligent systems – without it, each of such states would be treated differently, placed in a different reality. *One of the purposes of supervised learning is to develop the capacity for generalisation*.

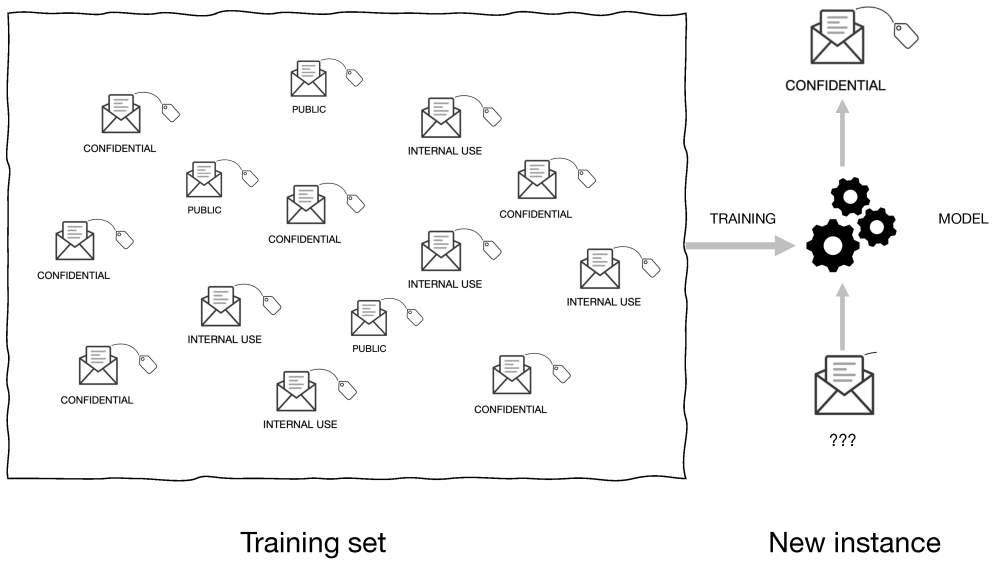

Figure 1.1 Illustration of the multi-label principle

Note: The task of the model is to predict a feature with a finite number of possible values (labels). In the presented example, the feature relates to types of email messages: they can be classified as confidential, internal or public. The model trained with the use of training data is later used to prognose the properties of other instances (in our case email messages).
Source: Own elaboration.

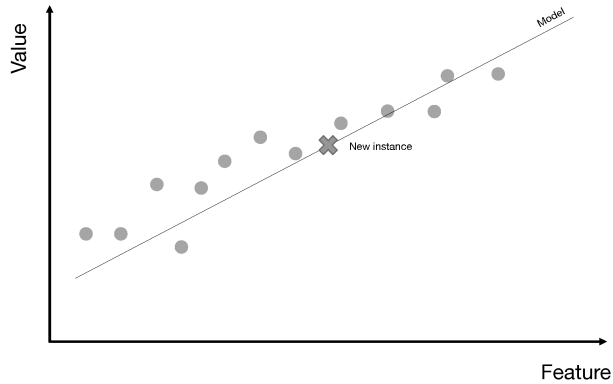

Figure 1.2 Illustration of a regression model

Note: As with classification, the purpose of the model is to prognose a certain feature, in this case, however, it is the explanatory variable that can have an infinite number of values. For instance, when projecting the price of an apartment depending on its floor area.
Source: Own elaboration.

The most important algorithms (or 'teaching methods') used in supervised learning include linear regression, logical regression, support vector machines, decision trees, random forests and neural networks.

In summary: the key element in supervised learning is the man-made dataset containing pairs of questions and answers. It is used in the teaching process as a source of feedback on the system's progress. The dataset is divided into training and testing subsets, the latter being used to evaluate the system's quality.

It is often the case that no set of questions and answers is readily available, or that rather than prediction, our goals are focused on e.g. pattern recognition, limiting the number of variables, or simply visualisation. In such cases, it is common to turn to the methods of unsupervised learning, i.e. learning unlabelled data.

One of the main roles of unsupervised learning relates to *data compression*, a skill that is, in a way, complementary to generalisation. As we contemplate reality, our brain is always searching for certain patterns, underlying principles of the world we live in. Such patterns emerging as 'formulas' allow us to 'compress' reality. For instance, rather than build a tabular listing of the possible terminal speeds of a body falling from different heights, we use the formula for speed in uniformly accelerated motion and save ourselves the need to do literally millions of calculations (individually for each of the possible

elevations) and just do one – using the discovered formula. Once identified, a pattern compresses information. And in order to find it, we can turn to the methods of unsupervised learning.

A typical application of unsupervised learning entails *clustering* (Figure 1.3). For instance, a telecom with access to information concerning details of its customers' phone calls may be interested in identifying and clustering the same. The grouping could be done based on behaviour with the members of a single group sharing a certain common feature and clearly standing out from other groups in some respect. What is important, the company does not know the character of the respective groups – the customers are not labelled, there is no training dataset to use – it is the machine learning model that is expected to identify the same.

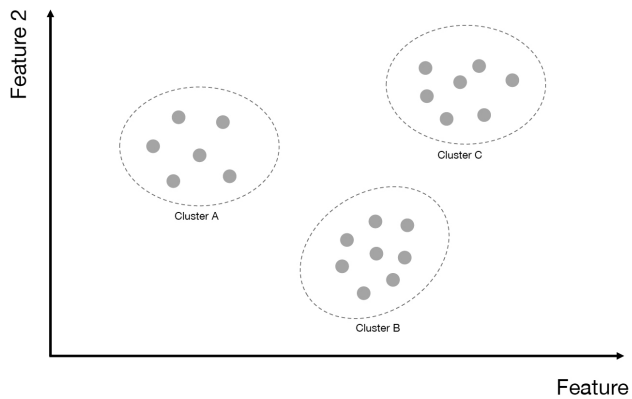

Figure 1.3 Illustration of a clustering model

Note: Here, the model's task is to identify a pattern: divide objects into groups of objects that are similar to each other and different from other groups. Unlike in the case of regression, the Y-axis does not correspond to a dependant variable but another feature of the object (naturally, there may easily be more than two features concerned). The model identifies groups (clusters) and labels them with its own, abstract designations. It is the job of the human to interpret the significance of the groups and rename them where appropriate.
Source: Own elaboration.

What is the point of clustering customers? The first obvious benefit stems from *data compression*: the human brain is equipped to better understand a situation with fewer qualitatively different elements (e.g. five to ten) rather than, say, millions of individual customers. Another benefit stems from *personalisation*

of offerings: it is easier to develop several distinct packages of diverse product offers than millions of almost identical ones.[2]

'Compression' can be applied not only to objects (e.g. customers) but also the number of features that describe them. Operators of popular social media sites can easily produce hundreds of different statistics about their users – the difficulty lies in extracting valuable information from such raw data. The human brain has its inherent limitations: we find it relatively easy to analyse two variables (by plotting two objects in a two-dimensional graph), three make the process somewhat more difficult (a three-dimensional graph or two-dimensional with the addition of e.g. another colour), and with every additional dimension, our capacity to understand reality suffers significantly. And what if the dimensions come in their hundreds? It turns out that supervised learning, in particular the methods of so-called dimensionality reduction, can significantly reduce the number of variables. When combined with constantly improving methods of visualisation, such solutions assist people in more efficient interpretation of facts.

The primary difference between supervised and unsupervised learning stems from the aforementioned lack of the teaching set. However, this does not mean that the algorithms have to learn without data: they are still provided but without preassigned labels (dependent variables) – it is up to the model to interpret them. The most popular algorithms used in unsupervised learning (i.e. methods of developing models) include the K-means method, Hierarchical Clustering, DBSCAN and Principal Component Analysis.

An 'in-between' method combining elements of both of the above is called *semi-supervised learning*. The method is useful in cases where a teaching dataset (i.e. question and answer pairs) is available but relatively limited – and our aim is to generalise that knowledge to many more cases. For instance, social media websites may be interested in implementing a system capable of identifying people in photographs uploaded to the system. How could this be achieved? In the first stage, we use supervised learning methods to group people in millions of untagged photographs. The system learns to identify a given person in many photographs (in clustering, a person is the equivalent of a *group*), still without knowing their name (the group code is always abstract, generated by the model and otherwise meaningless). Once this stage is complete, the system can conclude that e.g. person A115 appears in photographs 10, 14 and 167. In the second stage, the method of supervised learning is employed: based on the existing training set (in this case photographs of people associated with specific personal information), the model obtained in the previous step is taught how to associate abstract codes with specific people. Consequently, the combination of the two methods (supervised and unsupervised learning) yields a recognition model trained with a relatively small training set.

Having discussed the methods of supervised, semi-supervised and unsupervised learning, it is now time to move on to somewhat more 'autonomous' methods. But what exactly does autonomy entail here? Was the system not required to independently discover patterns and rules already in the unsupervised approach?

In machine learning, intelligent systems learn from data. In the three cases described above, the data had to be prepared and provided by humans. Even though in unsupervised learning only the features of objects were provided (no labels, i.e. no explanatory variables), they were nonetheless fed to the system by a human operator – like food to a small child. Meanwhile, autonomous systems are expected to act, in particular learn, without external human support. The question is how to design a 'didactical process' where the system is capable of independently providing feedback.

The first such method is referred to as *self-supervised learning* and is an interesting hybrid of supervised and unsupervised approaches. Similar to supervised learning, the system learns to recognise objects based on labelled datasets (i.e. *question <> answer* pairs). The key difference is that in this case, the labels are assigned by the system itself, without outside support. The system supervises itself. There are many interesting applications for this particular approach. Image recognition systems learn to independently classify objects based on contextual and spatial data (cf. Doersch et al. 2016), and robots train their visual systems to recognise objects based on data from other perception systems (Nava et al. 2019). The method also allows intelligent systems to explore and learn about the world on the strength of actual curiosity rather than enhancement stimuli (punishment and reward) (Pathak et al. 2017).

Another interesting approach, referred to as *reinforcement learning*, was inspired by advances made in psychology. As humans, we are taught by parents and teachers from an early age, but also explore our surroundings by ourselves and learn by trial and error. In order to achieve a certain goal, we perform various possible actions and observe their results. In time, we are either able to develop an efficient sequence of steps or, which ultimately may prove more effective, learn about the operating principles of a given device or rules applicable to a given environment. Autonomous systems using the methods or RL follow similar learning patterns. They will be described in more detail below, but for now it is worth pointing out that the trained system does not have access to a man-made dataset. Instead, it independently extracts the necessary information from its environment, usually in the course of the following cycle: *observation of the state of the environment > action > new information (new state of the environment and punishment/reward)*. The method is used in training and improving a large percentage of the solutions discussed in the subsequent chapters of this book, with the use of dozens (if not hundreds) of different algorithms.

Last but not least, *meta-learning* is a very interesting class of machine learning solutions. In this approach, the intelligent systems learn to find the best learning strategies, which could be described as *learning to learn*. The idea is to accelerate learning processes, limit the amount of crucial data and number of necessary experiments (*trial and error*). It has been observed that the methods described above tend to require large sets of good-quality data, which can be both effort- and cost-intensive. Hence, there is a demand for more effective learning methods – in the case of meta-learning, this is achieved by *learning to learn*. In particular, meta-learning can be used in conjunction with RL (so-called Unsupervised Meta-Learning for Reinforced Learning, see e.g. Gupta et al. 2019), which basically entails teaching the agent how to apply algorithms developed in the performance of previous tasks to solve new problems. In other words, the system is taught how to identify and use previously learnt skills in the performance of new tasks, which is not unlike *transfer learning* in the context of psychology and traditional education.

As follows from the above, machines can be trained in a variety of ways. One of the ways in which such methods can be classified relates to their level of autonomy understood as the extent of human supervision and support. The *supervised*, *semi-supervised* and *unsupervised learning* methods require data to be provided by humans. In turn, *self-supervised*, *reinforcement* and *meta-learning* methods entail autonomous extraction of data from the environment. In each case, the result of the 'didactic process' is an imparted model capable of effectively solving new problems. But this begs the question: how does one go about improving it further?

Much like in the case of machine learning, several possible strategies could be adopted here. Presented below are two of the most popular: *batch learning* and *online learning* (cf. Géron 2019).

In the case of batch learning, the system is taught based on all currently available data. Once the data are collected and processed, the algorithm is trained, tested and implemented in production upon approval. If new data are acquired (externally or on the basis of prior experience), the cycle needs to be repeated: training a new model and replacing the one currently in operation. Of course, the process can be automatised (and repeated e.g. on a daily basis) but such an approach would not be practical in the conditions of continuous inflow of new data significant to the quality of the model, e.g. in systems forecasting stock exchange price levels. The primary drawbacks of batch learning include the need for constant supplementary learning based on large sets of gathered data, high computing power requirement (the model is trained from the ground up every time) and lengthy learning processes. For those reasons, such processes are usually performed 'in the background' – on external servers – i.e. *offline* from the perspective of the business process.

The other option entails training with the use of *online* systems. In this approach, the model is improved incrementally, based on new incoming data. This can be done continuously (data stream) or in short intervals (data divided into mini-batches). The strategy tends to be employed in situations where data significant to the model's quality are received in a continuous manner (e.g. a production line or stock quotations), or if hardware limitations are a factor: not enough computing power to retrain the entire model time after time, or not enough memory (operating or storage) to effectively manage large datasets.

The efficiency of online learning is affected by two key factors. First, a decision has to be made as to the extent to which model updates will rely on new experiences as opposed to previously gathered data. This factor is determined by the so-called *learning rate*. If the value setting is high, new data will have a considerable impact on the model, effectively forcing it to ignore historical experience. If the setting is low, the *learning rate* will minimise the impact of new data on the current mode of operation. This can be rather naturally referred to the way in which we, humans, update our knowledge: some people are quick to forget about the past and are able to flexibly adapt to changes in their environments. Others are more conservative: they survey the world, take new experiences and phenomena into account, but take considerably more time to alter their behaviour. Each of the options has advantages and disadvantages. As with nearly everything else, the trick is to strike a reasonable balance, which strongly depends on the current situation and environment.

The other important factor in the context of online learning (although also significant in the offline approach) is related to *data quality*. As data influence the model, they also impact the decisions and behaviours of the intelligent system. Bad-quality data (incorrect, incomplete, etc.) will significantly hinder system performance – not unlike bad-quality food weakening the organism. For this reason, machine learning has to ensure not only high efficiency of the learning algorithms but also the quality and proper preparation of input data.

To recapitulate the key points presented above, let us once again refer to the metaphor of human learning.

The *process of machine learning* usually includes the stages of gathering and preparing data (or developing mechanisms of acquiring the same), selecting the learning algorithm, training the model and testing its performance. The same is mostly parallel to the classical *didactic process* of e.g. school teaching.

The learning subject, or student, is the *model*. The *teacher* is usually a human although, as mentioned above, intelligent machines should have the capacity to teach themselves. The model (i.e. our 'student') learns based on *data*. The data may be provided externally (e.g. by the teacher) or obtained independently, through active interactions with the environment or passive exploration.

Just like in a real classroom, *the primary goal of the educator should be to make the student independent of the teacher.* Thus, the *teaching goal* in our

context is to 'train' the model to be able to effectively solve new problems – ones never encountered before (e.g. during the training process). Moreover, a good teacher will also instil the capacity for independent self-improvement (see offline and online learning mechanisms). As with human education, machine learning takes advantage of a varied range of teaching (model training) methods. In our context, machine learning algorithms (e.g. k-means or hierarchical clustering) are a direct counterpart of teaching methodologies.

At the beginning of the process, the model ('student') usually has no competences to speak of (with 'supplementary learning' being a notable exception). *The didactic goal* is to develop a very specific set of competences. *The educational results* can be precisely measured at practically every step of the way by evaluating prediction errors, classification quality or levels of punishment/reward received. The exact measures depend on the type of skill one wishes to train. For instance, in the case of classification, the same will be the surface area under the receiver operating curve, while for regression, it will be the root mean square error.

The purpose of evaluating learning progress is to determine the model's ability to act under new circumstances. Hence, in supervised learning models, evaluation should be based on data not previously used in the learning process – similar to a school where examined students should not be asked to solve questions previously done in class.

As follows from the above, one could draw many, more or less justified, analogies between the methods of human and machine learning. The reason they were included in these deliberations was to help readers less interested in exploring the mathematical and technological particulars of the problem to nonetheless develop an intuitive grasp of how intelligent systems develop and operate. They also provide a good starting point for the next section which will discuss in greater detail various concepts applicable to RL – a crucial contributing factor in the development of autonomous solutions.

BASIC PRINCIPLES OF REINFORCEMENT LEARNING AS A FOUNDATION FOR AUTONOMOUS, SELF-LEARNING SYSTEMS

In the preceding deliberations related to machine learning, the 'machine' (hereinafter the agent) was trained with the use of previously prepared data. In supervised learning such data were labelled and in unsupervised learning they were not. The agent learned to be able to cope with a new situation and correctly identify the 'features' of new (previously unencountered) data (in supervised learning) or identify certain patterns (in unsupervised learning). At the end of the day, the problems solved by the methods described above could be reduced to three primary categories: classification (prognosis of a discrete

variable), regression (recognition of a continuous variable) and segmentation. Whatever the case, however, the same always boiled down to 'recognition', i.e. interpretation of externally provided data: the agent took no action, did not interact with its environment and had no specific 'mission' to perform.

But intelligent structures are expected to do more than simply recognise and interpret their surroundings. An autonomous robot performing a complex mission has to function within a specific environment, integrate with its surroundings, receive feedback on the results of its actions and use all of that to achieve a certain ultimate goal. Such a process is considerably more complex and involves qualitatively new elements such as 'actions', 'response from the environment' or 'goal'. Presented below is the concept of so-called reinforcement learning – a method that underlies a vast majority of contemporary autonomous systems, be it robots, drones or conversation bots. The following paragraphs will discuss the primary concepts involved, interactive learning methods and key algorithms, while the most important challenges faced by such systems will be mentioned in the summary. An intuitive grasp of these methods will facilitate a better understanding of solutions discussed in the following chapters as well as the rather fascinating relationships between mechanisms constituting the basis of decision making as observed in humans, organisations and machines.

Agent, Environment, Actions, Observations and Rewards

Our analysis should start with the explanation of some key concepts.

The computer program that we wish to teach how to effectively act under new circumstances is referred to as the *agent*. In this context, *effective performance* is understood as the ability to solve sequential decision-making problems under conditions of uncertainty in order to maximise a pre-defined goal function.

It is noteworthy at this point that the agent is only the decision centre, a piece of software: its sole task is to *make decisions*, not implement them. Consequently, a robot working autonomically on a production line is not an agent; the agent is the software controlling its operation (which, notably, may just as easily be running remotely, e.g. in the cloud). Arms, motors and other actuators are considered *elements of the environment* as far as the decision centre is concerned. This distinction may become more compelling once we consider that robots often operate within stochastic environments, where a decision to move e.g. right does not guarantee (it may only provide a certain level of probability) that a move to the right will indeed be executed: in practical applications it may indeed happen that the resulting motion will be to the left. We have all experienced a similar situation at one time or another, e.g.

when trying to walk on ice: the decision is one thing, but actual execution is a different story. . .

The agent has a certain inherent structure: interfaces that allow it to communicate with its environment (receive information about the state of the environment and transmit decisions concerning respective tasks) as well as the decision centre itself (the 'brain'), which makes decisions with the aim of maximising the so-called goal function, evaluates their effects and learns so that each subsequent iteration of the action yields even better results and contributes to mission execution. Therefore, the key components of the agent include *interfaces* and the '*brain*', while the primary processes involved are *interaction*, *evaluation* and *improvement*.

The agent operates within a certain *environment*. It encompasses everything external to the agent, of which it has no complete control. In particular, the arm of a robot is an element of its environment: as already mentioned, the intention of moving it to the right cannot guarantee such motion (e.g. due to failure or an obstacle).

From the perspective of the agent, *the environment is represented by a set of variables related to the problem*. For a robot, these could include e.g. the position of the object that is to be moved or the position and speed of the arm it uses. For a drone, it could be its speed (in three dimensions) or position and rotational speed of the respective rotors. *The state of the environment* is defined by a set of values assigned to said variables at a given time. It is a subset of the so-called *state space*: the set of all variables describing the environment and all of their possible values. In turn, *observation* performed by the agent at any given moment is a part of the state it perceives at the time. The distinction between the state of the environment and observation is very important: it is often the case that the agent has access only to a small subset of information regarding everything that is happening to it and around it. A good parallel would be the spectrum of electromagnetic waves we are able to perceive: only a fraction are visible but that does not mean we are not surrounded by e.g. radio waves.

The agent usually has a specific *task* to perform within the given environment. The task may have a definable conclusion (e.g. in the game of chess) or not (e.g. navigating a given space). The former are referred to as *episodic* and the latter as *continuous tasks*. In the case of episodic tasks, the sequence of steps from the beginning to the end of the task performance is called an *episode*, while the total of rewards collected in the course of a single episode (e.g. gold picked up while navigating a labyrinth) is referred to as the *return*. The agent is usually programmed to make decisions aimed at maximising the return; it may, however, repeat the episode several times to learn the optimum pattern. In the process of learning, continuous tasks are often limited by impos-

ing certain finite stages (e.g. number of steps), which brings them somewhat closer to their episodic counterparts.

The *goal* of the task is indicated by a special signal transmitted to the agent by the environment and described as the *reward*. Its value may be positive, negative or zero – whatever the case, the RL terminology will refer to it as a reward. The manner in which the environment rewards the agent is critical to the learning progress – not unlike in human learning in fact. The issue will be discussed in greater detail in the rest of this chapter (cf. Morales 2020) but it should be noted at this point that rewards may be delayed in time to a varying degree, awarded with varying frequency or have a varying nature (evaluative or instructive).

In some cases, this delay can lead to a situation where the actual consequences of the action taken will only become apparent after some time: immediately after eating a cookie, we feel great (high reward), but after a month of having ten cookies for supper every day we will start receiving troubling signals. The rewards evaluating our actions may be given very frequently (e.g. after each activity) or more rarely (e.g. once in every ten actions). The former case could be compared to a mother praising or criticising her child every other minute – this can accelerate learning but puts the agent at risk of copying the behavioural patterns of the teacher (mother), therefore hindering its future self-reliance. In the context of machine learning such a situation is described as *overtraining*. Less frequent feedback will have the opposite effect: slower learning but more space for experimentation, which helps to develop independence and discover new, more effective methods. Finally, *evaluative* feedback refers to the values of states achieved after the performance of actions, while *instructive* feedback reflects the values of the actions performed. The function determining the reward that the agent is to receive in the given state of the environment is called the *reward function*.

The goal of the agent is to perform a task posed to it by the environment. In order to do so, it must have the capacity to make decisions maximising the *total* reward (in the case of episodic tasks: the sum total of the returns accumulated during the task), which in turn requires *long-term planning*. The agent should also be able to *gather information* and, due to the fact that it cannot explore all the states of the environment, *generalise* its experiences to account for other, similar states. At the end of the day, it should have the ability to create the optimum *policy*: method of choosing actions in response to the given situation with a view to performing the task. From the mathematical point of view, it means the ability to learn the function mapping observation into action.

Tasks are performed through the agent's *interactions* with the environment (Figure 1.4). This usually takes place in cycles comprising *action*, *feedback*, *reflection* and *update*. In the first stage, the initial state of the environment is surveyed. Next, based on the available knowledge, a decision is made to

choose the optimum action. The action is performed, and feedback is received to determine the state of the environment resulting from the action (i.e. the influence exerted) and the reward received. Next, the effects are evaluated, policy is updated and the cycle begins anew.
The cycles can be scaled to very diverse time intervals – from milliseconds, through hours, to days – depending on the given problem and environment. After each cycle, the agent gains a certain *experience* represented by a specific dataset (often referred to as *tuple* in information technology jargon) pertaining to the initial state, undertaken action, resulting state and reward. The experience is usually gathered as a separate dataset used in the process of self-improvement, both by individual agents and entire teams.

In each state, the environment *allows* the agent to undertake certain actions – the scope of which can vary in time. As already mentioned, the action may (but does not have to) influence the environment. The function describing the way in which a given action affects a given state (i.e. mapping action > state) is called the *transition function*.

Now that we have discussed the basic terms and pattern of agent behaviour, we can turn our attention to two other, very important concepts: *Markov Decision Process* and *world model*.

When updating its performance, the agent has to make certain assumptions. One of the most important of the same is the so-called Markov property which can be explained in a single sentence: 'The future is independent of the past given the present'. In other words, the future of a system depends solely on the decisions made in the present – the past is no longer a factor. This rather extreme premise proves to be very effective in practice, it greatly simplifies calculations and facilitates solving complex problems. Decision processes consistent with the Markov property are described as *Markov Decision Processes*.

A *world model* can be compared to a map of a new territory: having the same will greatly increase the chances of finding the optimum solution. With a map of a city, one can easily find the shortest route from point A to point B without having to explore, which reduces the scope of the problem to simple planning. If one has no map, one is forced to try different variants and gradually create one, which takes time and, if obstacles emerge, can also prove risky.

In summary: actions performed by an agent alter the state of the environment (which is described by the *transition function*) and, in the given state, reward is provided as dictated by the *reward function*. Within the framework of RL, knowledge of the world model comes down to knowing those two functions: transition and reward. In other words, we could say that we know the world model if we know (1) what state *s'* will result from our action *a* performed in the state *s* and (2) what reward *r* will be received in the state *s'*.

Figure 1.4 *Simplified model of interaction between an agent trained through reinforcement learning and its environment*

Source: Own elaboration.

Learning through Interaction

With the basic nomenclature explained, it is time to take a closer look at some key concepts related to the methods of teaching agents as to how to efficiently perform tasks in an environment.

As already discussed, interactions between the agent and its environment take place in the cycles of observing the state of the environment, performing an action and gathering feedback (new state and reward). Each cycle is described as a *time-step*, while the complete set of information gathered during each time-step (initial state, action, final state, reward) is called *experience*. It was also mentioned that the gathered experience is evaluated by the agent (*reflection*) in order to *update* the decision-making policy. In other words, the agent improves its performance by learning through trial and error based on the experience gathered, which constitutes the essence of RL.

Feedback is crucial to the learning process. The forms and types of such information depend strongly on the given task and environment, which pose considerable challenges and greatly affect the learning strategies. The issue will be discussed in detail below – gaining an intuitive grasp thereof is not only key to understanding the problems of RL but may also prove useful in personal life.

Firstly, feedback from the environment can be *staggered*. The consequences of choosing a given action in the present state may only become apparent in a more distant future (not necessarily in the next time-step). The goal of the agent is to undertake *here and now* actions aimed at maximising the total of rewards gained during the episode, and not at maximising the rewards applicable to subsequent states. The agent must therefore be able to learn from such staggered feedback as well.

Strategies can vary and are highly dependent on the problem. For instance, in the case of episodic tasks with the possibility of performing multiple episodes, the agent may learn the world model on an episode-by-episode basis (i.e. discover the transition functions and rewards), and based on the continuously perfected model, make decisions with the aim of maximising all the rewards (from all the time-steps). This becomes somewhat problematic if one has only a single 'life' at one's disposal, calling for a different strategy, e.g. relying on the experiences and value systems of other agents (in our case: humans), which as we know, is hardly easy and in no way guarantees success.

Another factor diversifying information received from the environment is the *character of feedback*. In their acclaimed textbook, Sutton and Barto (2017) compared RL to other machine learning methodologies and concluded that a reward constitutes a signal evaluating the quality of action (so-called evaluative feedback) rather than an instruction indicating the correct action (so-called instructive feedback). This interesting classification of machine

learning methods based on distinguishing between two types of feedback received in the learning process (*evaluative* (assessment of the consequences of action) and *instructive* (recommendation of the best action in the given state regardless of the actual action of the agent)) may be easily applied also to the methods of human education.

Instructive learning basically boils down to imparting a certain set of *what-if* procedures: *in this state you should do this*. The role of the teacher is to determine whether the action was consistent with the procedure and evaluate the discrepancy (i.e. error). The main limiting factor in this context is the experience of the teacher (or the available training dataset): future problems may emerge in situations that the teacher was not aware of (or simply did not think to account for in the process of generalising his knowledge). In the case of *evaluative learning*, the role of the teacher is assumed by the environment itself (i.e. the situation in which the agent finds itself). It informs the agent of the consequences of a given action (good, bad, neutral) without suggesting the optimum course of action. It is the learner that has to evaluate the action taken in the given state (situation), which is not easy as it requires disciplined experimentation and honest self-assessment. At the same time, however, it creates the potential for exploration and finding new, more effective policies. In short: one can either be taught through instruction, by pointing out the adequate behaviour in a given situation, or through encouragement to undertake varied action, with the only support coming in the form of feedback on the consequences thereof. The former approach is characteristic of supervised learning, the latter of RL. Naturally, in practical applications there are various methods seeking to find a middle ground between these two extremes; nonetheless, the above classification will be very useful to us.

To recap, in the case of RL, the agent must be able to learn from evaluative (rather than instructive) feedback. The knowledge as to which action is optimal in the given state should be generated by the agent itself, and then applied to making effective decisions. Interestingly and quite paradoxically, knowing the values of respective actions is, in itself, insufficient: were we to always make decisions that our experience indicates as best in the given situation, we would never have a chance of verifying the possible consequences of sub-optimum actions.

In RL this phenomenon is referred to as the *dilemma of exploitation and exploration*. *Exploitation* is the policy of always choosing the action that our experience tells us will yield the highest reward possible. Unfortunately, this approach has two inherent risks. The first is the already mentioned inability to test the effects of actions other than the alleged 'optimum': the value of an action can be determined through experimentation, and exploitation prevents it by its very definition. The second risk stems from the possible changes to the environment: a decision that was optimum ten years ago may be less than

so today. Thus, a given incentive system may be optimum in one group of employees but may fail completely and even disincentivise others. As evident from the above, striking a good balance between *exploration* and *exploitation* is quite the challenge, not only in the context of machine learning.

Another key competence of a digital agent is the ability to *generalise*. To reiterate the relevant point, the agent learns by interacting with its environment, which generates feedback. It surveys the state of the environment, takes action, checks the state resulting from the action and evaluates the potential reward. Let us now consider the same process from the perspective of action and state space.

Again: the state space is the set of all possible states which the environment can have, and the action space is the set of all possible actions available to the agent at a given time (or more precisely: that the environment allows in its given state). Spaces can be both discrete and continuous, and this characteristic applies to all possible values assumed by variables describing the given environment or action. This sounds somewhat abstract, so it would be prudent to illustrate the point with a few examples.

Firstly, let us consider the so-called grid-world. Imagine an agent travelling across a 5×5 square board, starting its journey from the top-left square. The goal is to reach the bottom-right corner while maximising the reward collected on the way: there are various 'positive' (let's say gold) or 'negative' (e.g. traps) rewards waiting on the respective squares of the board. In this case, both the state space and action space are discrete. Each state is described by two variables (x and y) with a finite range (1 to 5). The available actions are also discrete: up, down, left, right (in some extreme positions only e.g. right or down).

Another 'classic' example is the so-called *inverted pendulum* problem. Imagine an agent whose task is to control a horizontal platform on which a vertical stick is balanced. The idea is to prevent the stick from tipping over for as long as possible (the scenario is performed with software simulating the laws of physics).

In this case the state space is continuous: even though the state of the stick can be described with only four parameters (horizontal position, horizontal speed, angle of inclination and angular velocity), each can assume an infinite number of values. Meanwhile, depending on the environment, the space can be discrete (e.g. move left or move right) or continuous (e.g. move left with speed v_l). The question whether the state space is discrete or continuous is immensely important when choosing the RL method of training the agent. In the case of discrete spaces with a limited number of dimensions, one could easily imagine a situation where the agent, given sufficiently many episodes (e.g. when training on a simulator) will ultimately 'visit' all the states and perform all the actions possible therein. This may take some time but (assuming that the environment is deterministic) the agent will be able to develop a complete

world model based on experience, which in turn will practically reduce the problem of exploration to simple planning. After the subsequent update, the policy will be reducible to a function which, given sufficient input data describing the given state, will return parameters describing action. Notably, due to the limited number of possible values attributable to the parameters of states and actions alike, that function will be multivariate but with a finite number of dimensions, matrices. In other words: the policy will be reduced to a finite number of *what-if* rules, where both *what* (action) and *if* (state) will have a finite number of possible values.

However, things tend to get significantly more complicated if one of the spaces, be it states or actions, becomes continuous. In this case, the variables describing e.g. the state may take an infinite number of possible values – which means that compiling a complete world model (i.e. two functions: (*state; action*) > *new state* and *state* > *reward*) would require an infinite number of experiments (due to the need to 'visit' all the possible states and try all the possible actions). And clearly, that is an impossibility.

This is exactly why we expect our agent to have the capacity for *generalisation*. It allows the agent to use its limited experience to formulate policies capable of recommending optimum action in a given state[3] even if it is potentially infinite. Generalisation most commonly entails attributing a recommended action to a certain value range of variables describing a given state. For instance, in the case of the *inverted pendulum*, the policy could include rules such as: *if the inclination is in the range of 85–90 degrees left and the speed does not exceed 10 cm/s, move left*. Notably, the agent should be able to generalise not just the policy (function of *action* > *state*) but also the transition function (map of *(state, action)* > *next state*) as well as the so-called value function (map of *(state, action)* > *value*).

In practical implementations, strong, non-linear approximate functions are used for this purpose: multiple mathematical transformations of input variables (cf. Nielsen 2015). The methods and techniques of machine learning, in particular neural networks, prove rather useful in this context. RL models utilising deep-learning neural networks go by the name of deep RL.

To recapitulate, in order to effectively operate in environments where continuous spaces of states and/or actions are present (i.e. nearly all practical applications), the agent should have the capacity to learn from fragmentary feedback: i.e. information reflecting only a small subset of all possible state <> action configurations. This can be achieved if the agent is able to generalise, which in turn is facilitated by approximators such as artificial neural networks.

Before we proceed to present the strategy of developing the above-mentioned skills in RL agents, let us reiterate (after Russell et al. 2010 and Morales 2020) the classification of environment features, levels of agent knowledge and types

of feedback, which will significantly aid us in navigating the dozens (if not hundreds) of different possible problems and RL algorithms.

Let us first have a look at *environment features*. As already mentioned, *the spaces of states* and *actions* can be either *discrete* (finite) or *continuous* (infinite). In turn, the functions of *transition* (i.e. map of *(state, action) > new action*) and *reward* (map of *state > reward*) may be either *deterministic* (when it is certain that e.g. action *a* in state *s* will produce state *s'*) or *stochastic* (when there is no such certainty and one can only estimate, based on previous experience, the probability *P* of action *a* performed in state *s* producing state *s'*).

Another element facilitating the classification of RL problems is the *level of the agent's knowledge about the environment*. As far as the *space of available action* is concerned, in practice it is always assumed that the agent had complete knowledge of the same. It is another matter entirely, however, when it comes to knowledge of the *state of the environment* – here, there are three possible scenarios: *the agent has exhaustive knowledge about the state of the environment* (i.e. *observation = state of the environment*), *the agent has only partial knowledge* (*observation* is a subset of the *state*: the agent knows something but does not have full insight) or *the agent has no knowledge of the state* (it is 'blind'). When it comes to knowledge of the *world model* (i.e. of the transition and reward functions), the agent can either know them (each taken individually) or not (in which case it will be forced to learn). Agents can also have varying access to information about the *reward*: again, this knowledge can either be exhaustive or sampled.

Finally, the determination of *reward* characteristics is also helpful in classifying RL problems. Firstly, the *moment* of receiving it is a factor: the agent may be rewarded immediately after completing an action (so-called one-shot reward) or over a certain period of time (reward is delayed) – in which case it is said that the reward is sequential. Another factor is the actual character of the reward: as already mentioned, it can reflect the consequences of actions (so-called *evaluative* feedback) or simply constitute an *instruction* (supervised feedback).

The classification in Table 1.1 can significantly facilitate the identification of optimum strategies and algorithms applicable to the agent's training. For instance, the agent can operate in an *environment* characterised by a discrete state space, continuous action space, stochastic transition function and deterministic reward function, with only sampled *knowledge* of the current state, no knowledge of the transition function, but exhaustive knowledge of the reward function, with an *evaluative* reward being provided *immediately after* the action.

Table 1.1 Classification of reinforcement learning problems

	feature of the environment	agent's knowledge
states	discrete / continuous space	exhaustive / limited / none
actions	discrete / continuous space	known
feedback	immediate / sequential	exhaustive / sampled
feedback	evaluative / instructive	exhaustive / sampled
transition function	deterministic / stochastic	exhaustive / none
reward function	deterministic / stochastic	exhaustive / none

Source: Own elaboration.

Before we proceed to present the main RL algorithms, let us first summarise our deliberations up to this point by indicating the most interesting challenges faced by an agent aiming to learn how to effectively perform in a given environment. Above all, the agent *must resist the temptation to immediately aim for the maximum reward*. It often needs to assume that although the choice of a given action promises to yield a high reward in the short term, all hell might break loose later on… If the reward is to be maximised in the perspective of the entire episode ('life'), adequate balance must be struck between fast but minor immediate gratification and delayed but overall much larger reward in the future. Naturally the key word in this sentence is *balance*: excessive ascesis is also not without many inherent dangers – indeed, this is the very source of the challenge.

Another problem relates to the *infinite multitude of states and choices*. Too much choice is not a good thing: a fact that any sales person or aspiring violinist will readily corroborate. Hence, due to often limited possibility of experimentation, the agent must learn the art of wise *generalisation*: to classify states and actions so that ultimately, a finite, manageable number of choices is returned. In this case *wise* is the key word: overly simplified classification, e.g. binary – reduced to dichotomies such as *good–bad* or *red–blue*, tends to be a dead end (leading to overly stereotypical thinking). On the other hand, an excessive number of categories yielding thousands of potential options will prolong and complicate decision-making processes, thus negatively affecting performance. In short, the agent should be able to generalise, but wisely.

The last important dilemma derives from the type of feedback and comes down to the choice between *exploitation* and *exploration*. Should I always act based on knowledge, backed with experience of what is the best for me? In other words, should I always choose actions which, based on my knowledge, are optimum? Or should I go wild from time to time: take a risk in the hope of unexpectedly arriving at new, even more attractive states? And if I were to combine the two, how do I decide on the ratio? Clearly, this is yet another

case where the agent should strive for a certain optimum balance – between following the safe route and taking risks in search of better ones.

Such inherently human dilemmas are approached, in a variety of ways, at the level of algorithms which, when implemented in software and end devices (e.g. a robot controller), allow the machine to become increasingly autonomous and, in a sense, more rational. The specific business applications thereof will be discussed in Chapter 2; for now, let us focus on the most important algorithms used in RL.

Key Reinforcement Learning Methods and Algorithms

As already mentioned above, RL algorithms come in a wide variety of shapes and forms, while the dynamic growth of this field of study continuously stimulates the development of new ones. To gain some measure of understanding of their possible applications, strengths and weaknesses, one needs to first adopt a simple, manageable taxonomy thereof.

One of the most popular classifications of RL algorithms (cf. OpenAI: Spinning Up 2018) distinguishes them first in terms of the level of knowledge about the environment, and then based on the object of learning as such. Knowledge about the world can be based on a certain world model (*model-based RL*) or not (*model-free RL*). Furthermore, *model-based* algorithms can be divided into those where the model is externally provided (*given the model*) and those that require it to be learnt (*learn the model*). In turn, *model-free* algorithms come in *value learning* and *policy optimisation* variants. Each of the mentioned categories will be briefly characterised below.

Let us first consider algorithms teaching the agent how to evaluate its decisions in a situation where it has *exhaustive knowledge on the state of the environment*, *the reward is delayed in time* and the *agent uses a world model* when making decisions. It is a rather rare and untypically convenient situation for the agent; nonetheless, it will provide a good starting point in the introduction of the ideas underlying RL algorithms. A classic example of such a problem is the already discussed task of navigating a labyrinth: travelling from point A to point B in such a way, so as to maximise the reward in a situation where the rewards waiting on respective squares can be either 'positive' (e.g. gold) or 'negative' (e.g. traps).[4] At any given moment, the agent has exhaustive knowledge about the state of the environment (agent's position), its task is to maximise the reward sum total and it can make an unlimited number of attempts in the learning process (*it has endless lives*).

The goal of the agent (i.e. also the 'didactic goal' of the RL process) is to identify the sequence of events that yields the maximum cumulative reward in the given episode (return). The goal can be reformulated as finding the optimum policy, where *optimality* refers to maximum return in the course of

the episode, and *policy* means the manner of choosing the action to be performed in each of the possible states.

Policy is a key concept in this context. From the perspective of mathematical formalism, it is the function (map) of *state > action*.[5] Naturally, the agent will have a choice of several possible policies within a single environment: e.g. in our labyrinth it can determine that it will always start by turning left, then proceed two squares forward, or turn right and move three squares forward, etc. Notably, such policies can be *deterministic* (specifying only a single action in a given state) or *stochastic* (recommending the probability distribution of actions in a given state).

Another vital element of the training process is the *level of knowledge about the world model* (i.e. the functions of transition and reward). As already stated, the function of transition is the map of *(state, action) > state'*, whereas the reward function is the map of *state > reward*. If the agent knows both these functions, it has exhaustive knowledge of the possible consequences of actions in the respective states as well as the rewards it can expect to obtain. If complete and accurate, such information reduces the problem to mere planning:[6] identifying the optimum policy, rather than learning the optimum course of action.

So how does one define the policy in a situation where one has full knowledge of the consequences of any action in any state? The problem basically boils down to the following question: *of all the possible policies, which one is the most valuable?*, which in turn leads to the problem of *determining the value of a given policy*.

The policy for each possible state indicates the course of action that the agent should take: in the case of the example *grid-world*, knowing the world means being able to identify the route for traversing it in accordance with any given policy. At the same time, the agent is perfectly aware of the reward that any given state can yield. Consequently, assuming that we always start on square A and finish on square B, and that the world does not change in the course of the model (admittedly, some very strong assumptions…), the value of each strategy can be calculated as the sum total of all rewards collected by navigating the route indicated thereby. And that value provides the basis for identifying the optimum policy.

Naturally, in practical applications the calculation of policy value tends to be somewhat more complex than that. Two new concepts come into play in this process: the *state-value function (V)* and the *action-value function (Q)*.

The state-value function Vπ(s) allows a certain *value* to be attributed to a state. This value is understood as the expected sum total of all the rewards that will be collected by an agent beginning its 'journey' from the given state, provided that it proceeds in accordance with the given policy. In our case (*grid-world*) calculating the function will assign each square on the board with

a certain value, which informs the agent: *if you start your action from this square and follow the policy π, this is the reward you will get*.

An interesting question worth considering when calculating the state-value is the delay of reward. The dilemma is this: when we find ourselves in a given state, to what extent should we account for rewards obtainable in a very long-term perspective when assessing its value? RL methodology solves the dilemma as follows. The value of a given state is calculated as (1) the reward obtainable immediately after achieving the same, plus (2) the value of the subsequent state (i.e. the next state to be reached when following the given policy) times the discounting factor which can assume values from 0 to 1. The decision on the actual level is, in a way, a measure of the agent's (or more specifically, the teacher's) hedonism. If the value is 0, it means we proceed solely based on the benefit generated by the given state (extreme hedonism): subsequent rewards have no importance whatsoever. If the value is 1, all possible future rewards are equally as important as the immediate reward obtainable upon reaching the given state: this extreme means maximising the reward in the perspective of the entire episode ('life').

Another fundamental function in many algorithms is that of *action-value* $Q\pi$ (s, a). It determines the total cumulative reward to be obtained by an agent beginning its journey in the state *s* and taking action *a* in this state. At first glance, it would seem that the function's role is only auxiliary: knowledge of the value function and policy should be sufficient to evaluate a given policy. But this approach has been found to work only in the situation of exhaustive knowledge of the world model, which is rather uncommon in practical applications.

With the defined and calculated functions of states and actions (without complicating things with many important methods that allow the same to be actually plausible in practice, e.g. Bellman equation or dynamic programming), we can now turn to the problem of identifying the optimum policy.

To once again use our example, let us imagine an agent in square *s* faced with the choice of one of four available actions: go up, down, left or right. Which one should it choose? The answer can be fairly easily obtained on the basis of (1) one's knowledge of the world (in this case knowledge of the state one will end up in after choosing each of the options), and (2) the pre-calculated function of the state-value. The first policy that immediately springs to mind is simple: let us choose the action that will lead us to the state with the highest value. In fact, this is so inherently tempting that the RL nomenclature tends to refer to it as the *greedy policy* – but, as will be demonstrated next, it does not necessarily have to be the best choice.

In our previous example, the agent knew the world model: it was able to precisely predict the consequences of its actions in all the states; in particular, it knew what states would follow from all of the possible actions and

what rewards would be gained as a result. Hence, the problem of identifying the optimum policy came down to proper planning: using the functions of state-value and action-value, the agent could identify the optimum policy without the need for experimentation (exploration). However, the process becomes somewhat more complex in a situation where the agent cannot use such a model. We then have only two possible options: either find a way to learn the same or find another method of identifying the optimum actions. Methods following the former approach are classified as model-based (the agent initially does not know the model but decides to learn it), while algorithms that do not rely on a world model at all are called model-free (as it turns out, one can do pretty well even without a 'map').

Let us first discuss the methods of cognising the world. As mentioned before, knowledge of the world is equivalent to knowledge of the transition function (map of *(state, action) > state'*) and reward function (map of *state > reward*). But how does one acquire this knowledge?

Firstly, we could consider a rather convenient situation where the task is episodic in nature (it has a beginning and an end), and the agent can learn by performing an unlimited number of episodes (there are no limitations to exploration). In this case, learning can be based on complete episodes by repeating the experiment numerous times and registering all the states reached by the agent, consequences of the actions taken and rewards gained. In time (with sufficiently many repetitions), we would end up with the complete set of information needed to build the world model (transition and reward functions), which in turn would be sufficient to identify the optimum policy (calculate the function of state- and action-value, etc.). The methods allowing calculation of the action-value function based on complete episode iterations are referred to as Monte Carlo (MC) methods.

However, in the real world, the agent's situation is hardly ever convenient enough to allow actually 'visiting' all the states and testing all the possible options (imagine trying to taste all the dishes offered in all the restaurants in your city…). This may be due to time or resource limitations (e.g. the power of computers used for simulations) and, naturally, the greater the 'density' of states (number of possible values of the respective parameters describing a state) and actions (number of actions possible in a given state), the greater the number of options that need to be examined. Consequently, it becomes crucial to develop a strategy for *approximation* (rather than exact calculation) of the action-value function. RL terminology refers to this as the *problem of prediction*: the quality of prognoses relative to the value of a given action in a given state is key to identifying the optimum policy, which comes down to the simple observation that *a policy is only as good as the prognosis used in its development*. A well-estimated action-value function (i.e. knowledge of the

value of the given action in the given situation) constitutes the basis for identifying the optimum policy (solving the so-called *control problem*).

In situations where estimating the functions of state-value and action-value based on complete episodes is found to be unfeasible, methods other than MC have to be employed. The first of such methods entails prediction on the basis of temporal difference (aptly called the temporal difference method (TD)). The initially randomly initiated function of value is later updated step by step (and not at the end of the episode as in the case of MC), whereby a step is understood as any single interaction between the agent and the environment (i.e. a sequence of observation > action > feedback). Updating the value of a given action in a given state comes down to adjusting the current value by accounting for the so-called prediction error: the difference between the current estimate and the actual experience. In considerably simplified terms, this means that if the value of the new state calculated based on the current state-value function is at level V (estimate) and the actually experienced value is at level T (target), we should adjust the value function for the given state by T−V (i.e. so-called prediction error). The difference between MC and TD methods stems from the fact that in the former, the value function is updated only at the end of each episode, whereas in the latter, after each consecutive step, which significantly reduces the time needed to learn what is preferable (the value function).

Methods such as TD and its variants (e.g. n-step TD or TD-Lambda) aim to address one of the major limitations of RL – that is, limited availability of samples or possibility of conducting experiments (so-called *sample efficiency*). Learning by trial and error is safe in environments simulated by computers (although even then we can encounter limitations such as available computing power) but can be less than that when conducted in the actual physical world; just imagine using this method when training an autonomous vehicle, with the length of one episode defined relative to the life expectancy of the driver or other road users.

The methods of identifying the optimum policy described so far were based on *learning the value* (so-called Q-learning) of both *states* and *actions*. It turns out, however, that an effective policy can also be developed by directly optimising the policy itself – some of the major methods following this approach will be presented below.

First, however, let us reiterate some of the main points already discussed. Knowledge of the world model (i.e. functions of transition and reward) allows one to identify the optimum strategy (one that maximises the total attainable reward) without the need to interact with the environment. However, the practical applicability of such an approach is negligible, hence the agent is forced

to learn by trial and error. As it explores the environment, in each new situation it encounters it must answer the following questions:

1. What do I ask the environment? What information about the given state is the most important to obtain?
2. How good are my current estimated action-values?
3. What task should I engage in now?
    a. Should I act based on my previous experience (choose the action with the highest value score)? < *exploitation*
    b. Or should I take a chance and attempt a sub-optimum action in this situation? < *exploration*

As clearly follows from the above, the *exploit or explore* dilemma plays a very significant role in the process, becoming one of the key elements of the ultimate policy.[7] The key to finding the right balance can be expressed as follows: *our goal is to develop the best possible policy, so that we can later exploit it; but in order to do so, we must first engage in intensive exploration*. In this sense exploration builds the foundation for future, effective exploitation. Morales (2020) illustrates this with a rather apt analogy: It is like searching for a gold vein in El Dorado: before you can effectively exploit it, you must first spend a lot of time looking for it (perform a lot of explorations).

Exploration strategies in RL can be divided into *random*, *optimistic* and *based on the space of information states*. An agent following the policy of *random exploration*, in most cases chooses the action with the highest value (maximum result of the state-value function Q), but at times will also opt for a lower value action. The term *at times* is defined by the parameter ε which can take values from 0 to 1 and denotes the probability of taking a chance and selecting an action other than the one known to yield the highest value (based on our current experience). The value of ε is a policy parameter. If it is set to 0, the agent will follow the greedy policy (no experimentation), whereas if ε=1, the agent will completely ignore previous experience and always opt for exploration. Notably, after each such experiment, the function of Q is updated – the learning process continues. A good example of a practical application of the 0.25 value could be a policy where in 75 per cent of the cases, we select our favourite food at the restaurant, while in the other 25 per cent of the cases, we follow the recommendation of the waiter. Clearly, as with the dilemma of delayed pleasure determined by the parameter of (hedonism <> salvation after death), here too (conservatism <> recklessness) many analogies can be found with everyday choices. The primary difference, however, lies in the number of available episodes: reinforced learning usually provides many more than we can hope to enjoy in real life.

Naturally, random policies come in many diverse variants. In practical applications, training a model begins with a high level of exploration, which tends to be gradually reduced as the process progresses through subsequent episodes (by lowering the ratio). A somewhat more refined model (so-called SoftMax exploration strategy) correlates the probability of choosing a given sub-optimum action with the value of such action – one could describe it as *random with a preference for…*

Optimistic strategies such as *optimistic initialisation strategy*, *upper confidence bound* or *Thompson sampling strategy* are based on the assumption that high uncertainty is welcome in the decision-making process (somewhat along the lines of *no risk, no fun*). By randomly assigning the initial values of the value function, we initiate it together with the function associating each action with a certain level of confidence. Next, in each subsequent step, we quantify the uncertainty related to respective decisions made and promote the decisions with higher uncertainty in the process of exploration. In *information state-space strategies*, the information states (including uncertainty) of the agent are treated as features of the environment. It perceives uncertainty as an additional dimension of information about the state, and consequently less explored states are perceived differently.

How can one approach the task of streamlining policies? The most popular method employed in a variety of algorithms is the so-called *General Policy Optimisation*, which entails an iteration of three, deceivingly simple steps:

1. Take the current policy.
2. Evaluate it.
3. Improve it.

(repeat until you reach satisfactory results)

Although seemingly trivial, these steps have been practically employed in the development of many inspiring methods and techniques.

Let us then have a closer look at what exactly is meant by *evaluation* and *improvement*.

Starting from the beginning, the 'entry level' policy determines the agent's actions in a given state. It could be e.g. the greedy policy: where in each state the agent always opts for the highest-value action.[8] Or one of the more or less explorative ones: the highest-value option is chosen in most cases but, now and again with the probability of ε, one lets luck decide the outcome. It can also be any of the other possible variants. What matters is that the policy is 'accompanied' by estimation of the action-value function $Q\pi$ (s, a) which assigns each of the possible actions in each of the possible states with a certain value understood as the total reward expected when performing the given task in the given state in accordance with the initial policy.

The second stage entails evaluation of the policy's effectiveness. The key word in the previous paragraph was *estimation*. Our goal is to improve the current policy – i.e. we humbly acknowledge that we have less than complete confidence in the estimated value of function Q and agree to conduct an experiment to verify it. This is particularly important in the context of early iterations when estimations of action-values are pretty much random (or assumed based on experience from other projects, available heuristics, etc.).

And how does one go about verifying the accuracy of one's estimates as to what is and what is not correct? By experimenting of course! The agent *gains new experiences*, usually through interactions (i.e. sequences of *observing the state > taking an action > gathering feedback*) but also, in the case of the *model-based method*, by consulting the world model. After conducting the experiment, i.e. gathering new experiences, *the agent updates its assessment of the value function* using methods such as MC, TD Learning or others. This is a key stage in the learning process: we begin with the 'old' assessment of the action-value and state-value (function Q) and arrive at a new assessment backed by experimental results (function Q').

The final, third stage entails *updating the policy*. It usually comes down to replacing the original function of Q with the updated value function Q' and/or modifying other parameters, e.g. ε. The thus modified policy becomes the 'entry-level' policy in the subsequent iteration.

At this point it is worth underscoring the critical value of stage 2: estimating the quality of the policy. This could be compared to broadly understood critiques: customer opinions, feedback from the trainer or self-reflection. The idea is simple: we have certain rules, we implement them, ask for feedback from the environment and change accordingly. In this context, it is very important to remain aware of certain factors such as that our *critic* may be biased or have bad habits himself, as well as remember that pleasing everyone is a widely recognised impossibility. For these and other reasons, the process of improving value function estimations should always be meticulously planned and executed.

There are many other aspects of *General Policy Iteration* that ought to be considered, for instance, which policy to employ when generating new experiences: our current policy (in which case methods are categorised as *on-policy*, a good example being the SARSA algorithm) or an entirely different one (e.g. someone else's experiences – such methods are categorised as *off-policy* – e.g. the *Q-learning* family of algorithms). There is also the question of when to update the policy value estimation: after each episode (as in MC methods – the approach is described as *offline* learning) or after each step (*online* learning).

The limitations of this analysis prevent a more in-depth look into those interesting but very technical issues. One should note, however, that *continuous state and action spaces* create an entirely separate category of problems

forcing agents to develop a capacity for generalisation and stochastic policies (i.e. ones that recommend only a certain probability distribution of action in a given state). The method of generalisation takes advantage of deep neural networks and such methods as deterministic policy gradients, including an entire range of *actor-critic* algorithms.

Challenges

There are still many challenges related to RL. At the same time, the field continues to dynamically develop and RL algorithms are used in a growing range of applications, which only adds to the complexity of the matter. Dulac-Arnold et al. (2019) provide an interesting synopsis of problems faced by autonomous system developers:

1. *Effective learning offline*, i.e. based on already gathered experience rather than direct interactions between agents and their environments. Trial and error methods are not always safe (e.g. vehicles) or effective (e.g. experiments with various strategies in online marketing where the perceivable effects of actions can be significantly delayed).
2. *Fast learning based on limited data* (the problem of sample efficiency).
3. Learning in multivariate continuous state and action spaces.
4. *Safety* issues in learning processes.
5. *Learning tasks performed in conditions of high uncertainty* (e.g. only partially observable or stochastic environments).
6. Learning in situations where the goal functions (reward systems) are underdefined, susceptible to risk or oriented towards multiple goals.
7. The ability to explain the learnt action rulesets to operators supervising autonomous systems (so-called self-explainable AI).
8. Inference that has to take place in near real time.
9. *Extensive and unpredictable operating delays* related to the operation of actuators, sensors, or reward signals.

Clearly, there is no shortage of challenges. But one hopes that weighed against the rapid development technologies auxiliary to AI (e.g. computing power) and undeniable business potential, those challenges will ultimately contribute to the development of this very interesting field of research.

Studies analysing methods of training agents to effectively act under new circumstances and often in fast-changing environments are very inspiring: they bring together many seemingly distant fields such as algorithmics, optimisation, psychology, cognitive neuroscience, management and decision making. At the same time, many mechanisms developed to improve RL processes could also be successfully applied to everyday life or management. For this

reason, this field of study is certainly worth learning about, even if one were not to pursue a career in the development of autonomous, self-learning systems – the entry barrier may be high, but the effort is still likely to prove worthwhile.

SELF-LEARNING AUTONOMOUS SYSTEMS IN THE MODERN WORLD

Before we proceed to discussing the business potential of self-learning autonomous systems, let us briefly summarise the main points related to the development of intelligent solutions.

The goal of studies on AI is to create systems capable of performing at least as well as their human counterparts. Such solutions are able to operate in a variety of environments, at different levels of predictability and complexity. In order to be effective, they should be provided with a clearly defined function of purpose, capacity for effective perception (of both the environment and internal parameters), build-in or developable behavioural rules and the ability to undertake action influencing the environment.

The advancement of a solution and the related complexity of algorithms are most strongly dependent on the predictability of the environment's behaviour and the agent's own perception capacity. The methods currently employed in AI aim to minimise the impact of such uncertainties. They aid digital agents in reducing the uncertainty related to registration of parameters (e.g. image-, text- and speech-recognition algorithms), environmental dynamics (predictive algorithms), evaluation of actions taken and the consequences thereof (machine learning methods). Additionally, precise motor solutions contribute to the capacity for adequate response, while new methods of gathering experience and forms of presenting knowledge facilitate the process of determining the impact of specific actions in specific environments.

One of the main goals in this context is to provide systems with autonomy. But what does that mean in practice?

The first aspect of autonomy was already discussed in the section focusing on machine learning. The methods described therein were categorised relative to the level of human supervision: from supervised learning to meta-learning. The level of human involvement in the learning process is a natural measure of its autonomy – the best autonomous solutions not only do not require human support but can also select the methods of learning that best suit their respective purposes (*learning to learn*).

Varying levels of autonomy can be observed in the context of Business Intelligence or more generally decision-making systems. One of the methods employed in the classification of different types of business analysis assumes the distinction between *descriptive* (*What happened?*), *diagnostic* (*Why did it happen?*), *predictive* (*What might happen?*), *prescriptive* (*What should*

be done about that?) and *adaptive analyses* (*How do we build a system that would effectively adapt to the environment?*). Let us consider this classification from the perspective of a decision maker. Contemporary managers often make decisions based on descriptive and diagnostic analyses whose forecasts are typically treated as a point of interest (with the possible exception of stock exchange brokers for whom effective prediction is the very basis of decision making). Overall, they rarely base their choices on the results of prescriptive analyses, and hardly ever on adaptive analyses. Meanwhile, even the full inclusion of only predictive analyses in the decision-making processes can significantly change the playing field: let us imagine a board meeting during which the decision makers, rather than consider a course of action dictated by the past, look instead to the future and decide what would be the optimum choices should one assume that the algorithm's predictions are indeed accurate… Naturally, this would go against some deeply ingrained habits but could also bring an entirely new quality to the decision-making table.

Prescriptive analyses go a step further: they recommend the optimum course of action. A system developed by Aera Technologies, which will be discussed later in the book, utilises advanced analytical and predictive systems processing large amounts of data gathered from various other systems operating in an organisation to provide managers with recommendations of future action in many functional areas: from logistics, through production and sales, to finance. It is not difficult to imagine that as such solutions become more effective and flexible; the role of managers may ultimately be reduced to formulating relevant questions, evaluating the quality of recommendations and their ultimate approval, modification or rejection. In turn, this may eventually allow for the materialisation of the so-called *self-driven enterprise* concept. At present, this vision remains somewhat utopian but as such systems continue to dynamically develop, one would be wrong to entirely dismiss it as a viable future possibility.

To a varying extent, autonomous systems have long had a place in industry. In fact, robotics has already been heavily studied in terms of interesting taxonomies of autonomy levels. Beer et al. define robot autonomy as 'the extent to which a robot can sense its environment, plan based on that environment, and act upon that environment with the intent of reaching some task-specific goal (either given to or created by the robot) without external control' (2014, p.77). The authors proposed helpful questions (guidelines) applicable when evaluating the autonomy level of a given robot, e.g. What tasks are to be performed by the robot? What aspects of said tasks are to be included? To what extent are those tasks to be performed independently? Answering the same will help one to determine the *autonomy category of the robot*. In this context, Beer et al. propose ten distinct levels (from minimum to maximum autonomy): *manual operation*, *remote operation*, *assisted remote operation*, *batch*

processing, *decision-making support*, *joint human-supervised control*, *joint robot-supervised control*, *operating supervision*, *supervision* and *full autonomy*. The resulting evaluation of the autonomy level is later used to determine its potential impact on human–robot interaction with due consideration for the perspective of the robot, the human and the social milieu.

Surgical robots constitute a particularly interesting subset of autonomous machines. Ficuciello et al. (2019) identify four levels of autonomy applicable thereto, based on the level of so-called meaningful human control. At level 0, a surgent is in full control of the robot – its role is reduced to increasing the efficiency and precision of surgical operations with the view to compensating for certain sensory and motor limitations of the human operator. At level 1, the robot assists in the surgery – serving the role of a so-called *robotic surgery assistant*. Its primary function, particularly valuable in the context of so-called micro-surgery, is to actively limit and modify the movement trajectory of surgical instruments. At this level, the surgeon has the option of disabling this functionality, thus forcing the choice of the human-defined trajectory. This could be compared to mechanisms evening the movement of pencils or brushes in drawing applications on e.g. a tablet. At level 2, the human operator defines a task and the robot proceeds to autonomously perform it. The surgeon's role is reduced to hands-free monitoring of the task's performance and stepping in to adjust it where necessary – in this case, the robot is under discrete rather than continuous control (as opposed to level 1). Finally, at level 3, the robot independently prepares strategies for the performance of surgical tasks and the role of the human comes down to selecting and approving the optimum strategies and then supervising their execution. Clearly, we are now only a step away from medical devices previously only seen in science fiction movies.

Let us now proceed to autonomous cars. This industry is probably the most advanced in terms of practical application of AI solutions. Hence, quite naturally, it has long developed its own dedicated taxonomy. The Society of Automotive Engineering identified six levels of vehicle autonomy based on a detailed identification of respective human- and vehicle-controlled tasks (cf. SAE 2018).

And so, at levels 0, 1 and 2, the human driver is responsible for controlling the vehicle's movement (acceleration, braking, steering, etc.), even if certain assistance functionalities are engaged. At level 0, the vehicle itself provides little support, mainly in the form of alerts or rapid response, e.g. automatic braking in hazardous situations, blind spot or lane departure warnings. Level 1 provides assistance in terms of steering *or* acceleration/braking, e.g. line centring or adaptive cruise control. Level 2 is similar, although in this case steering *and* acceleration/braking assistance is provided simultaneously.

Higher levels of autonomy are available at levels 3 through 5. Here, the driver is no longer actively involved in the functional processes: only at level

3 is his or her input required when the vehicle 'demands' it. At levels 3 and 4, the vehicle operates autonomously as long as certain conditions are met. Level 5 entails full autonomy of the vehicle, irrespective of conditions: driver input is unnecessary and sometimes even impossible (e.g. if steering implements are not even provided).

It is equally interesting to consider autonomy from the perspective of autonomous aerial vehicles. Derenick (2020) defines drone autonomy as the ability to answer three questions: *Where am I?*, *Where am I going?* and *How do I get there?* Apparently, in the sometimes very difficult conditions that such machines have to cope with, these problems turn out to be anything but trivial. Let us imagine a drone used for mapping out a mine. Firstly, it will have no access to geolocation (e.g. GPS), it will therefore have to rely on advanced analyses of data received from various sensors and maps, often generated in real time, to independently answer the first of the questions: *Where am I?* The answer to the question *Where am I going?* requires planning, which basically entails decomposition of the mission goal into partial tasks, and the latter into elementary behaviours. At the level of behaviours, optimum trajectories are established to help the drone to reach its designated destination. Finally, the question *How do I get there?* requires autonomous performance of the task, without any external supervision, often without viable communication systems. When this is coupled with obvious weight limitations, which also translates to limited computing power (a drone cannot carry a powerful computer), limited energy (battery) and susceptibility to external hazards such as gusts of wind, we can begin to appreciate the considerably more convenient 'situation' of autonomous vehicles.

As follows from the above, the specific definition of autonomy is strongly dependent on the context and application of an intelligent system – nonetheless, it always comes down to a gradual reduction of dependence on the human operator. Still, it is up to the human to decide what competences are to be delegated to the machine. So far, these have been limited to the performance of tasks – soon, however, we might allow machines to also identify goals, and not only particular (e.g. when planning the course of a surgery) but also strategic ones. There is much to suggest that researchers are now gradually laying the groundwork for just that (e.g. inverse RL methods described in Chapter 4). One should remain aware of these trends and be able to recognise the various methods of extracting value from intelligent systems, which will be the topic of the rest of this book.

SUMMARY

At the beginning of this chapter, intelligent systems and technologies were defined as *capable of effectively operating under new circumstances*. A detailed

explanation of the respective components of this definition was then provided, with a particular focus on the ability to respond to fast-changing circumstances characterised by high uncertainty and only partial availability of information. Next, a whole range of machine learning methods were discussed, with a more in-depth consideration of RL, which demonstrated the complexity of issues related to training effective solutions intended for real-world applications.

One could get the impression that advanced machine learning algorithms remain the domain of researchers testing their effectiveness in technical simulations or at laboratories, research centres and industrial facilities. This could not be further from the truth: autonomous robots have long been successfully implemented on production lines, and autonomous cars have already travelled millions of miles on 'real' roads. Indeed, one could hardly think of a better test for an intelligent system (within the meaning of our definition) than a crowded highway travelled at night in heavy sleet. The systems are already so mature that regulators have initiated legislative processes aimed at officially allowing such vehicles to participate in public traffic (cf. reports from the work of the National Highway Traffic Safety Administration in the United States (Reuters 2020)).

Clearly, the vision of intelligent systems *capable of effective operation under new circumstances* is no longer a utopian idea. Their gradual proliferation is commonly expected – which begs the question: *How does one effectively utilise their potential to generate value in one's organisation?* This problem will be the focus of the subsequent chapters of this book.

NOTES

1. It is also possible that while the environment remains unchanged, the task itself evolves – such an environment is described as semi-dynamic.
2. It is noteworthy that the benefits for the customer are also considerable: a vast majority of people actually tend to opt for fewer rather than more options.
3. I.e. a function of state > action that ensures the maximum possible reward yield for the episode.
4. Again, in RL nomenclature feedback is referred to as reward regardless of whether its value is positive, zero or negative.
5. In a general case: the map of state > actions probability distribution.
6. As a result, one cannot talk of RL relying on the trial and error approach. However, this example is often evoked in RL handbooks as it facilitates a fairly simple introduction of certain key concepts, which will later provide the basis for developing learning methods applicable to environments that require exploration.
7. As already mentioned, the strategy entailing the choice of the highest value action in each state (described as greedy) is one of many rather than the only available choice.
8. Technically, it comes down to searching among the available actions for one with the highest value of function Q in the given state.

REFERENCES

Beer, J. M., A. D. Fisk and W. A. Rogers (2014), 'Toward a Framework for Levels of Robot Autonomy in Human–Robot Interaction', *Journal of Human-Robot Interaction*, **3** (2), 74–99.

Bostrom, N. (2014), *Superintelligence*, Oxford: Oxford University Press.

Davis, H. M. (1949), 'Mathematical Machines', *Scientific American*, **April**, 12.

Derenick, J. (2020), 'What Is Autonomy?', *Industrial*, accessed 22 March 2020 at www.exyn.com/2020/01/27/what-is-autonomy/.

Doersch, C., A. Gupta and A. A. Efros (2016), 'Unsupervised Visual Representation Learning by Context Prediction', *ArXiv:1505.05192 [Cs]*, accessed 21 March 2020 at http://arxiv.org/abs/1505.05192.

Dulac-Arnold, G., D. Mankowitz and T. Hester (2019), 'Challenges of Real-World Reinforcement Learning', *ArXiv:1904.12901 [Cs, Stat]*, accessed 19 March 2020 at http://arxiv.org/abs/1904.12901.

Ficuciello, F., G. Tamburrini, A. Arezzo, L. Villani and B. Siciliano (2019), 'Autonomy in Surgical Robots and Its Meaningful Human Control', *Paladyn, Journal of Behavioral Robotics*, **10** (1), 30–43.

Géron, A. (2019), *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems*, Newton, MA: O'Reilly.

Gupta, A., B. Eysenbach, C. Finn and S. Levine (2019), 'Unsupervised Meta-Learning for Reinforcement Learning', *ArXiv:1806.04640 [Cs, Stat]*, accessed 21 March 2020 at http://arxiv.org/abs/1806.04640.

Heidecke, J. (2019), *Inverse Reinforcement Learning Tutorial, Part I*, accessed 20 March 2020 at https://thinkingwires.com/posts/2018-02-13-irl-tutorial-1.html.

Morales, M. (2020), 'Grokking Deep Reinforcement Learning', accessed 6 December 2019 at www.manning.com:443/books/grokking-deep-reinforcement-learning.

Nava, M., J. Guzzi, R. O. Chavez-Garcia, L. M. Gambardella and A. Giusti (2019), 'Learning Long-Range Perception Using Self-Supervision from Short-Range Sensors and Odometry', *ArXiv:1809.07207 [Cs, Stat]*, accessed 21 March 2020 at http://arxiv.org/abs/1809.07207.

Nielsen, M. A. (2015), *Neural Networks and Deep Learning*, accessed 18 March 2020 at http://neuralnetworksanddeeplearning.com.

OpenAI: Spinning Up (2018), accessed 19 March 2020 at https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below.

Pathak, D., P. Agrawal, A. A. Efros and T. Darrell (2017), 'Curiosity-Driven Exploration by Self-Supervised Prediction', *ArXiv:1705.05363 [Cs, Stat]*, accessed 21 March 2020 at http://arxiv.org/abs/1705.05363.

Reuters (2020), 'UPDATE 1: US Agency Proposes Revising Auto Safety Rules to Speed Self-Driving Cars', 17 March, accessed 21 March 2020 at www.reuters.com/article/autos-selfdriving-idUSL1N2BA0WI.

Russell, S. J., P. Norvig and E. Davis (2010), *Artificial Intelligence: A Modern Approach*, 3rd ed., Upper Saddle River, NJ: Prentice Hall.

SAE (2018), *Levels of Driving Automation*, accessed 22 March 2020 at www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles.

Sutton, R. S. and A. G. Barto (2017), *Reinforcement Learning*, 2nd ed., Cambridge, MA: MIT Press.