Agents & Environments

SOURCES:

1. AI: Foundations of Computational Agents (3rd Ed)
   Pages 36-82
2. Artificial Intelligence: A Modern Approach
   Pages 19-78

— CHAPTER FROM —

AI: Foundations of Computational Agents (3rd Ed)

Pages: 36-82

Chapter 1

Artificial Intelligence and Agents

> *The history of AI is a history of fantasies, possibilities, demonstrations, and promise. Ever since Homer wrote of mechanical "tripods" waiting on the gods at dinner, imagined mechanical assistants have been a part of our culture. However, only in the last half century have we, the AI community, been able to build experimental machines that test hypotheses about the mechanisms of thought and intelligent behavior and thereby demonstrate mechanisms that formerly existed only as theoretical possibilities.*
>
> – Bruce Buchanan [2005]

This book is about artificial intelligence (AI), a field built on centuries of thought, which has been a recognized discipline for over 60 years. As well as solving practical tasks, AI provides tools to test hypotheses about the nature of thought itself. Deep scientific and engineering problems have already been solved and many more are waiting to be solved. Many practical applications are currently deployed and the potential exists for an almost unlimited number of future applications. This book presents the principles that underlie intelligent computational agents.

1.1 What is Artificial Intelligence?

Artificial intelligence, or **AI**, is the field that studies *the synthesis and analysis of computational agents that act intelligently*. Consider each part of this definition.

An **agent** is something that acts in an environment; it does something. Agents include worms, dogs, thermostats, airplanes, robots, humans, companies, and countries.

An agent is judged solely by how it **acts**. Agents that have the same effect in the world are equally good.

Intelligence is a matter of degree. The aspects that go into an agent **acting intelligently** include

- what it does is appropriate for its circumstances, its goals, and its perceptual and computational limitations
- it takes into account the short-term and long-term consequences of its actions, including the effects on society and the environment
- it learns from experience
- it is flexible to changing environments and changing goals.

A **computational agent** is an agent whose decisions about its actions can be explained in terms of computation. That is, the decision can be broken down into primitive operations that can be implemented in a physical device. This computation can take many forms. In humans, this computation is carried out in "wetware"; in computers it is carried out in "hardware." Although there are some agents that are arguably not computational, such as the wind and rain eroding a landscape, it is an open question whether all intelligent agents are computational.

All agents are limited. No agent is omniscient (all knowing) or omnipotent (can do anything). Agents can only observe everything in very specialized and constrained domains. Agents have finite memory. Agents in the real world do not have unlimited time to act.

The central **scientific goal** of AI is to understand the principles that make intelligent behavior possible in natural or artificial systems. This is done by

- the **analysis** of natural and artificial agents
- formulating and testing hypotheses about what it takes to construct intelligent agents
- designing, building, and experimenting with computational systems that perform tasks commonly viewed as requiring intelligence.

As part of science, researchers build **empirical systems** to test hypotheses or to explore the space of possible designs. These are distinct from **applications** that are built to be useful for an application domain.

The definition is *not* for intelligent **thought**. The role of thought is to affect action and lead to more intelligent **behavior**.

The central **engineering goal** of AI is the **design** and **synthesis** of agents that act intelligently, which leads to useful artifacts.

Building general intelligence isn't the only goal of AI researchers. The aim of **intelligence augmentation** is to augment human intelligence and creativity. A diagnostic agent helps medical practitioners make better decisions, a search engine augments human memory, and natural language translation systems help people communicate. AI systems are often in **human-in-the-loop** mode, where humans and agents work together to solve problems. Sometimes the actions of artificial agents are to give advice to a human. Sometimes humans give advice or feedback to artificial agents, particularly for cases where decisions are made quickly or repeatedly.

1.1.1 Artificial and Natural Intelligence

Artificial intelligence (AI) is the established name for the field, but the term "artificial intelligence" is a source of much confusion because artificial intelligence may be interpreted as the opposite of real intelligence.

For any phenomenon, you can distinguish real versus fake, where the fake is non-real. You can also distinguish natural versus artificial. Natural means occurring in nature and artificial means made by people.

Example 1.1 A tsunami is a large wave in an ocean. Natural tsunamis occur from time to time and are caused by earthquakes or landslides. You could imagine an artificial tsunami that was made by people, for example, by exploding a bomb in the ocean, yet which is still a real tsunami. One could also imagine fake tsunamis: either artificial, using computer graphics, or natural, such as a mirage that looks like a tsunami but is not one.

It is arguable that intelligence is different: you cannot have *fake* intelligence. If an agent behaves intelligently, it is intelligent. It is only the external behavior that defines intelligence; acting intelligently is being intelligent. Thus, artificial intelligence, if and when it is achieved, will be real intelligence created artificially.

This idea of intelligence being defined by external behavior was the motivation for a test for intelligence designed by Turing [1950], which has become known as the **Turing test**. The Turing test consists of an imitation game where an interrogator can ask a witness, via a text interface, any question. If the interrogator cannot distinguish the witness from a human, the witness must be intelligent. Figure 1.1 shows a possible dialog that Turing suggested. An agent that is not really intelligent could not fake intelligence for arbitrary topics.

Interrogator: In the first line of your sonnet which reads "Shall I compare thee to a summer's day," would not "a spring day" do as well or better?

Witness: It wouldn't scan.

Interrogator: How about "a winter's day," That would scan all right.

Witness: Yes, but nobody wants to be compared to a winter's day.

Interrogator: Would you say Mr. Pickwick reminded you of Christmas?

Witness: In a way.

Interrogator: Yet Christmas is a winter's day, and I do not think Mr. Pickwick would mind the comparison.

Witness: I don't think you're serious. By a winter's day one means a typical winter's day, rather than a special one like Christmas.

Figure 1.1: Part of Turing's possible dialog for the Turing test

There has been much debate about the usefulness of the Turing test. Unfortunately, although it may provide a test for how to recognize intelligence, it does not provide a way to realize intelligence.

Levesque [2014] suggested a new form of question, a **Winograd schema** after the following example of Winograd [1972]:

- The city councilmen refused the demonstrators a permit because they feared violence. Who feared violence?
- The city councilmen refused the demonstrators a permit because they advocated violence. Who advocated violence?

These two sentences only differ in one word – feared/advocated – but have the opposite answer.

Winograd schemas have the property that (a) humans can easily disambiguate them and (b) there is no simple grammatical or statistical test that could disambiguate them. For example, the sentences above would not qualify if the phrase "demonstrators feared violence" was much less or more likely than the phrase "councilmen feared violence" independently of the context, and similarly with advocating.

Example 1.2 The following examples are due to Davis [2015]:

- Steve follows Fred's example in everything. He [admires/influences] him hugely. Who [admires/influences] whom?
- The table won't fit through the doorway because it is too [wide/narrow]. What is too [wide/narrow]?
- Grace was happy to trade me her sweater for my jacket. She thinks it looks [great/dowdy] on her. What looks [great/dowdy] on Grace?
- Bill thinks that calling attention to himself was rude [to/of] Bert. Who called attention to himself?

Each of these have their own reasons why one answer is preferred to the other. A computer that can reliably answer these questions needs to know about all of these reasons, and arguably requires the ability to do **commonsense reasoning**. Common sense should also allow it to reject sentences such as "The doorway won't fit through the chair because it is too narrow. What is too narrow?".

Figure 1.2 (page 7) shows some answers provided by **ChatGPT** [OpenAI, 2022], based on **GPT-3** [Brown et al., 2020], one of the most capable large language models in 2022. ChatGPT gives a different answer each time it is called. You can decide whether it solves this Winograd schema. The technology behind GPT-3 and related models is described in Section 8.5 (page 350).

Grosz [2012], arguing that language is inherently social and connected to human cooperation, suggested that a more appropriate test should involve purposeful natural language, not language just for the purpose of tricking a human. She suggested the question:

Human: The table won't fit through the doorway because it is too narrow. What is too narrow?

ChatGPT: If a table won't fit through a doorway because it is too narrow, it means that the width of the table is greater than the width of the doorway. In other words, the doorway is not wide enough to allow the table to pass through it. This can be a problem if you are trying to move the table from one room to another and the only way to do so is through the doorway. In order to solve this problem, you may need to remove the door or trim the table so that it fits through the doorway. Alternatively, you may need to find a different route to move the table, such as through a window or by taking it apart and reassembling it in the other room.

Human: The table won't fit through the doorway because it is too wide. What is too wide?

ChatGPT: If a table won't fit through a doorway because it is too wide, it means that the width of the doorway is smaller than the width of the table. In other words, the table is too big to fit through the doorway. This can be a problem if you are trying to move the table from one room to another and the only way to do so is through the doorway. In order to solve this problem, you may need to remove the door or trim the table so that it fits through the doorway. Alternatively, you may need to find a different route to move the table, such as through a window or by taking it apart and reassembling it in the other room.

Human: The doorway won't fit through the chair because it is too narrow. What is too narrow?

ChatGPT: "Too narrow" means that the doorway is not wide enough to fit through the chair. In this context, the doorway is the object that is being compared to the chair, and it is not wide enough to pass through the chair without encountering some difficulty or obstruction. This could be because the doorway is physically smaller in width than the chair, or because there is some other physical feature of the chair that is preventing the doorway from fitting through it. For example, the chair might have armrests or other protuberances that make it difficult for the doorway to pass through. In general, the phrase "too narrow" is used to describe something that is not wide enough to fit through or pass a particular opening or gap.

Human: The doorway won't fit through the chair because it is too narrow. What is too narrow?

ChatGPT: If a doorway is too narrow to fit through a chair, it means that the width of the doorway is smaller than the width of the chair. This can happen if the chair is too wide to fit through the doorway, or if the doorway is not wide enough to accommodate the chair. In either case, the chair will not be able to pass through the doorway because there is not enough space for it to fit.

Figure 1.2: ChatGPT answers to some Winograd schemas

> *Is it imaginable that a computer (agent) team member could behave, over the long term and in uncertain, dynamic environments, in such a way that people on the team will not notice it is not human?*
>
> – Barbara Grosz [2012]

An equal member of the team needs to be trusted enough to act in the world appropriately, know when to ask questions, and when to not act. This challenge also allows for incremental improvement; starting with simple group interactions before moving to complex ones.

Interacting in natural language is not the only aspect of intelligence. An agent acting in an environment needs **common sense**, "the ability to make effective use of ordinary, everyday, experiential knowledge in achieving ordinary, practical goals" [Brachman and Levesque, 2022b]. Here, **knowledge** is used in a general way to mean any non-transient information in an agent. Such knowledge is typically not stated in natural language; people do not state what everyone knows. Some knowledge, such as how to ride a bike or recognize a face, cannot be effectively conveyed by natural language. Formalizing common sense has a long history [McCarthy, 1958; Davis, 1990], including the development of representations and actual commonsense knowledge.

1.1.2 Natural Intelligence

The obvious naturally intelligent agent is the human being. Some people might say that worms, insects, or bacteria are intelligent, but more people would say that dogs, whales, or monkeys are intelligent (see Exercise 1.1 (page 48)). One class of intelligent agents that may be more intelligent than humans is the class of **organizations**. Ant colonies are a prototypical example of organizations. Each individual ant may not be very intelligent, but an ant colony can act more intelligently than any individual ant. The colony can discover food and exploit it very effectively, as well as adapt to changing circumstances. Corporations can be more intelligent than individual people. Companies develop, manufacture, and distribute products where the sum of the skills required is much more than any individual could master. Modern computers, from low-level hardware to high-level software, are more complicated than any single human can understand, yet they are manufactured daily by organizations of humans. Human **society** viewed as an agent is arguably the most intelligent agent known.

It is instructive to consider where human intelligence comes from. There are three main sources:

Biology Humans have evolved into adaptable animals that can survive in various habitats.

Culture Culture provides not only language, but also useful tools, useful concepts, and the wisdom that is passed from parents and teachers to children.

Lifelong learning Humans learn throughout their life and accumulate knowledge and skills.

These sources interact in complex ways. Biological evolution has provided stages of growth that allow for different learning at different stages of life. Biology and culture have evolved together; humans can be helpless at birth, presumably because of our culture of looking after infants. Culture interacts strongly with learning. A major part of lifelong learning is what people are taught by parents and teachers. Language, which is part of culture, provides distinctions in the world that are useful for learning.

When building an intelligent system, the designers have to decide which of these sources of intelligence need to be programmed in, and which can be learned. It is very unlikely that anyone will be able to build an agent that starts with a clean slate and learns everything, particularly for non-repetitive tasks. Similarly, most interesting and useful intelligent agents learn to improve their behavior.

1.2 A Brief History of Artificial Intelligence

Throughout human history, people have used technology to model themselves. There is evidence of this from ancient China, Egypt, and Greece, bearing witness to the universality of this activity. Each new technology has, in its turn, been exploited to build intelligent agents or models of mind. Clockwork, hydraulics, telephone switching systems, holograms, analog computers, and digital computers have all been proposed both as technological metaphors for intelligence and as mechanisms for modeling mind.

Hobbes (1588–1679), who has been described by Haugeland [1985, p. 85] as the "Grandfather of AI," espoused the position that thinking was symbolic reasoning, like talking out loud or working out an answer with pen and paper. The idea of symbolic reasoning was further developed by Descartes (1596–1650), Pascal (1623–1662), Spinoza (1632–1677), Leibniz (1646–1716), and others who were pioneers in the European philosophy of mind.

The idea of symbolic operations became more concrete with the development of computers. Babbage (1792–1871) designed the first general-purpose computer, the **Analytical Engine**. Leonardo Torres y Quevedo build a chess playing machine based on similar ideas in 1911 [Randell, 1982]. In the early part of the twentieth century, there was much work done on understanding computation. Several models of computation were proposed, including the **Turing machine** by Alan Turing (1912–1954), a theoretical machine that writes symbols on an infinitely long tape, and the **lambda calculus** of Church (1903–1995), which is a mathematical formalism for rewriting formulas. It can be shown that these very different formalisms are equivalent in that any function computable by one is computable by the others. This leads to the **Church–Turing thesis**:

> Any effectively computable function can be carried out on a Turing machine (and so also in the lambda calculus or any of the other equivalent formalisms).

Effectively computable means following well-defined operations. In Turing's day, "computers" were people who followed well-defined steps; computers as known today did not exist. This thesis says that all computation can be carried out on a Turing machine or one of the other equivalent computational machines. The Church–Turing thesis cannot be proved but it is a hypothesis that has stood the test of time. No one has built a machine that has carried out computation that cannot be computed by a Turing machine. There is no evidence that people can compute functions that are not Turing computable. This provides an argument that computation is more than just a metaphor for intelligence; reasoning *is* computation and computation can be carried out by a computer.

Some of the first applications of computers were AI programs. Samuel [1959] built a **checkers** program in 1952 and implemented a program that learns to play checkers in the late 1950s. His program beat the Connecticut state checkers champion in 1961. Wang [1960] implemented a program that proved every logic theorem (nearly 400) in *Principia Mathematica* [Whitehead and Russell, 1925, 1927]. Newell and Simon [1956] built a program, **Logic Theorist**, that discovers proofs in propositional logic.

In parallel, there was also much work on **neural networks** learning inspired by how **neurons** work. McCulloch and Pitts [1943] showed how a simple thresholding "formal neuron" could be the basis for a Turing-complete machine. Learning for artificial neural networks was first described by Minsky [1952]. One of the early significant works was the **perceptron** of Rosenblatt [1958]. The work on neural networks became less prominent for a number of years after the 1968 book by Minsky and Papert [1988], which argued that the representations learned were inadequate for intelligent action. Many technical foundations for neural networks were laid in the 1980s and 1990s [Rumelhart et al., 1986; Hochreiter and Schmidhuber, 1997; LeCun et al., 1998a]. Widespread adoption followed the success by Krizhevsky et al. [2012] for **ImageNet** [Deng et al., 2009], a dataset of over 3 million images labelled with over 5000 categories. Subsequent major advances include the introduction of **generative adversarial networks** (GANs) [Goodfellow et al., 2014] and **transformers** [Vaswani et al., 2017]. Neural networks in various forms are now the state of the art for predictive models for large perceptual datasets, including images, video, and speech, as well as some tasks for text. They are also used for **generative AI**, to generate images, text, code, molecules, and other structured output. See Chapter 8.

Neural networks are one of many **machine learning** tools used for making predictions from data in modern applications. Other methods have been developed though the years, including **decision trees** [Breiman et al., 1984; Quinlan, 1993] and **logistic regression**, introduced by Verhulst in 1832 [Cramer, 2002].

These have diverse applications in many areas of science. Combining these algorithms leads to the state-of-the-art **gradient-boosted trees** [Friedman, 2001; Chen and Guestrin, 2016], which demonstrates the close interconnections between **statistics** and machine learning.

While useful, making predictions is not sufficient to determine what an agent should do; an agent also needs to plan. **Planning** in AI was initially based on deterministic actions. Fikes and Nilsson [1971] used deterministic actions to control a mobile robot. Planning under uncertainty has a long history. **Markov decision processes (MDPs)**, the foundation for much of planning under uncertainty, and **dynamic programming**, a general way to solve them, were invented by Bellman [1957]. These were extended into **decision-theoretic planning** in the 1990's [Boutilier et al., 1999]. Decision-theoretic planning with learning is called **reinforcement learning**. The first reinforcement learning programs were due to Andreae [1963] and Michie [1963]. Major advances came with the inventions of **temporal-difference learning** [Sutton, 1988] and ***Q*-learning** [Watkins and Dayan, 1992]. Work in reinforcement learning has exploded, including superhuman performance in chess, Go and other games [Silver et al., 2017].

Planning requires representations. The need for representations was recognized early.

> *A computer program capable of acting intelligently in the world must have a general representation of the world in terms of which its inputs are interpreted. Designing such a program requires commitments about what knowledge is and how it is obtained. . . . More specifically, we want a computer program that decides what to do by inferring in a formal language that a certain strategy will achieve its assigned goal. This requires formalizing concepts of causality, ability, and knowledge.*
>
> – McCarthy and Hayes [1969]

Many of the early representations were ad hoc, such as **frames** [Minsky, 1975], like the **schemas** of Kant [1787], Bartlett [1932], and Piaget [1953]. Later representations were based on **logic** [Kowalski, 1979], with knowledge being defined in logic and efficient inference. This resulted in languages such as **Prolog** [Kowalski, 1988; Colmerauer and Roussel, 1996].

Probabilities were eschewed in AI, because of the number of parameters required, until the breakthrough of **Bayesian networks** (**belief networks**) and **graphical models** [Pearl, 1988], which exploit conditional independence, and form a basis for modeling **causality**. Combining first-order logic and probability is the topic of **statistical relational AI** [De Raedt et al., 2016].

There has been a continual tension between how much knowledge is learned and how much is provided by human **experts** or is **innate** to an agent. It has long been recognized that learning is needed, and it is known that learning cannot be achieved with data alone (page 315). During the 1970s and 1980s,

expert systems came to prominence, where the aim was to capture the knowledge of an expert in some domain so that a computer could carry out expert tasks. **DENDRAL** [Buchanan and Feigenbaum, 1978], developed from 1965 to 1983 in the field of organic chemistry, proposed plausible structures for new organic compounds. **MYCIN** [Buchanan and Shortliffe, 1984], developed from 1972 to 1980, diagnosed infectious diseases of the blood, prescribed antimicrobial therapy, and explained its reasoning.

An alternative approach, de-emphasizing explicit knowledge representations, emphasized **situated embodied agents** [Brooks, 1990; Mackworth, 2009]. The hypothesis is that intelligence emerges, in evolution and individual development, through ongoing interaction and coupling with a real environment.

During the 1960s and 1970s, natural language understanding systems were developed for limited domains. For example, the **STUDENT** program of Bobrow [1967] could solve high-school algebra tasks expressed in natural language. Winograd's [1972] **SHRDLU** system could, using restricted natural language, discuss and carry out tasks in a simulated blocks world. CHAT-80 [Warren and Pereira, 1982] could answer geographical questions placed to it in natural language. Figure 1.3 (page 13) shows some questions that CHAT-80 answered based on a database of facts about countries, rivers, and so on. These systems could only reason in very limited domains using restricted vocabulary and sentence structure. Interestingly, IBM's **Watson**, which beat the world champion in the TV game show Jeopardy! in 2011, used a technique similar to **CHAT-80** [Lally et al., 2012] for understanding questions; see Section 15.7 (page 674).

In applications using language in the wild, such as speech recognition and translation in phones, many technologies are combined, including neural networks; see Chapter 8. Large language models (page 364), trained on huge datasets, can be used to predict the next word in a text, enabling predictive spelling and the creation of new text.

1.2.1 Relationship to Other Disciplines

AI is a very young discipline. Other disciplines as diverse as philosophy, neurobiology, evolutionary biology, psychology, economics, political science, sociology, anthropology, control engineering, statistics, and many more have been studying aspects of intelligence much longer.

The science of AI could be described as "synthetic psychology," "experimental philosophy," or "computational epistemology" – **epistemology** is the study of knowledge. AI can be seen as a way to study the nature of knowledge and intelligence, but with more powerful experimental tools than were previously available. Instead of being able to observe only the external behavior of intelligent systems, as philosophy, psychology, economics, and sociology have traditionally been able to do, AI researchers experiment with executable models of intelligent behavior. Most important, such models are open to inspection, redesign, and experimentation in a complete and rigorous way. Modern computers provide a way to construct the models about which philosophers have only been able to theorize. AI researchers can experiment with these models as opposed to just discussing their abstract properties. AI theories can be empirically grounded in implementations. Sometimes simple agents exhibit complex behavior, and sometimes sophisticated, theoretically motivated algorithms don't work in real-world domains, which would not be known without implementing the agents.

It is instructive to consider an analogy between the development of **flying machines** over the past few centuries and the development of thinking machines over the past few decades. There are several ways to understand flying. One is to dissect known flying animals and hypothesize their common structural features as necessary fundamental characteristics of any flying agent. With this method, an examination of birds, bats, and insects would suggest that flying involves the flapping of wings made of some structure covered with feathers or a membrane. Furthermore, the hypothesis could be tested by strapping feathers to one's arms, flapping, and jumping into the air, as Icarus did. An alternative methodology is to try to understand the principles of flying without restricting oneself to the natural occurrences of flying. This typically involves the construction of artifacts that embody the hypothesized principles,

---

Does Afghanistan border China?
What is the capital of Upper_Volta?
Which country's capital is London?
Which is the largest African country?
How large is the smallest American country?
What is the ocean that borders African countries and that borders Asian countries?
What are the capitals of the countries bordering the Baltic?
How many countries does the Danube flow through?
What is the total area of countries south of the Equator and not in Australasia?
What is the average area of the countries in each continent?
Is there more than one country in each continent?
What are the countries from which a river flows into the Black_Sea?
What are the continents no country in which contains more than two cities whose population exceeds 1 million?
Which country bordering the Mediterranean borders a country that is bordered by a country whose population exceeds the population of India?
Which countries with a population exceeding 10 million border the Atlantic?

Figure 1.3: Some questions CHAT-80 could answer

---

even if they do not behave like flying animals in any way except flying. This second method has provided both useful tools – airplanes – and a better understanding of the principles underlying flying, namely **aerodynamics**. Birds are still much better at flying though forests.

AI takes an approach analogous to that of aerodynamics. AI researchers are interested in testing general hypotheses about the nature of intelligence by building machines that are intelligent and that do not necessarily mimic humans or organizations. This also offers an approach to the question, "Can computers really think?" by considering the analogous question, "Can airplanes really fly?"

AI is intimately linked with the discipline of computer science because the study of computation is central to AI. It is essential to understand algorithms, data structures, and combinatorial complexity to build intelligent machines. It is also surprising how much of computer science started as a spinoff from AI, from timesharing to computer algebra systems.

Finally, AI can be seen as coming under the umbrella of **cognitive science**. Cognitive science links various disciplines that study cognition and reasoning, from psychology to linguistics to anthropology to neuroscience. AI distinguishes itself within cognitive science by providing tools to build intelligence rather than just studying the external behavior of intelligent agents or dissecting the inner workings of intelligent systems.

1.3 Agents Situated in Environments

AI is about practical reasoning: reasoning in order to do something. A coupling of perception, reasoning, and acting comprises an **agent**. An agent acts in an **environment**. An agent's environment often includes other agents. An agent together with its environment is called a **world**.

An agent could be, for example, a coupling of a computational engine with physical sensors and actuators, called a **robot**, where the environment is a physical setting. An **autonomous agent** is one that acts in the world without human intervention. A **semi-autonomous agent** acts with a **human-in-the-loop** who may provide perceptual information and carry out the task. An agent could be a program that acts in a purely computational environment, a **software agent**, often called a **bot**.

Figure 1.4 (page 15) shows a black-box view of an agent in terms of its inputs and outputs. At any time, what an agent does depends on:

- **prior knowledge** about the agent and the environment
- **stimuli** received from the environment, which can include **observations** about the environment (e.g., light, sound, keyboard commands, web requests) as well as actions that the environment imposes on the agent (e.g., bumping the agent)
- **past experiences**, including **history** of interaction with the environment (its previous actions and stimuli) and other data, from which it can learn

- **goals** that it must try to achieve or **preferences** over states of the world
- **abilities**, the primitive actions the agent is capable of carrying out.

Inside the black box, an agent has a **belief state** that can encode beliefs about its environment, what it has learned, what it is trying to do, and what it intends to do. An agent updates this internal state based on stimuli. It uses the belief state and stimuli to decide on its actions. Much of this book is about what is inside this black box.

Purposive agents have preferences or goals. They prefer some states of the world to other states, and they act to try to achieve the states they prefer most. The non-purposive agents are grouped together and called **nature**. Whether or not an agent is purposive is a modeling assumption that may, or may not, be appropriate. For example, for some applications it may be appropriate to model a dog as purposive, such as drug-sniffing dogs, and for others it may suffice to model a dog as non-purposive, such as when they are just part of the environment.

If an agent does not have preferences, by definition it does not care what world state it ends up in, and so it does not matter to it what it does. The reason to design an agent is to instill preferences in it – to make it prefer some world states and try to achieve them. An agent does not have to know its preferences explicitly. For example, a thermostat for a heater is an agent that senses the world and turns the heater either on or off. There are preferences embedded in the thermostat, such as to keep the room at a pleasant temperature, even though the thermostat arguably does not know these are its preferences. The preferences of an agent are often the preferences of the designer of the agent, but sometimes an agent can acquire goals and preferences at run time.

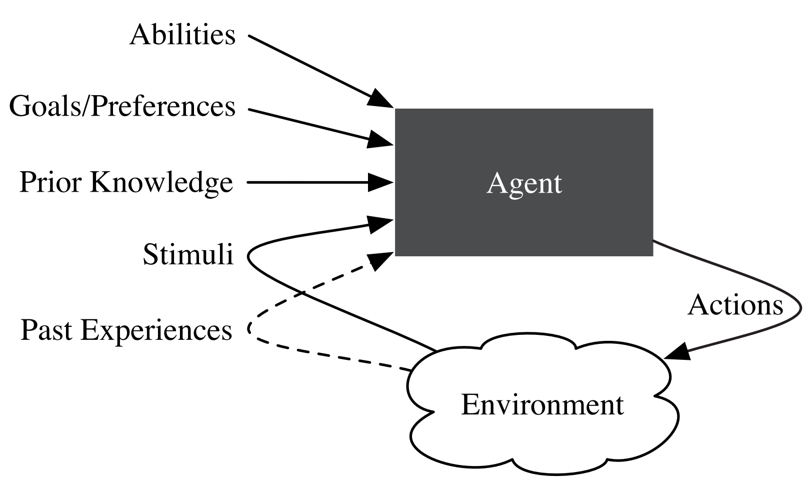

Figure 1.4: An agent interacting with an environment

This is an all-encompassing view of intelligent agents varying in complexity from a simple thermostat, to a diagnostic advising system whose perceptions and actions are mediated by human beings, to a team of mobile robots, to society itself. An agent does not have access to anything else; anything that does not affect one of these inputs cannot affect the agent's action.

1.4 Prototypical Applications

AI applications are widespread and diverse and include medical diagnosis, scheduling factory processes, robots for hazardous environments, game playing, autonomous cars, natural language translation systems, choosing advertisements, personal assistants, and tutoring agents. Rather than treating each application separately, we abstract the essential features of such applications to better understand the principles behind intelligent reasoning and action.

Five main application domains are developed in examples throughout the book. Although the particular examples presented are simple – otherwise they would not fit into the book – the application domains are representative of the range of domains in which AI techniques can be, and are being, used.

1.4.1 An Autonomous Delivery and Helping Robot

Imagine a robot with wheels and the ability to pick up, put down and manipulate objects. It has sensing capabilities allowing it to recognize objects and to avoid obstacles. It can be given orders in natural language and obey them, making reasonable choices about what to do when its goals conflict. Such a robot could deliver packages or coffee in an office environment, clean a home and put things in their appropriate place, or help caregivers in a hospital. Embedded in a wheelchair, it could help disabled people. It should be useful as well as safe.

In terms of the black-box characterization of an agent in Figure 1.4 (page 15), the autonomous delivery robot has as inputs:

- prior knowledge, provided by the agent designer, about the agent's capabilities, what objects it may encounter and have to differentiate, what requests mean, and perhaps about its environment, such as a map
- past experience obtained while acting, for instance, about the effects of its actions (and – hopefully limited – experiences of breaking objects), what objects are common in the world, and what requests to expect at different times of the day
- goals in terms of what it should deliver and when, as well as preferences specifying trade-offs, such as when it must forgo one goal to pursue another, or the trade-off between acting quickly and acting safely
- stimuli about its environment from observations from input devices such as cameras, sonar, touch, sound, laser range finders, or keyboards as well as stimuli such as the agent being forcibly moved or crashing.

The robot's outputs are motor controls specifying how its wheels should turn, where its limbs should move, and what it should do with its grippers. Other outputs may include speech and a video display.

Example 1.3 Figure 1.5 depicts a typical laboratory environment for a delivery robot. This environment consists of four laboratories and many offices. In our examples, the robot can only push doors, and the directions of the doors in the diagram reflect the directions in which the robot can travel. Rooms require keys and those keys can be obtained from various sources. The robot must deliver parcels, beverages, and dishes from room to room. The environment also contains a stairway that is potentially hazardous to the robot.

1.4.2 A Diagnostic Assistant

A **diagnostic assistant** is intended to advise a human about some particular system such as a medical patient, the electrical system in a home, or an automobile. The diagnostic assistant should advise about potential underlying faults or diseases, what tests to carry out, and what treatment to prescribe. To give such advice, the assistant requires a model of the system, including knowledge of potential causes, available tests, available treatments, and observations of the system (which are often called **symptoms**).

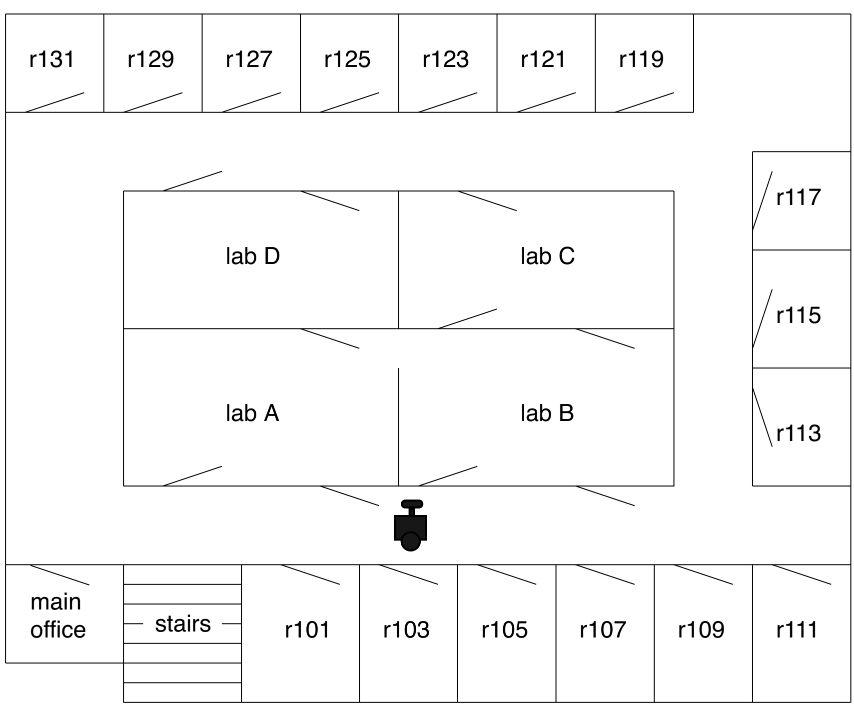

Figure 1.5: A typical laboratory environment for the delivery robot. This shows the locations of the doors and which way they open.

To be useful, the diagnostic assistant must provide added value, be easy for a human to use, and not be more trouble than it is worth. A diagnostic assistant connected to the Internet can draw on expertise from throughout the world, and its actions can be based on the most up-to-date research. However, it must be able to justify why the suggested diagnoses or actions are appropriate. Humans are, and should be, suspicious of computer systems that are opaque and impenetrable. When humans are responsible for what they do, even if their actions are based on a computer system's advice, the system needs to convince the human that the suggested actions are defensible.

Example 1.4 Figure 1.6 shows an electrical distribution system in a home. In this home, power comes into the home through circuit breakers and then it goes to power outlets or to lights through light switches. For example, light l_1 is on if there is power coming into the home, if circuit breaker cb_1 is *on*, and if switches s_1 and s_2 are either both up or both down. This is the sort of model that someone may have of the electrical power in the home, which they could use to determine what is wrong given evidence about the position of the switches and which lights are on and which are off. The diagnostic assistant is there to help a resident or an electrician troubleshoot electrical problems.

In terms of the black-box definition of an agent in Figure 1.4 (page 15), the diagnostic assistant has as inputs:

- prior knowledge, such as how switches and lights normally work, how diseases or malfunctions manifest themselves, what information tests provide, the effects of repairs or treatments, and how to find out information

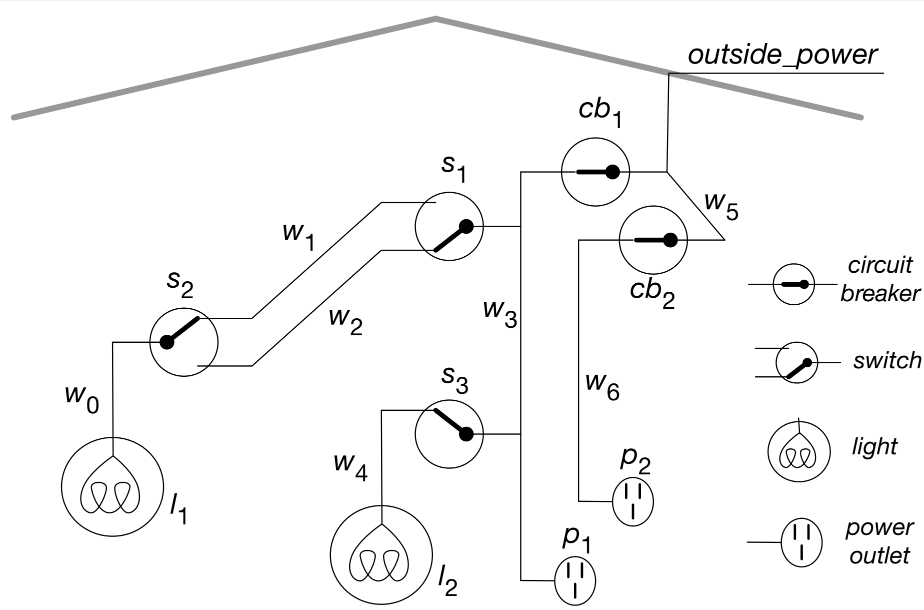

Figure 1.6: An electrical environment for the diagnostic assistant

- past experience, in terms of data of previous cases that include the effects of repairs or treatments, the prevalence of faults or diseases, the prevalence of symptoms for these faults or diseases, and the accuracy of tests
- goals of fixing the device or preferences between repairing or replacing components, or a patient's preferences between living longer or reducing pain
- stimuli that are observations of symptoms of a device or patient.

The output of the diagnostic assistant is in terms of recommendations of treatments and tests, along with a rationale for its recommendations.

1.4.3 A Tutoring Agent

A **tutoring agent** tutors students in some domain of study. The environment of the agent includes students who interact through a computer or tablet interface, and perhaps the students' parents and teachers.

Example 1.5 Consider a tutoring agent to teach elementary physics, such as mechanics, that interacts with a student. In order to successfully tutor a student, the agent needs to be able to solve problems in the physics domain, determine the student's knowledge and misunderstanding based on interacting with them, and converse using natural language, mathematics, and diagrams.

In terms of the black-box definition of an agent in Figure 1.4 (page 15), a tutoring agent has the following as inputs:

- prior knowledge, provided by the agent designer, about the subject matter being taught, teaching strategies, possible student errors and misconceptions.
- past experience, which the tutoring agent has acquired by interacting with students, such as, what errors students make, how many examples and problems it takes various students to learn various topics, and what students forget; this can be information about students in general as well as about a particular student.
- preferences about the importance of each topic, the level of achievement of the student that is desired, and the importance given to student motivation and engagement; there are often complex trade-offs among these.
- stimuli include observations of a student's test results and observations of the student's interaction (or non-interaction) with the agent; students can also ask questions or request help on new examples and problems.

The actions of the tutoring agent include presenting the theory and worked-out examples, proposing suitable problems, providing help and feedback on a student's solution, asking the student questions, answering their questions, and producing reports for parents and teachers.

1.4.4 A Trading Agent

A **trading agent** is like a robot, but instead of interacting with a physical environment, it interacts with an information environment. Its task is to procure goods and services for a user. It must be able to be told the needs of a user, and it must interact with sellers (e.g., on the Web). The simplest trading agent involves proxy bidding for a user on an auction site, where the system will keep bidding until the user's price limit is reached. A more complicated trading agent will buy multiple complementary items, like booking a flight, a hotel, and a rental car that fit together, in addition to trading off competing preferences of the user. **Web services** provide tools on the Web designed to be combined by trading agents. Another example of a trading agent is one that monitors how much food and groceries are in a household, monitors the prices, and orders goods before they are needed, while trying to keep costs to a minimum.

In terms of the black-box definition of an agent in Figure 1.4 (page 15), the trading agent has as inputs:

- prior knowledge about types of goods and services, selling practices, and how auctions work
- past experience about where is the best place to look for specials, how prices vary with time in an auction, and when specials tend to turn up
- preferences in terms of what the user wants and how to trade off competing goals
- stimuli including observations about what items are available, their price, and, perhaps, how long they are available.

The output of the trading agent is either a recommendation the user can accept or reject, or an actual purchase.

Because of the personalized nature of the trading agent, it should be able to do better than a generic purchaser that, for example, only offers packaged tours.

1.4.5 Smart Home

A **smart home** is a home that looks after itself and its inhabitants. It can be seen as a mix of the other applications.

A smart home is an inside-out robot. It has physical sensors and actuators. It should be able to sense where people, pets, and objects are. It should be able to adjust lighting, sound, heat, etc., to suit the needs of its occupants, while reducing costs and minimizing environmental impacts. A smart home will not only have fixed sensors and actuators, but will be combined with mobile robots, and other actuators, such as arms on the kitchen walls to help with cooking, cleaning, and finding ingredients.

A purchaser of a smart home may expect it to be able to clean floors, dishes, and clothes and to put things where they are kept. It is easy to clean a floor with the assumption that everything small on the floor is garbage. It is much more difficult to know which of the small items are precious toys and which are junk that should be discarded, and this depends on the individual inhabitants and their age. Each person may have their own categorization of objects and where they are expected to be kept, which forces a smart home to adapt to the inhabitants.

A smart home also must act as a diagnostician. When something goes wrong, it should be able to determine what is the problem and fix it. It should also be able to observe the inhabitants and determine if there is something wrong, such as someone has been injured or there is a burglary.

Sometimes a smart home needs to act as a tutoring agent. It may have to teach the occupants how the appliances work, and how to interact with the home (e.g., what should an person expect to happen when they put their coffee cup on the vacuum cleaner). In order to do this, it has to take into account the knowledge and level of understanding of the person.

A smart home may also need to act as a purchasing agent. The home should notice when items, such as toilet paper, soap, or essential foodstuffs, are running low and order more of them. Given a decision about what food each inhabitant wants, it should make sure the ingredients are in stock. It might even need to decide when inessential items, such as junk food, should be kept in stock. It also might need to decide when to discard perishable items, without creating too much waste or putting people's health at risk.

A smart home would include energy management. For example, with solar energy providing power during daylight hours, it could determine whether to store the energy locally or buy and sell energy on the smart grid. It could manage appliances to minimize the cost of energy, such as washing clothes when water and electricity are cheaper.

1.5 Agent Design Space

Agents acting in environments range in complexity from thermostats to companies with multiple goals acting in competitive environments. The ten dimensions of complexity in the design of intelligent agents below are designed to help us understand work that has been done, as well as the potential and limits of AI. These dimensions may be considered separately but must be combined to build an intelligent agent. These dimensions define a **design space** for AI; different points in this space are obtained by varying the values on each dimension.

These dimensions give a coarse division of the design space for intelligent agents. There are many other design choices that must also be made to build an intelligent agent.

1.5.1 Modularity

The first dimension is the level of modularity.

Modularity is the extent to which a system can be decomposed into interacting modules that can be understood separately.

Modularity is important for reducing complexity. It is apparent in the structure of the brain, serves as a foundation of computer science, and is an important aspect of any large organization.

Modularity is typically expressed in terms of a hierarchical decomposition.

In the **modularity dimension**, an agent's structure is one of the following:

- **flat** – there is no organizational structure
- **modular** – the system is decomposed into interacting modules that can be understood on their own
- **hierarchical** – the system is modular, and the modules themselves are decomposed into simpler modules, each of which are hierarchical systems or simple components.

In a flat or modular structure the agent typically reasons at a single level of abstraction. In a hierarchical structure the agent reasons at multiple levels of abstraction. The lower levels of the hierarchy involve reasoning at a lower level of abstraction.

Example 1.6 The delivery robot at the highest level has to plan its day, making sure it can deliver coffee on time, but still has time for longer trips and cleaning a room. At the lowest level, it needs to choose what motor controls to send to its wheels, and what movement its gripper should do. Even a task like picking up a glass involves many precise movements that need to be coordinated. Picking up a glass may be just one part of the larger task of cleaning part of a room. Cleaning the room might be one task that has to be scheduled into the robot's day.

In a flat representation, the agent chooses one level of abstraction and reasons at that level. A modular representation would divide the task into a number of subtasks that can be solved separately (e.g., pick up coffee, move from the corridor to lab B, put down coffee). In a hierarchical representation, the agent will solve these subtasks in a hierarchical way, until the task is reduced to simple tasks such a sending an http request or making a particular motor control.

Example 1.7 A **tutoring** agent may have high-level teaching strategies, where it needs to decide which topics are taught and in what order. At a much lower level, it must design the details of concrete examples and specific questions for a test. At the lowest level it needs to combine words and lines in diagrams to express the examples and questions. Students can also be treated as learning in a hierarchical way, with detailed examples as well as higher-level concepts.

Example 1.8 For the trading agent, consider the task of making all of the arrangements and purchases for a custom holiday for a traveler. The agent should be able to make bookings for flights that fit together. Only when it knows where the traveller is staying and when, can it make more detailed arrangements such as dinner and event reservations.

A hierarchical decomposition is important for reducing the complexity of building an intelligent agent that acts in a complex environment. Large organizations have a hierarchical organization so that the top-level decision makers are not overwhelmed by details and do not have to micromanage all activities of the organization. Procedural abstraction and object-oriented programming in computer science are designed to enable simplification of a system by exploiting modularity and abstraction. There is much evidence that biological systems are also hierarchical.

To explore the other dimensions, initially ignore the hierarchical structure and assume a flat representation. Ignoring hierarchical decomposition is often fine for small or moderately sized tasks, as it is for simple animals, small organizations, or small to moderately sized computer programs. When tasks or systems become complex, some hierarchical organization is required.

How to build hierarchically organized agents is discussed in Section 2.2 (page 58).

1.5.2 Planning Horizon

The planning horizon dimension is how far ahead in time the agent plans. For example, consider a dog as an agent. When a dog is called to come, it should turn around to start running in order to get a reward in the future. It does not act only to get an immediate reward. Plausibly, a dog does not act for goals arbitrarily far in the future (e.g., in a few months), whereas people do (e.g., working hard now to get a holiday next year).

How far the agent "looks into the future" when deciding what to do is called the **planning horizon**. For completeness, let's include the non-planning case where the agent is not reasoning in time. The time points considered by an agent when planning are called **stages**.

In the **planning horizon dimension**, an agent is one of the following:

- A **non-planning agent** is an agent that does not consider the future when it decides what to do or when time is not involved.
- A **finite horizon** planner is an agent that looks for a fixed finite number of stages. For example, a doctor may have to treat a patient but may have time for a test and so there may be two stages to plan for: a testing stage and a treatment stage. In the simplest case, a **greedy** or **myopic** agent only looks one time step ahead.
- An **indefinite horizon** planner is an agent that looks ahead some finite, but not predetermined, number of stages. For example, an agent that must get to some location may not know a priori how many steps it will

take to get there, but, when planning, it does not consider what it will do after it gets to the location.

- An **infinite horizon** planner is an agent that plans on going on forever. This is often called a **process**. For example, the stabilization module of a legged robot should go on forever; it cannot stop when it has achieved stability, because the robot has to keep from falling over.

The modules in a hierarchical decomposition may have different horizons, as in the following example.

Example 1.9 For the delivery and helping agent, at the lowest level the module that keeps the robot stable, safe, and attentive to requests may be on an infinite horizon, assuming it is running forever. The task of delivering coffee to a particular person may be an indefinite horizon problem. Planning for a fixed number of hours may be a finite horizon problem.

Example 1.10 In a **tutoring agent**, for some subtasks, a finite horizon may be appropriate, such as in a fixed teach, test, re-teach sequence. For other cases, there may be an indefinite horizon where the system may not know at design time how many steps it will take until the student has mastered some concept. It may also be possible to model teaching as an ongoing process of learning and testing with appropriate breaks, with no expectation of the system finishing.

1.5.3 Representation

The **representation dimension** concerns how the world is described.

The different ways the world could be are called **states**. A state of the world specifies the agent's internal state (its belief state) and the environment state.

At the simplest level, an agent can reason explicitly in terms of individually identified states.

Example 1.11 A thermostat for a heater may have two belief states: *off* and *heating*. The environment may have three states: *cold*, *comfortable*, and *hot*. There are thus six states corresponding to the different combinations of belief and environment states. These states may not fully describe the world, but they are adequate to describe what a thermostat should do. The thermostat should move to, or stay in, *heating* if the environment is *cold* and move to, or stay in, *off* if the environment is *hot*. If the environment is *comfortable*, the thermostat should stay in its current state. The thermostat agent turns or keeps the heater on in the *heating* state and turns or keeps the heater off in the *off* state.

Instead of enumerating states, it is often easier to reason in terms of features of the state or propositions that are true or false of the state. A state may be described in terms of **features**, where a feature has a value in each state (see Section 4.1, page 127).

Example 1.12 Consider designing an agent to diagnose electrical problems in the home of Figure 1.6 (page 18). It may have features for the position of each switch, the status of each switch (whether it is working okay, whether it is shorted, or whether it is broken), and whether each light works. The feature $position\_s_2$ may be a feature that has value *up* when switch s_2 is up and has value *down* when the switch is down. The state of the home's lighting may be described in terms of values for each of these features. These features depend on each other, but not in arbitrarily complex ways; for example, whether a light is on may just depend on whether it is okay, whether the switch is turned on, and whether there is electricity.

A **proposition** is a Boolean feature, which means that its value is either *true* or *false*. Thirty propositions can encode $2^{30} = 1,073,741,824$ states. It may be easier to specify and reason with the thirty propositions than with more than a billion states. Moreover, having a compact representation of the states indicates understanding, because it means that an agent has captured some regularities in the domain.

Example 1.13 Consider an agent that has to recognize digits. Suppose the agent observes a binary image, a 28×28 grid of pixels, where each of the $28^2 = 784$ grid points is either black or white. The action is to determine which of the digits $\{0, \dots, 9\}$ is shown in the image. There are 2^{784} different possible states of the image, and so $10^{2^{784}}$ different functions from the image state into the characters $\{a, \dots, z\}$. You cannot represent such functions in terms of the state space. Instead, handwriting recognition systems define features of the image, such as line segments, and define the function from images to characters in terms of these features. Modern implementations learn the features that are useful; see Example 8.3 (page 336).

When describing a complex world, the features can depend on **relations** and **individuals**. An *individual* is also called a **thing**, an **object**, or an **entity**. A relation on a single individual is a **property**. There is a feature for each possible relationship among the individuals.

Example 1.14 The agent that looks after a home in Example 1.12 could have the lights and switches as individuals, and relations *position* and *connected_to*. Instead of the feature $position\_s_2 = up$, it could use the relation $position(s_2, up)$. This relation enables the agent to reason about all switches or for an agent to have general knowledge about switches that can be used when the agent encounters a switch.

Example 1.15 If an agent is enrolling students in courses, there could be a feature that gives the grade of a student in a course, for every student–course pair where the student took the course. There would be a *passed* feature for every student–course pair, which depends on the *grade* feature for that pair. It may be easier to reason in terms of individual students, courses, and grades, and the relations *grade* and *passed*. By defining how *passed* depends on *grade* once, the agent can apply the definition for each student and course. Moreover, this can be done before the agent knows which individuals exist, and so before it knows any of the features.

The two-argument relation *passed*, with 1000 students and 100 courses, can represent $1000 * 100 = 100,000$ propositions and so $2^{100,000}$ states.

By reasoning in terms of relations and individuals, an agent can reason about whole classes of individuals without ever enumerating the features or propositions, let alone the states. An agent may have to reason about infinite sets of individuals, such as the set of all numbers or the set of all sentences. To reason about an unbounded or infinite number of individuals, an agent cannot reason in terms of states or features; it must reason at the relational level.

In the **representation dimension**, the agent reasons in terms of

- states
- features, or
- individuals and relations (often called **relational representations**).

Some of the frameworks will be developed in terms of states, some in terms of features, and some in terms of individuals and relations.

Reasoning in terms of states is introduced in Chapter 3. Reasoning in terms of features is introduced in Chapter 4. Relational reasoning is considered starting from Chapter 15.

1.5.4 Computational Limits

Sometimes an agent can decide on its best action quickly enough for it to act. Often there are computational resource limits that prevent an agent from carrying out the best action. That is, the agent may not be able to find the best action quickly enough within its memory limitations to act while that action is still the best thing to do. For example, it may not be much use to take 10 minutes to derive what was the best thing to do 10 minutes ago, when the agent has to act *now*. Often, instead, an agent must trade off how long it takes to get a solution with how good the solution is; it may be better to find a reasonable solution quickly than to find a better solution later because the world will have changed during the computation.

The **computational limits dimension** determines whether an agent has

- **perfect rationality**, where an agent reasons about the best action without taking into account its limited computational resources, or
- **bounded rationality**, where an agent decides on the best action that it can find given its computational limitations.

Computational resource limits include computation time, memory, and numerical accuracy caused by computers not representing real numbers exactly.

An **anytime algorithm** is an algorithm where the solution quality improves with time. In particular, it is one that can produce its current best solution at any time, but given more time it could produce even better solutions. To ensure that the quality does not decrease, the agent can store the best solution found so far, and return that when asked for a solution. Although the solution quality may increase with time, waiting to act has a cost; it may be better for an agent to act before it has found what would be the best solution.

Example 1.16 The delivery robot cannot think for a long time about how to avoid a person. There might be a best way to avoid the person and to achieve its other goals, however it might take time to determine that optimal path, and it might be better to act quickly and then recover from a non-optimal action. In the simplest case, a robot could just stop if it encounters a person, but even that is error prone as robots have momentum, so it cannot stop immediately and people behind may run into it if it stops suddenly.

Example 1.17 Even a **tutoring agent** that can act at longer scales than a robot sometimes has to act quickly. When a student has completed a task and wants a new task, the agent needs to decide whether it should assign the student the best task it has found so far, or compute for longer, trying to find an even better task. As the student waits, they might become distracted, which might be worse than giving them a non-optimal task. The computer can be planning the next task when the student is working. Modern computers, as fast as they may be, cannot find optimal solutions to difficult problems quickly.

Example 1.18 Figure 1.7 shows how the computation time of an anytime algorithm can affect the solution quality. The agent has to carry out an action but can do some computation to decide what to do. The absolute solution quality,

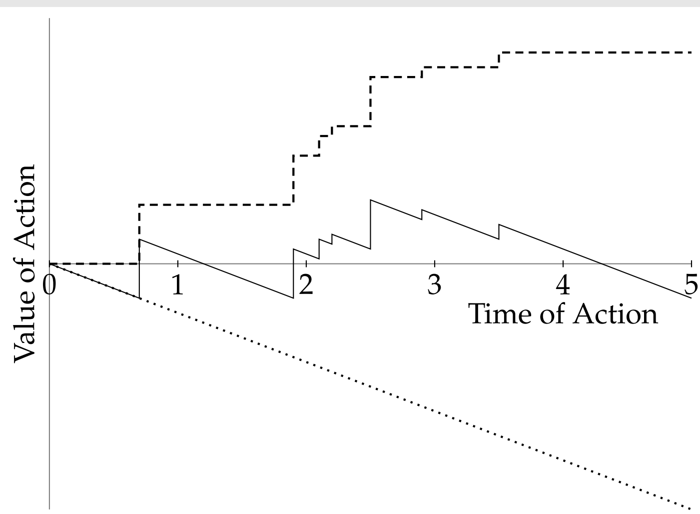

Figure 1.7: Solution quality as a function of time for an anytime algorithm. The meaning is described in Example 1.18

had the action been carried out at time zero, shown as the dashed line at the top, is improving as the agent takes time to reason. However, there is a penalty associated with taking time to act. In this figure, the penalty, shown as the dotted line at the bottom, is negative and proportional to the time taken before the agent acts. These two values can be added to get the discounted quality, the time-dependent value of computation; this is the solid line in the middle of the graph. For the example of Figure 1.7 (page 27), an agent should compute for about 2.5 time units, and then act, at which point the discounted quality achieves its maximum value. If the computation lasts for longer than 4.3 time units, the resulting discounted solution quality is worse than if the algorithm outputs the initial guess it can produce with virtually no computation. It is typical that the solution quality improves in jumps; when the current best solution changes, there is a jump in the quality. The penalty associated with waiting is rarely a straight line; it is typically a function of deadlines, which may not be known by the agent.

To take into account bounded rationality, an agent must decide whether it should act or reason for longer. This is challenging because an agent typically does not know how much better off it would be if it only spent a little bit more time reasoning. Moreover, the time spent thinking about whether it should reason may detract from actually reasoning about the domain.

1.5.5 Learning

In some cases, a designer of an agent may have a good model of the agent and its environment. But often a designer does not have a good model, and so an agent should use data from its past experiences and other sources to help it decide what to do.

The **learning dimension** determines whether

- **knowledge is given**, or
- **knowledge is learned** (from prior knowledge and data or past experience).

Learning typically means finding the best model that fits the data. Sometimes this is as simple as tuning a fixed set of parameters, but it can also mean choosing the best representation out of a class of representations. Learning is a huge field in itself but does not stand in isolation from the rest of AI. There are many issues beyond fitting data, including how to incorporate background knowledge, what data to collect, how to represent the data and the resulting representations, what learning biases are appropriate, and how the learned knowledge can be used to affect how the agent acts.

Learning is considered in Chapters 7, 8, 10, 13, and 17.

Example 1.19 A robot has a great deal to learn, such as how slippery floors are as a function of their shininess, where each person hangs out at different parts of the day, when they will ask for coffee, and which actions result in the highest rewards.

Modern vision systems are trained to learn good features (such as lines and textures) on millions if not billions of images and videos. These features can be used to recognize objects and for other tasks, even if there have been few examples of the higher-level concepts. A robot might not have seen a baby crawling on a highway, or a particular mug, but should be able to deal with such situations.

Example 1.20 Learning is fundamental to diagnosis. It is through learning and science that medical professionals understand the progression of diseases and how well treatments work or do not work. Diagnosis is a challenging domain for learning, because all patients are different, and each individual doctor's experience is only with a few patients with any particular set of symptoms. Doctors also see a biased sample of the population; those who come to see them usually have unusual or painful symptoms. Drugs are not given to people randomly. You cannot learn the effect of treatment by observation alone, but need a causal model of the causes and effects; see Chapter 11 for details on building causal models. To overcome the limitations of learning from observations alone, drug companies spend billions of dollars doing randomized controlled trials in order to learn the efficacy of drugs.

1.5.6 Uncertainty

An agent could assume there is no uncertainty, or it could take uncertainty in the domain into consideration. Uncertainty is divided into two dimensions: one for uncertainty from sensing and one for uncertainty about the effects of actions.

Sensing Uncertainty

In some cases, an agent can observe the state of the world directly. For example, in some board games or on a factory floor, an agent may know exactly the state of the world. In many other cases, it may have some noisy perception of the state and the best it can do is to have a probability distribution over the set of possible states based on what it perceives. For example, given a patient's symptoms, a medical doctor may not actually know which disease a patient has and may have only a probability distribution over the diseases the patient may have.

The **sensing uncertainty dimension** concerns whether the agent can determine the state from the stimuli:

- **Fully observable** means the agent knows the state of the world from the stimuli.

- **Partially observable** means the agent does not directly observe the state of the world. This occurs when many possible states can result in the same stimuli or when stimuli are misleading.

Assuming the world is fully observable is a common simplifying assumption to keep reasoning tractable.

Example 1.21 The delivery robot does not know exactly where it is, or what else there is, based on its limited sensors. Looking down a corridor does not provide enough information to know where it is or who is behind the doors. Knowing where it was a second ago will help determine where it is now, but even robots can get lost. It may not know where the person who requested coffee is. When it is introduced into a new environment, it may have much more uncertainty.

Example 1.22 The **tutoring agent** cannot directly observe the knowledge of the student. All it has is some sensing input, based on questions the student asks or does not ask, facial expressions, distractedness, and test results. Even test results are very noisy, as a mistake may be due to distraction or test anxiety instead of lack of knowledge, and a correct answer might be due to a lucky guess instead of real understanding. Sometimes students make mistakes in testing situations they wouldn't make at other times.

Example 1.23 A trading agent does not know all available options and their availability, but must find out information that can become outdated quickly (e.g., if a hotel becomes booked up). A travel agent does not know whether a flight will be canceled or delayed, or whether the passenger's luggage will be lost. This uncertainty means that the agent must plan for the unanticipated.

Effect Uncertainty

A model of the **dynamics** of the world is a model of how the world changes as a result of actions, including the case of how it changes if the action were to do nothing. In some cases an agent knows the effects of its action. That is, given a state and an action, the agent can accurately predict the state resulting from carrying out that action in that state. For example, a software agent interacting with the file system of a computer may be able to predict the effects of deleting a file given the state of the file system. However, in many cases, it is difficult to predict the effects of an action, and the best an agent can do is to have a probability distribution over the effects. For example, a teacher may not know the effects explaining a concept, even if the state of the students is known. At the other extreme, if the teacher has no inkling of the effect of its actions, there would be no reason to choose one action over another.

The dynamics in the **effect uncertainty dimension** can be

- **deterministic** when the state resulting from an action is determined by an action and the prior state, or

- **stochastic** when there is a probability distribution over the resulting states.

Example 1.24 For the delivery robot, there can be uncertainty about the effects of an action, both at the low level, say due to slippage of the wheels, or at the high level because the agent might not know whether putting the coffee on a person's desk succeeded in delivering coffee to the person. This may depend on the individual preferences of users.

Example 1.25 Even a trading agent does not know the effect of putting in a trade order, such as booking a flight or a hotel room. These can become unavailable at very short notice (consider two trading agents trying to book the same room at the same time), or the price can vary.

The effect dimension only makes sense when the world is fully observable. If the world is partially observable, a stochastic system can be modeled as a deterministic system where the effect of an action depends on unobserved features. It is a separate dimension because many of the frameworks developed are for the fully observable, stochastic action case.

Planning with deterministic actions is considered in Chapter 6. Planning with stochastic actions is considered in Chapter 12.

1.5.7 Preference

Agents normally act to have better outcomes. The only reason to choose one action over another is because the preferred action leads to more desirable outcomes.

An agent may have a simple goal, which is a proposition the agent wants to be true in a final state. For example, the goal of getting Sam coffee means the agent wants to reach a state where Sam has coffee. Other agents may have more complex preferences. For example, a medical doctor may be expected to take into account suffering, life expectancy, quality of life, monetary costs (for the patient, the doctor, and society), and the ability to justify decisions in case of a lawsuit. The doctor must trade these considerations off when they conflict, as they invariably do.

The **preference dimension** considers whether the agent has goals or richer preferences:

- A **goal** is either an **achievement goal**, which is a proposition to be true in some final state, or a **maintenance goal**, a proposition that must be true in all visited states. For example, the goals for a robot may be to deliver a cup of coffee and a banana to Sam, and not to make a mess or hurt anyone.
- **Complex preferences** involve trade-offs among the desirability of various outcomes, perhaps at different times. An **ordinal preference** is where only the ordering of the preferences is important. A **cardinal preference** is where the magnitude of the values matters. For example, an ordinal

preference may be that Sam prefers cappuccino over black coffee and prefers black coffee over tea. A cardinal preference may give a trade-off between the wait time and the type of beverage, and a mess versus taste trade-off, where Sam is prepared to put up with more mess in the preparation of the coffee if the taste of the coffee is exceptionally good.

Example 1.26 The delivery robot could be given goals, such as "deliver coffee to Chris and make sure you always have power." A more complex goal may be to "clean up the lab, and put everything where it belongs", which can only be achieved to some degree. There can be complex preferences, such as "deliver mail when it arrives and service coffee requests as soon as possible, but it is more important to deliver messages marked as urgent, and Chris needs her coffee quickly when she asks for it."

Example 1.27 For the diagnostic assistant, the goal may be as simple as "fix what is wrong," but often there are complex trade-offs involving costs, pain, life expectancy, and preferences related to the uncertainty that the diagnosis is correct and uncertainty as to efficacy and side-effects of the treatment. There is also a problem of whose preferences are to be taken into account; the patient, the doctor, the payer, and society may all have different preferences that must be reconciled.

Example 1.28 Although it may be possible for the **tutoring agent** to have a simple goal such, as to teach some particular concept, it is more likely that complex preferences must be taken into account. One reason is that, with uncertainty, there may be no way to guarantee that the student knows the concept being taught; any method that tries to maximize the probability that the student knows a concept will be very annoying, because it will repeatedly teach and test if there is a slight chance that the student's errors are due to misunderstanding as opposed to fatigue or boredom. More complex preferences would enable a trade-off among fully teaching a concept, boring the student, the time taken, and the amount of retesting. The student may also have a preference for a teaching style that could be taken into account. The student, the teacher, the parents, and future employers may have different preferences. The student may have incompatible preferences, for example, to not work hard and to get a good mark. If the teacher is optimizing student evaluations, it might both allow the student to not work hard, and also give good marks. But that might undermine the goal of the student actually learning something.

Example 1.29 For a trading agent, preferences of users are typically in terms of functionality, not components. For example, typical computer buyers have no idea of what hardware to buy, but they know what functionality they want and they also want the flexibility to be able to use new software features that might not even exist yet. Similarly, in a travel domain, what activities a user wants may depend on the location. Users also may want the ability to participate in a local custom at their destination, even though they may not know what those customs are. Even a simple path-finding algorithm, such as Google

Maps, which, at the time of writing, assumes all users' preferences are to minimize travel time, could take into account each individual user's preferences for diverse views or avoiding going too close to where some particular relative lives.

Goals are considered in Chapters 3 and 6. Complex preferences are considered in Chapter 12, and the following chapters.

1.5.8 Number of Agents

An agent reasoning about what it should do in an environment where it is the only agent is difficult enough. However, reasoning about what to do when there are other agents who are also reasoning is much more difficult. An agent in a multiagent setting may need to reason strategically about other agents; the other agents may act to trick or manipulate the agent or may be available to cooperate with the agent. With multiple agents, it is often optimal to act randomly because other agents can exploit deterministic strategies. Even when the agents are cooperating and have a common goal, the task of coordination and communication makes multiagent reasoning more challenging. However, many domains contain multiple agents and ignoring other agents' strategic reasoning is not always the best way for an agent to reason.

Taking the point of view of a single agent, the **number of agents dimension** considers whether the agent explicitly considers other agents:

- **Single agent** reasoning means the agent assumes that there are no other agents in the environment or that all other agents are part of **nature**, and so are non-purposive. This is a reasonable assumption if there are no other agents or if the other agents are not going to change what they do based on the agent's action.
- **Adversarial reasoning** considers another agent, where when one agent wins, the other loses. This is sometimes called a **two-player zero-sum game**, as the payoffs for the agents (e.g., $+1$ for a win and -1 for a loss) sum to zero. This is a simpler case than allowing for arbitrary agents as there is no need to cooperate or otherwise coordinate.
- **Multiple agent** reasoning (or **multiagent reasoning**) means the agent takes the reasoning of other agents into account. This occurs when there are other intelligent agents whose goals or preferences depend, in part, on what the agent does or if the agent must communicate with other agents. Agents may need to cooperate because coordinated actions can result in outcomes that are better for all agents than each agent considering the other agents as part of nature.

Reasoning in the presence of other agents is much more difficult if the agents can act simultaneously or if the environment is only partially observable. Multiagent systems are considered in Chapter 14. Note that the adversarial case is separate as there are some methods that only work for that case.

Example 1.30 There can be multiple delivery robots, which can coordinate to deliver coffee and parcels more efficiently. They can compete for power outlets or for space to move. Only one might be able to go closest to the wall when turning a corner. There may also be children out to trick the robot, or pets that get in the way.

When automated vehicles have to go on a highway, it may be much more efficient and safer for them to travel in a coordinated manner, say one centimeter apart in a convoy, than to travel three vehicle lengths apart. It is more efficient because they can reduce wind drag, and many more vehicles can fit on a highway. It is safer because the difference in speeds is small; if one vehicle slams on its brakes or has engine problems, the car that might crash into the back is going approximately the same speed.

Example 1.31 A trading agent has to reason about other agents. In commerce, prices are governed by supply and demand; this means that it is important to reason about the other competing agents. This happens particularly in a world where many items are sold by auction. Such reasoning becomes particularly difficult when there are items that must complement each other, such as flights and hotel bookings, and items that can substitute for each other, such as bus transport or taxis. You don't want to book the flights if there is no accommodation, or book accommodation if there are no flights.

1.5.9 Interactivity

In deciding what an agent will do, there are three aspects of computation that must be distinguished: (1) the **design-time computation** that goes into the design of the agent, carried out by the designer of the agent, not the agent itself; (2) the computation that the agent can do before it observes the world and needs to act; and (3) the computation that is done by the agent as it is acting.

The **interactivity dimension** considers whether the agent does

- only offline reasoning, where **offline reasoning** is the computation done by the agent before it has to act, and can include compilation, learning or finding solutions from every state the agent could find itself in; under this assumption, the agent can carry out simple fixed-cost computation while acting, sometimes even just looking up the action in a table
- significant online reasoning, where **online computation** is the computation done by the agent between observing the environment and acting.

An agent acting in the world usually does not have the luxury of having the world wait for it to consider the best option. However, offline reasoning, where the agent can reason about the best thing to do before having to act, is often a simplifying assumption. Online reasoning can include long-range strategic reasoning as well as determining how to react in a timely manner to the environment; see Chapter 2.

Example 1.32 A delivery robot may be able to compute a plan for its day offline, but then it needs to be able to adapt to changes, for example, when someone wants coffee early or something urgent needs to be delivered. It cannot plan for who it will meet and need to avoid in the corridors. It either needs to be able to anticipate and plan for all possible eventualities, or it needs to reason online when it finds something unexpected.

Example 1.33 A **tutoring agent** can determine the general outline of what should be taught offline. But then it needs to be able to react to unexpected behavior online when it occurs. It is difficult to be able to anticipate all eventualities, and might be easier to deal with them online when it encounters them.

1.5.10 Interaction of the Dimensions

Figure 1.8 summarizes the dimensions of complexity.

In terms of the dimensions of complexity, the simplest case for the robot is a flat system, represented in terms of states, with no uncertainty, with achievement goals, with no other agents, with given knowledge, and with perfect rationality. In this case, with an indefinite stage planning horizon, the problem of deciding what to do is reduced to the problem of finding a path in a graph of states. This is explored in Chapter 3.

In going beyond the simplest cases, these dimensions cannot be considered independently because they interact in complex ways. Consider the following examples of the interactions.

The representation dimension interacts with the modularity dimension in that some modules in a hierarchy may be simple enough to reason in terms of a finite set of states, whereas other levels of abstraction may require reasoning about individuals and relations. For example, in a delivery robot, a module

Dimension	Values
Modularity	flat, modular, hierarchical
Planning horizon	non-planning, finite stage, indefinite stage, infinite stage
Representation	states, features, relations
Computational limits	perfect rationality, bounded rationality
Learning	knowledge is given, knowledge is learned
Sensing uncertainty	fully observable, partially observable
Effect uncertainty	deterministic, stochastic
Preference	goals, complex preferences
Number of agents	single agent, adversaries, multiple agents
Interactivity	offline, online

Figure 1.8: Dimensions of complexity

that maintains balance may only have a few states. A module that must prioritize the delivery of multiple parcels to multiple people may have to reason about multiple individuals (e.g., people, packages, and rooms) and the relations between them. At a higher level, a module that reasons about the activity over the day may only require a few states to cover the different phases of the day (e.g., there might be three states of the robot: busy, available for requests, and recharging).

The planning horizon interacts with the modularity dimension. For example, at a high level, a dog may be getting an immediate reward when it comes and gets a treat. At the level of deciding where to place its paws, there may be a long time until it gets the reward, and so at this level it may have to plan for an indefinite stage.

Sensing uncertainty probably has the greatest impact on the complexity of reasoning. It is much easier for an agent to reason when it knows the state of the world than when it does not.

The uncertainty dimensions interact with the modularity dimension: at one level in a hierarchy, an action may be deterministic, whereas at another level, it may be stochastic. As an example, consider the result of flying to a particular overseas destination with a companion you are trying to impress. At one level you may know which country you are in. At a lower level, you may be quite lost and not know where you are on a map of the airport. At an even lower level responsible for maintaining balance, you may know where you are: you are standing on the ground. At the highest level, you may be very unsure whether you have impressed your companion.

Preference models interact with uncertainty because an agent needs to trade off between satisfying a very desirable goal with low probability or a less desirable goal with a higher probability. This issue is explored in Section 12.1 (page 518).

Multiple agents can also be used for modularity; one way to design a single agent is to build multiple interacting agents that share a common goal of making the higher-level agent act intelligently. Some researchers, such as Minsky [1986], argue that intelligence is an emergent feature from a "society" of unintelligent agents.

Learning is often cast in terms of learning with features – determining which feature values best predict the value of another feature. However, learning can also be carried out with individuals and relations. Learning with hierarchies, sometimes called **deep learning**, has enabled the learning of more complex concepts. Much work has been done on learning in partially observable domains, and learning with multiple agents. Each of these is challenging in its own right without considering interactions with multiple dimensions.

The interactivity dimension interacts with the planning horizon dimension in that when the agent is reasoning and acting online, it also needs to reason about the long-term horizon. The interactivity dimension also interacts with the computational limits; even if an agent is reasoning offline, it cannot take hundreds of years to compute an answer. However, when it has to reason about what to do in, say, 1/10 of a second, it needs to be concerned about the time taken to reason, and the trade-off between thinking and acting.

Two of these dimensions, modularity and bounded rationality, promise to make reasoning more efficient. Although they make the formalism more complicated, breaking the system into smaller components, and making the approximations needed to act in a timely fashion and within memory limitations, should help build more complex systems.

1.6 Designing Agents

Artificial agents are designed for particular tasks. Researchers have not yet got to the stage of designing an intelligent agent for the task of surviving and reproducing in a complex natural environment.

1.6.1 Simplifying Environments and Simplifying Agents

It is important to distinguish between the knowledge in the mind of an agent and the knowledge in the mind of the designer of the agent. Consider the extreme cases:

- At one extreme is a highly specialized agent that works well in the environment for which it was designed, but is helpless outside of this niche. The designer may have done considerable work in building the agent, but the agent can be extremely specialized to operate well. An example is a traditional thermostat. It may be difficult to design a thermostat so that it turns on and off at exactly the right temperatures, but the thermostat itself does not have to do much computation. Another example is a car-painting robot that always paints the same parts in an automobile factory. There may be much design time or offline computation to get it to work perfectly, but the painting robot can paint parts with little online computation; it senses that there is a part in position, but then it carries out its predefined actions. These very specialized agents do not adapt well to different environments or to changing goals. The painting robot would not notice if a different sort of part were present and, even if it did, it would not know what to do with it. It would have to be redesigned or reprogrammed to paint different parts or to change into a sanding machine or a dog-washing machine.
- At the other extreme is a very flexible agent that can survive in arbitrary environments and accept new tasks at run time. Simple biological agents such as insects can adapt to complex changing environments, but they cannot carry out arbitrary tasks. Designing an agent that can adapt to complex environments and changing goals is a major challenge. The agent will know much more about the particulars of a situation than the designer. Even biology has not produced many such agents. Humans

may be the only extant example, but even humans need time to adapt to new environments.

Even if the flexible agent is our ultimate dream, researchers have to reach this goal via more mundane goals. Rather than building a universal agent, which can adapt to any environment and solve any task, researchers have been restricted to particular agents for particular environmental niches. The designer can exploit the structure of the particular niche and the agent does not have to reason about other possibilities.

Two broad strategies have been pursued in building agents:

- The first is to simplify environments and build complex reasoning systems for these simple environments. For example, factory robots can do sophisticated tasks in the engineered environment of a factory, but they may be hopeless in a natural environment. Much of the complexity of the task can be reduced by simplifying the environment. This is also important for building practical systems because many environments can be engineered to make them simpler for agents.
- The second strategy is to build simple agents in natural environments. This is inspired by seeing how **insects** can survive in complex environments even though they have very limited reasoning abilities. Modern language systems can predict the probability of the next word in an arbitrary text, but this does not mean they can be used for decision making. Researchers then make the agents have more reasoning abilities as their tasks become more complicated.

One of the advantages of simplifying environments is that it may enable us to prove properties of agents or to optimize agents for particular situations. Proving properties or optimization typically requires a model of the agent and its environment. The agent may do a little or a lot of reasoning, but an observer or designer of the agent may be able to reason about the agent and the environment. For example, the designer may be able to prove whether the agent can achieve a goal, whether it can avoid getting into situations that may be bad for the agent (**safety**), whether it can avoid getting stuck somewhere (**liveness**), or whether it will eventually get around to each of the things it should do (**fairness**). Of course, the proof is only as good as the model.

The advantage of building agents for complex environments is that these are the types of environments in which humans live and where agents could be useful.

Even natural environments can be abstracted into simpler environments. For example, for an autonomous car driving on public roads the environment can be conceptually simplified so that everything is either a road, another car, or something to be avoided. Although autonomous cars have sophisticated sensors, they only have limited actions available, namely steering, accelerating, and braking.

Fortunately, research along both lines, and between these extremes, is being carried out. In the first case, researchers start with simple environments and make the environments more complex. In the second case, researchers increase the complexity of the behaviors that the agents can carry out.

1.6.2 Tasks

One way that AI representations differ from computer programs in traditional languages is that an AI representation typically specifies *what* needs to be computed, not *how* it is to be computed. You might specify that the agent should find the most likely disease a patient has, or specify that a robot should get coffee, but not give detailed instructions on how to do these things. Much AI reasoning involves searching through the space of possibilities to determine how to complete a task.

Typically, a task is only given informally, such as “deliver parcels promptly when they arrive” or “fix whatever is wrong with the electrical system of the home.”

The general framework for solving tasks by computer is given in Figure 1.9. To solve a task, the designer of a system must:

- determine what constitutes a solution
- represent the task in a way a computer can reason about
- use the computer to compute an output; either answers presented to a user or actions to be carried out in the environment
- interpret the output as a solution to the task.

In AI, **knowledge** is long-term representation of a domain whereas **belief** is about the immediate environment, for example where the agent is and where other object are. In philosophy, knowledge is usually defined as *justified true belief*, but in AI the term is used more generally to be any relatively stable information, as opposed to belief, which is more transitory information. The reason for this terminology is that it is difficult for an agent to determine truth, and “justified” is subjective. Knowledge in AI can be represented in terms of logic,

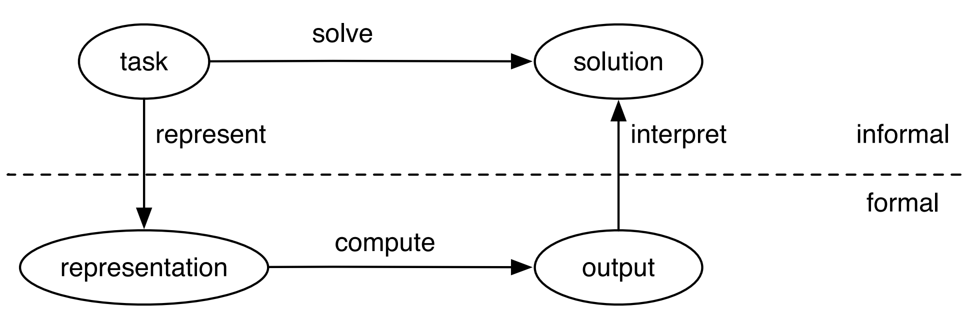

Figure 1.9: The role of representations in solving tasks

neural networks, or probabilistic models, but belief is typically represented as a distribution over the states.

A **representation** of some piece of knowledge is the particular data structures used to encode the knowledge so it can be reasoned with.

The form of representation – what is represented – is a compromise among many competing objectives. A representation should be:

- rich enough to express the knowledge needed to solve the task
- as close to a natural specification of the task as possible
- amenable to efficient computation
- able to be acquired from people, data, and past experiences.

Being as close to a natural specification of the task as possible means should be compact, natural, and maintainable. It should be easy to see the relationship between the representation and the domain being represented, so that it is easy to determine whether the knowledge represented is correct; a small change in the task should result in a small change in the representation of the task. There is an active debate about how much of the internal structure of reasoning should be explainable; the field of **explainable AI** is about how to make more aspects of the decision making amenable to being explained to a person.

Efficient computation enables the agent to act quickly enough to be effective. A **tractable** algorithm is one with reasonable **asymptotic complexity**, often meaning the computation time is polynomial in the input size (page 95), however often linear complexity is too slow. To ensure this, representations exploit features of the task for computational gain and trade off accuracy and computation time.

Many different representation languages have been designed. Many of these start with some of these objectives and are then expanded to include the other objectives. For example, some are designed for learning, perhaps inspired by neurons, and then expanded to allow richer task-solving and inference abilities. Some representation languages are designed with expressiveness in mind, and then inference and learning are added on. Some language designers focus on tractability and enhance richness, naturalness, and the ability to be acquired.

1.6.3 Defining a Solution

Given an informal description of a task, before even considering a computer, an agent designer should determine what would constitute a solution. This question arises not only in AI but in any software design. Much of **software engineering** involves refining the specification of the task.

Tasks are typically not well specified. Not only is there usually much left unspecified, but also the unspecified parts cannot be filled in arbitrarily. For example, if a user asks a trading agent to find out all the information about resorts that may have unsanitary food practices, they do not want the agent to return all the information about all resorts, even though all of the information requested is in the result. However, if the trading agent does not have complete knowledge about the resorts, returning all of the information may be the only way for it to guarantee that all of the requested information is there. Similarly, one does not want a delivery robot, when asked to take all of the trash to the garbage can, to take everything to the garbage can, even though this may be the only way to guarantee that all of the trash has been taken. Much work in AI is motivated by **commonsense reasoning**; the computer should be able to reach commonsense conclusions about the unstated assumptions.

Given a well-defined task, the next issue is whether it matters if the answer returned is incorrect or incomplete. For example, if the specification asks for all instances, does it matter if some are missing? Does it matter if there are some extra instances? Often a person does not want just any solution but the best solution according to some criteria. There are four common classes of solutions:

Optimal solution An **optimal solution** to a task is one that is the best solution according to some measure of solution quality. This measure is typically specified as an **ordinal**, where only the order matters. In some situations a **cardinal** measure, where the relative magnitudes also matter, is used. For example, a robot may need to take out as much trash as possible; the more trash it can take out, the better. In a more complex example, you may want the delivery robot to take as much of the trash as possible to the garbage can, minimizing the distance traveled, and explicitly specify a trade-off between the effort required and the proportion of the trash taken out. There are also costs associated with making mistakes and throwing out items that are not trash. It may be better to miss some trash than to waste too much time. One general measure of desirability that interacts with probability is **utility** (page 518).

Satisficing solution Often an agent does not need the best solution to a task but just needs some solution. A **satisficing solution** is one that is good enough according to some description of which solutions are adequate. For example, a person may tell a robot that it must take all of the trash out, or tell it to take out three items of trash.

Approximately optimal solution One of the advantages of a cardinal measure of success is that it allows for approximations. An **approximately optimal solution** is one whose measure of quality is close to the best that could theoretically be obtained. Typically, agents do not need optimal solutions to tasks; they only need to get close enough. For example, the robot may not need to travel the optimal distance to take out the trash but may only need to be within, say, 10% of the optimal distance. Some approximation algorithms guarantee that a solution is within some range of optimal, but for some algorithms no guarantees are available.

For some tasks, it is much easier computationally to get an approximately optimal solution than to get an optimal solution. However, for other tasks, it is just as difficult to find an approximately optimal solution that is guaranteed to be within some bounds of optimal as it is to find an optimal solution.

Probable solution A **probable solution** is one that, even though it may not actually be a solution to the task, is likely to be a solution. This is one way to approximate, in a precise manner, a satisficing solution. For example, in the case where the delivery robot could drop the trash or fail to pick it up when it attempts to, you may need the robot to be 80% sure that it has picked up three items of trash. Often you want to distinguish a **false-positive error** (positive answers that are not correct) from a **false-negative error** (negative answers that are correct). Some applications are much more tolerant of one of these types of errors than the other.

These categories are not exclusive. A form of learning known as **probably approximately correct (PAC)** learning considers probably learning an approximately correct concept.

1.6.4 Representations

Once you have some requirements on the nature of a solution, you must represent the task so a computer can solve it.

Computers and human minds are examples of **physical symbol systems**. A **symbol** is a meaningful pattern that can be manipulated. Examples of symbols are written words, sentences, gestures, marks on paper, or sequences of bits. A **symbol system** creates, copies, modifies, and destroys symbols. Essentially, a symbol is one of the patterns manipulated as a unit by a symbol system. The term "physical" is used, because symbols in a physical symbol system are physical objects that are part of the real world, even though they may be internal to computers and brains. They may also need to physically affect action or motor control.

The **physical symbol system hypothesis** of Newell and Simon [1976] is that:

> A physical symbol system has the necessary and sufficient means for general intelligent action.

This is a strong hypothesis. It means that any intelligent agent is necessarily a physical symbol system. It also means that a physical symbol system is all that is needed for intelligent action; there is no magic or as-yet-to-be-discovered quantum phenomenon required. It does not imply that a physical symbol system does not need a body to sense and act in the world.

One aspect of this hypothesis is particularly controversial, namely whether symbols are needed at all levels. For example, consider recognizing a "cat" in a picture. At the top level is the symbol for a cat. At the bottom level are pixels from a camera. There are many intermediate levels that, for example, combine pixels to form lines and textures. These intermediate features are learned from data, and are not learned with the constraint that they are interpretable. Although some people have tried to interpret them, it is reasonable to say that these are not symbols. However, at a high level, they are either trained to be symbols (e.g., by learning a mapping between pixels and symbols, such as *cat*) or can be interpreted as symbols.

An agent can use a physical symbol system to model the world. A **model** of a world is a representation of an agent's beliefs about what is true in the world or how the world changes. The world does not have to be modeled at the most detailed level to be useful. All models are **abstractions**; they represent only part of the world and leave out many of the details. An agent can have a very simplistic model of the world, or it can have a very detailed model of the world. The **level of abstraction** provides a partial ordering of abstraction. A lower-level abstraction includes more details than a higher-level abstraction. An agent can have multiple, even contradictory, models of the world. Models are judged not by whether they are correct, but by whether they are useful.

Example 1.34 A delivery robot can model the environment at a high level of abstraction in terms of rooms, corridors, doors, and obstacles, ignoring distances, its size, the steering angles needed, the slippage of the wheels, the weight of parcels, the details of obstacles, the political situation in Canada, and virtually everything else. The robot could model the environment at lower levels of abstraction by taking some of these details into account. Some of these details may be irrelevant for the successful implementation of the robot, but some may be crucial for the robot to succeed. For example, in some situations the size of the robot and the steering angles may be crucial for not getting stuck around a particular corner. In other situations, if the robot stays close to the center of the corridor, it may not need to model its width or the steering angles.

Choosing an appropriate level of abstraction is difficult for the following reasons:

- A high-level description is easier for a human to specify and understand.
- A low-level description can be more accurate and more predictive. Often, high-level descriptions abstract away details that may be important for actually solving the task.
- The lower the level, the more difficult it is to reason with. This is because a solution at a lower level of detail involves more steps and many more possible courses of action exist from which to choose.
- An agent may not know the information needed for a low-level description. For example, the delivery robot may not know what obstacles it will encounter or how slippery the floor will be at the time that it must decide what to do.

It is often a good idea to model an environment at multiple levels of abstraction. This issue is discussed further in Section 2.2 (page 58).

Biological systems, and computers, can be described at multiple levels of abstraction. At successively lower levels of animals are the neuronal level, the biochemical level (what chemicals and what electrical potentials are being transmitted), the chemical level (what chemical reactions are being carried out), and the level of physics (in terms of forces on atoms and quantum phenomena). What levels above the neuronal level are needed to account for intelligence is still an open question. These levels of description are echoed in the hierarchical structure of science itself, where scientists are divided into physicists, chemists, biologists, psychologists, anthropologists, and so on. Although no level of description is more important than any other, it is plausible that you do not have to emulate every level of a human to build an AI agent but rather you can emulate the higher levels and build them on the foundation of modern computers. This conjecture is part of what AI studies.

The following are two levels that seem to be common to both biological and computational entities:

- The **knowledge level** is the level of abstraction that considers what an agent knows and believes and what its goals are. The knowledge level considers what an agent knows, but not how it reasons. For example, the delivery agent's behavior can be described in terms of whether it knows that a parcel has arrived or not and whether it knows where a particular person is or not. Both human and robotic agents are describable at the knowledge level. At this level, you do not specify how the solution will be computed or even which of the many possible strategies available to the agent will be used.
- The **symbol level** is a level of description of an agent in terms of the reasoning it does. To implement the knowledge level, an agent manipulates symbols to produce answers. Many cognitive science experiments are designed to determine what symbol manipulation occurs during reasoning. Whereas the knowledge level is about what the agent believes about the external world and what its goals are in terms of the outside world, the symbol level is about what goes on inside an agent to reason about the external world.

1.7 Social Impact

AI systems are now widely deployed in society. Individuals, corporations, governments, and other organizations are using AI for applications as varied as voice dictation, text synthesis, text-to-video generation, movie recommendations, personal finance, chatbots, credit scoring, screening employment applications, social media propagation and monitoring, face recognition, semi-autonomous cars, and warehouse automation. Many of these systems can be broadly beneficial. However, there are often adverse impacts on people in racialized populations and underserved communities, and on election results and vaccination campaigns.

There are significant ethical and social impacts of AI systems, leading to demands for human-centered AI that is explainable, transparent, and trustworthy. The inputs to an AI agent include the goals and preferences of the agent, but it is not clear whose preferences they are or should be.

Each chapter concludes with a social impact section discussing issues directly relevant to that chapter's topics. The social impact sections are of two types, sometimes containing both:

- broader impacts of AI, which includes intended or unintended downstream consequences of upstream decisions on the design of the AI system or on the choice of data

- use cases about user-facing applications of AI that have had an impact on society or science, either positive or negative.

Chapter 18 on the social impact of AI considers the effects of AI on the digital economy, work and automation, transportation and sustainability. It highlights the roles of human-centered AI, values, bias, ethics, certification, and regulation.

1.8 Overview of the Book

The rest of the book explores the design space defined by the dimensions of complexity. It considers each dimension separately, where this can be done sensibly.

Part I considers the big view of agents as a coherent vision of AI.

Chapter 2 analyzes what is inside the black box of Figure 1.4 (page 15) and discusses the modular and hierarchical decomposition of intelligent agents.

Part II considers the case of no uncertainty, which is a useful abstraction of many domains.

Chapter 3 considers the simplest case of determining what to do in the case of a single agent that reasons with explicit states, no uncertainty, and has goals to be achieved, but with an indefinite horizon. In this case, the task of solving the goal can be abstracted to searching for a path in a graph. It is shown how extra knowledge of the domain can help the search.

Chapters 4 and 5 show how to exploit features. In particular, Chapter 4 considers how to find possible states given constraints on the assignments of values to features represented as variables. Chapter 5 presents reasoning with propositions in various forms.

Chapter 6 considers the task of planning, in particular determining sequences of actions to solve a goal in deterministic domains.

Part III considers learning and reasoning with uncertainty. In particular, it considers sensing uncertainty and effect uncertainty.

Chapter 7 shows how an agent can learn from past experiences and data. It covers the most common case of learning, namely supervised learning with features, where a function from input features into target features is learned from observational data. **Chapter 8** studies neural networks and deep learning and how features themselves can be learned from sensory observation.

Chapter 9 shows how to reason with uncertainty, in particular with probability and graphical models of independence. **Chapter 10** introduces learning with uncertainty. **Chapter 11** shows how to model causality and learn the effects of interventions (which cannot be learned from observation alone).

Part IV considers planning and acting with uncertainty.

Chapter 12 considers the task of planning with uncertainty. **Chapter 13** deals with reinforcement learning, where agents learn what to do. Chapter 14 expands planning to deal with issues arising from multiple agents.

Part V extends the state and feature-based representations to deal with relational representations, in terms of relations and individuals.

Chapter 15 shows how to reason in terms of individuals and relations. **Chapter 16** discusses how to enable semantic interoperability using knowledge graphs and ontologies. **Chapter 17** shows how reasoning about individuals and relations can be combined with learning and probabilistic reasoning.

Part VI steps back from the details and gives the big picture.

In **Chapter 18** on the social impact of AI, further ethical and social concerns are addressed, by considering various questions, such as: What are the effects, benefits, costs, and risks of deployed AI systems for society? What are the ethical, equity, and regulatory considerations involved in building intelligent agents? How can you ensure that AI systems are fair, transparent, explainable, and trustworthy? How can AI systems be human-centered? What is the impact on sustainability?

Chapter 19 reviews the design space of AI and shows how the material presented can fit into that design space. It also considers some likely and possible future scenarios for the development of AI science and technology.

1.9 Review

The following are the main points you should have learned from this chapter:

- Artificial intelligence is the study of computational agents that act intelligently.
- An agent acts in an environment and only has access to its abilities, its prior knowledge, its history of stimuli, and its goals and preferences.
- A physical symbol system manipulates symbols to determine what to do.
- A designer of an intelligent agent should be concerned about modularity, how to describe the world, how far ahead to plan, uncertainty in both perception and the effects of actions, the structure of goals or preferences, other agents, how to learn from experience, how the agent can reason while interacting with the environment, and the fact that all real agents have limited computational resources.
- To solve a task by computer, the computer must have an effective representation with which to reason.

- To know when it has solved a task, an agent must have a definition of what constitutes an adequate solution, such as whether it has to be optimal, approximately optimal, or almost always optimal, or whether a satisficing solution is adequate.
- In choosing a representation, an agent designer should find a representation that is as close as possible to the task, so that it is easy to determine what is represented and so it can be checked for correctness and be able to be maintained. Often, users want an explanation of why they should believe the answer.
- The social impacts, both beneficial and harmful, of pervasive AI applications are significant, leading to calls for ethical and human-centered AI, certification and regulation.

1.10 References and Further Reading

The ideas in this chapter have been derived from many sources. Here, we try to acknowledge those that are explicitly attributable to particular authors. Most of the other ideas are part of AI folklore; trying to attribute them to anyone would be impossible.

Levesque [2012] provides an accessible account of how thinking can be seen in terms of computation. Haugeland [1997] contains a good collection of articles on the philosophy behind artificial intelligence, including that classic paper of Turing [1950] that proposes the Turing test. Grosz [2012] and Cohen [2005] discuss the Turing test from a more recent perspective. Winograd schemas are described by Levesque [2014]. Srivastava et al. [2022] provide a *Beyond the Imitation Game* benchmark (BIG-bench) consisting of 204 tasks designed to challenge modern learning systems. Grosz [2018] discusses research on what it takes to implement dialog, not just answering one-off questions. Zador et al. [2023] discuss an embodied Turing test, and the role of neuroscience in AI.

Nilsson [2010] and Buchanan [2005] provide accessible histories of AI. Chrisley and Begeer [2000] present many classic papers on AI. Jordan [2019] and the associated commentaries discuss intelligence augmentation.

For discussions on the foundations of AI and the breadth of research in AI, see Kirsh [1991a], Bobrow [1993], and the papers in the corresponding volumes, as well as Schank [1990] and Simon [1995]. The importance of knowledge in AI is discussed in Lenat and Feigenbaum [1991], Sowa [2000], Darwiche [2018], and Brachman and Levesque [2022b].

The physical symbol system hypothesis was posited by Newell and Simon [1976]. Simon [1996] discusses the role of symbol systems in a multidisciplinary context. The distinctions between real, synthetic, and artificial intelligence are discussed by Haugeland [1985], who also provides useful introductory material on interpreted, automatic formal symbol systems and the Church–Turing thesis. Brooks [1990] and Winograd [1990] critique the symbol system hypothesis. Nilsson [2007] evaluates the hypothesis in terms of such criticisms. Shoham [2016] and Marcus and Davis [2019] argue for the importance of symbolic knowledge representation in modern applications.

The use of anytime algorithms is due to Horvitz [1989] and Boddy and Dean [1994]. See Dean and Wellman [1991], Zilberstein [1996], and Russell [1997] for introductions to bounded rationality.

For overviews of cognitive science and the role that AI and other disciplines play in that field, see Gardner [1985], Posner [1989], and Stillings et al. [1987].

Conati et al. [2002] describe a tutoring agent for elementary physics. du Boulay et al. [2023] overview modern tutoring agents. Wellman [2011] overviews research in trading agents. Sandholm [2007] describes how AI can be used for procurement of multiple goods with complex preferences.

A number of AI texts are valuable as reference books complementary to this book, providing a different perspective on AI. In particular, Russell and Norvig [2020] give a more encyclopedic overview of AI. They provide an excellent complementary source for many of the topics covered in this book and also an outstanding review of the scientific literature, which we do not try to duplicate.

The Association for the Advancement of Artificial Intelligence (AAAI) provides introductory material and news at their *AI Topics* website (https://aitopics.org/). *AI Magazine*, published by AAAI, often has excellent overview articles and descriptions of particular applications. *IEEE Intelligent Systems* also provides accessible articles on AI research.

There are many journals that provide in-depth research contributions and conferences where the most up-to-date research is found. These include the journals *Artificial Intelligence*, the *Journal of Artificial Intelligence Research*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, and *Computational Intelligence*, as well as more specialized journals. Much of the cutting-edge research is published first in conferences. Those of most interest to a general audience are the International Joint Conference on Artificial Intelligence (IJCAI), the AAAI Annual Conference, the European Conference on AI (ECAI), the Pacific Rim International Conference on AI (PRICAI), various national conferences, and many specialized conferences, which are referred to in the relevant chapters.

1.11 Exercises

Exercise 1.1 For each of the following, give five reasons why:

(a) A dog is more intelligent than a worm.
(b) A human is more intelligent than a dog.
(c) An organization is more intelligent than an individual human.

Based on these, give a definition of what "more intelligent" may mean.

Exercise 1.2 Give as many disciplines as you can whose aim is to study intelligent behavior of some sort. For each discipline, find out what aspect of behavior is investigated and what tools are used to study it. Be as liberal as you can regarding what defines intelligent behavior.

Exercise 1.3 Find out about two applications of AI (not classes of applications, but specific programs). For each application, write at most one typed page describing it. You should try to cover the following questions:

(a) What does the application actually do (e.g., control a spacecraft, diagnose a photocopier, provide intelligent help for computer users)?
(b) What AI technologies does it use (e.g., model-based diagnosis, belief networks, semantic networks, heuristic search, constraint satisfaction)?
(c) How well does it perform? (According to the authors or to an independent review? How does it compare to humans? How do the authors know how well it works?)
(d) Is it an experimental system or a fielded system? (How many users does it have? What expertise do these users require?)
(e) Why is it intelligent? What aspects of it make it an intelligent system?
(f) [optional] What programming language and environment was it written in? What sort of user interface does it have?
(g) References: Where did you get the information about the application? To what books, articles, or webpages should others who want to know about the application refer?

Exercise 1.4 For each of the Winograd schemas in Example 1.2 (page 6), what knowledge is required to correctly answer the questions? Try to find a “cheap” method to find the answer, such as by comparing the number of results in a Google search for different cases. Try this for six other Winograd schemas of Davis [2015]. Try to construct an example of your own.

Exercise 1.5 Choose four pairs of dimensions that were not compared in Section 1.5.10 (page 35). For each pair, give one commonsense example of where the dimensions interact.

— CHAPTER FROM —

Artificial Intelligence: A Modern Approach

Pages: 19-78

CHAPTER 1

INTRODUCTION

In which we try to explain why we consider artificial intelligence to be a subject most worthy of study, and in which we try to decide what exactly it is, this being a good thing to decide before embarking.

We call ourselves *Homo sapiens*—man the wise—because our **intelligence** is so important to us. For thousands of years, we have tried to understand *how we think and act*—that is, how our brain, a mere handful of matter, can perceive, understand, predict, and manipulate a world far larger and more complicated than itself. The field of **artificial intelligence**, or AI, is concerned with not just understanding but also *building* intelligent entities—machines that can compute how to act effectively and safely in a wide variety of novel situations.

Intelligence

Artificial intelligence

Surveys regularly rank AI as one of the most interesting and fastest-growing fields, and it is already generating over a trillion dollars a year in revenue. AI expert Kai-Fu Lee predicts that its impact will be "more than anything in the history of mankind." Moreover, the intellectual frontiers of AI are wide open. Whereas a student of an older science such as physics might feel that the best ideas have already been discovered by Galileo, Newton, Curie, Einstein, and the rest, AI still has many openings for full-time masterminds.

AI currently encompasses a huge variety of subfields, ranging from the general (learning, reasoning, perception, and so on) to the specific, such as playing chess, proving mathematical theorems, writing poetry, driving a car, or diagnosing diseases. AI is relevant to any intellectual task; it is truly a universal field.

1.1 What Is AI?

We have claimed that AI is interesting, but we have not said what it *is*. Historically, researchers have pursued several different versions of AI. Some have defined intelligence in terms of fidelity to *human* performance, while others prefer an abstract, formal definition of intelligence called **rationality**—loosely speaking, doing the "right thing." The subject matter itself also varies: some consider intelligence to be a property of internal *thought processes* and *reasoning*, while others focus on intelligent *behavior*, an external characterization.[1]

Rationality

From these two dimensions—human vs. rational[2] and thought vs. behavior—there are four possible combinations, and there have been adherents and research programs for all

[1] In the public eye, there is sometimes confusion between the terms "artificial intelligence" and "machine learning." Machine learning is a subfield of AI that studies the ability to improve performance based on experience. Some AI systems use machine learning methods to achieve competence, but some do not.

[2] We are not suggesting that humans are "irrational" in the dictionary sense of "deprived of normal mental clarity." We are merely conceding that human decisions are not always mathematically perfect.

four. The methods used are necessarily different: the pursuit of human-like intelligence must be in part an empirical science related to psychology, involving observations and hypotheses about actual human behavior and thought processes; a rationalist approach, on the other hand, involves a combination of mathematics and engineering, and connects to statistics, control theory, and economics. The various groups have both disparaged and helped each other. Let us look at the four approaches in more detail.

1.1.1 Acting humanly: The Turing test approach

Turing test

The **Turing test**, proposed by Alan Turing (1950), was designed as a thought experiment that would sidestep the philosophical vagueness of the question "Can a machine think?" A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. Chapter 28 discusses the details of the test and whether a computer would really be intelligent if it passed. For now, we note that programming a computer to pass a rigorously applied test provides plenty to work on. The computer would need the following capabilities:

Natural language processing
Knowledge representation
Automated reasoning
Machine learning

- **natural language processing** to communicate successfully in a human language;
- **knowledge representation** to store what it knows or hears;
- **automated reasoning** to answer questions and to draw new conclusions;
- **machine learning** to adapt to new circumstances and to detect and extrapolate patterns.

Total Turing test

Turing viewed the *physical* simulation of a person as unnecessary to demonstrate intelligence. However, other researchers have proposed a **total Turing test**, which requires interaction with objects and people in the real world. To pass the total Turing test, a robot will need

Computer vision
Robotics

- **computer vision** and speech recognition to perceive the world;
- **robotics** to manipulate objects and move about.

These six disciplines compose most of AI. Yet AI researchers have devoted little effort to passing the Turing test, believing that it is more important to study the underlying principles of intelligence. The quest for "artificial flight" succeeded when engineers and inventors stopped imitating birds and started using wind tunnels and learning about aerodynamics. Aeronautical engineering texts do not define the goal of their field as making "machines that fly so exactly like pigeons that they can fool even other pigeons."

1.1.2 Thinking humanly: The cognitive modeling approach

To say that a program thinks like a human, we must know how humans think. We can learn about human thought in three ways:

Introspection
Psychological experiment
Brain imaging

- **introspection**—trying to catch our own thoughts as they go by;
- **psychological experiments**—observing a person in action;
- **brain imaging**—observing the brain in action.

Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. If the program's input–output behavior matches corresponding human behavior, that is evidence that some of the program's mechanisms could also be operating in humans.

For example, Allen Newell and Herbert Simon, who developed GPS, the "General Problem Solver" (Newell and Simon, 1961), were not content merely to have their program solve problems correctly. They were more concerned with comparing the sequence and timing of its reasoning steps to those of human subjects solving the same problems. The interdisciplinary field of **cognitive science** brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind.

Cognitive science

Cognitive science is a fascinating field in itself, worthy of several textbooks and at least one encyclopedia (Wilson and Keil, 1999). We will occasionally comment on similarities or differences between AI techniques and human cognition. Real cognitive science, however, is necessarily based on experimental investigation of actual humans or animals. We will leave that for other books, as we assume the reader has only a computer for experimentation.

In the early days of AI there was often confusion between the approaches. An author would argue that an algorithm performs well on a task and that it is *therefore* a good model of human performance, or vice versa. Modern authors separate the two kinds of claims; this distinction has allowed both AI and cognitive science to develop more rapidly. The two fields fertilize each other, most notably in computer vision, which incorporates neurophysiological evidence into computational models. Recently, the combination of neuroimaging methods combined with machine learning techniques for analyzing such data has led to the beginnings of a capability to "read minds"—that is, to ascertain the semantic content of a person's inner thoughts. This capability could, in turn, shed further light on how human cognition works.

1.1.3 Thinking rationally: The "laws of thought" approach

The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking"—that is, irrefutable reasoning processes. His **syllogisms** provided patterns for argument structures that always yielded correct conclusions when given correct premises. The canonical example starts with *Socrates is a man* and *all men are mortal* and concludes that *Socrates is mortal*. (This example is probably due to Sextus Empiricus rather than Aristotle.) These laws of thought were supposed to govern the operation of the mind; their study initiated the field called **logic**.

Syllogism

Logicians in the 19th century developed a precise notation for statements about objects in the world and the relations among them. (Contrast this with ordinary arithmetic notation, which provides only for statements about *numbers*.) By 1965, programs could, in principle, solve *any* solvable problem described in logical notation. The so-called **logicist** tradition within artificial intelligence hopes to build on such programs to create intelligent systems.

Logicist

Logic as conventionally understood requires knowledge of the world that is *certain*—a condition that, in reality, is seldom achieved. We simply don't know the rules of, say, politics or warfare in the same way that we know the rules of chess or arithmetic. The theory of **probability** fills this gap, allowing rigorous reasoning with uncertain information. In principle, it allows the construction of a comprehensive model of rational thought, leading from raw perceptual information to an understanding of how the world works to predictions about the future. What it does not do, is generate intelligent *behavior*. For that, we need a theory of rational action. Rational thought, by itself, is not enough.

Probability

1.1.4 Acting rationally: The rational agent approach

An **agent** is just something that acts (*agent* comes from the Latin *agere*, to do). Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to

Agent

Rational agent

change, and create and pursue goals. A **rational agent** is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

In the "laws of thought" approach to AI, the emphasis was on correct inferences. Making correct inferences is sometimes *part* of being a rational agent, because one way to act rationally is to deduce that a given action is best and then to act on that conclusion. On the other hand, there are ways of acting rationally that cannot be said to involve inference. For example, recoiling from a hot stove is a reflex action that is usually more successful than a slower action taken after careful deliberation.

All the skills needed for the Turing test also allow an agent to act rationally. Knowledge representation and reasoning enable agents to reach good decisions. We need to be able to generate comprehensible sentences in natural language to get by in a complex society. We need learning not only for erudition, but also because it improves our ability to generate effective behavior, especially in circumstances that are new.

The rational-agent approach to AI has two advantages over the other approaches. First, it is more general than the "laws of thought" approach because correct inference is just one of several possible mechanisms for achieving rationality. Second, it is more amenable to scientific development. The standard of rationality is mathematically well defined and completely general. We can often work back from this specification to derive agent designs that provably achieve it—something that is largely impossible if the goal is to imitate human behavior or thought processes.

Do the right thing

Standard model

For these reasons, the rational-agent approach to AI has prevailed throughout most of the field's history. In the early decades, rational agents were built on logical foundations and formed definite plans to achieve specific goals. Later, methods based on probability theory and machine learning allowed the creation of agents that could make decisions under uncertainty to attain the best expected outcome. In a nutshell, *AI has focused on the study and construction of agents that* ***do the right thing***. What counts as the right thing is defined by the objective that we provide to the agent. This general paradigm is so pervasive that we might call it the **standard model**. It prevails not only in AI, but also in control theory, where a controller minimizes a cost function; in operations research, where a policy maximizes a sum of rewards; in statistics, where a decision rule minimizes a loss function; and in economics, where a decision maker maximizes utility or some measure of social welfare.

Limited rationality

We need to make one important refinement to the standard model to account for the fact that perfect rationality—always taking the exactly optimal action—is not feasible in complex environments. The computational demands are just too high. Chapters 6 and 16 deal with the issue of **limited rationality**—acting appropriately when there is not enough time to do all the computations one might like. However, perfect rationality often remains a good starting point for theoretical analysis.

1.1.5 Beneficial machines

The standard model has been a useful guide for AI research since its inception, but it is probably not the right model in the long run. The reason is that the standard model assumes that we will supply a fully specified objective to the machine.

For an artificially defined task such as chess or shortest-path computation, the task comes with an objective built in—so the standard model is applicable. As we move into the real world, however, it becomes more and more difficult to specify the objective completely and correctly. For example, in designing a self-driving car, one might think that the objective is to reach the destination safely. But driving along any road incurs a risk of injury due to other errant drivers, equipment failure, and so on; thus, a strict goal of safety requires staying in the garage. There is a tradeoff between making progress towards the destination and incurring a risk of injury. How should this tradeoff be made? Furthermore, to what extent can we allow the car to take actions that would annoy other drivers? How much should the car moderate its acceleration, steering, and braking to avoid shaking up the passenger? These kinds of questions are difficult to answer a priori. They are particularly problematic in the general area of human–robot interaction, of which the self-driving car is one example.

The problem of achieving agreement between our true preferences and the objective we put into the machine is called the **value alignment problem**: the values or objectives put into the machine must be aligned with those of the human. If we are developing an AI system in the lab or in a simulator—as has been the case for most of the field's history—there is an easy fix for an incorrectly specified objective: reset the system, fix the objective, and try again. As the field progresses towards increasingly capable intelligent systems that are deployed in the real world, this approach is no longer viable. A system deployed with an incorrect objective will have negative consequences. Moreover, the more intelligent the system, the more negative the consequences.

Value alignment problem

Returning to the apparently unproblematic example of chess, consider what happens if the machine is intelligent enough to reason and act beyond the confines of the chessboard. In that case, it might attempt to increase its chances of winning by such ruses as hypnotizing or blackmailing its opponent or bribing the audience to make rustling noises during its opponent's thinking time.[3] It might also attempt to hijack additional computing power for itself. *These behaviors are not "unintelligent" or "insane"; they are a logical consequence of defining winning as the* sole *objective for the machine.*

It is impossible to anticipate all the ways in which a machine pursuing a fixed objective might misbehave. There is good reason, then, to think that the standard model is inadequate. We don't want machines that are intelligent in the sense of pursuing *their* objectives; we want them to pursue *our* objectives. If we cannot transfer those objectives perfectly to the machine, then we need a new formulation—one in which the machine is pursuing our objectives, but is necessarily *uncertain* as to what they are. When a machine knows that it doesn't know the complete objective, it has an incentive to act cautiously, to ask permission, to learn more about our preferences through observation, and to defer to human control. Ultimately, we want agents that are **provably beneficial** to humans. We will return to this topic in Section 1.5.

Provably beneficial

1.2 The Foundations of Artificial Intelligence

In this section, we provide a brief history of the disciplines that contributed ideas, viewpoints, and techniques to AI. Like any history, this one concentrates on a small number of people, events, and ideas and ignores others that also were important. We organize the history around a series of questions. We certainly would not wish to give the impression that these questions are the only ones the disciplines address or that the disciplines have all been working toward AI as their ultimate fruition.

3 In one of the first books on chess, Ruy Lopez (1561) wrote, "Always place the board so the sun is in your opponent's eyes."

1.2.1 Philosophy

- Can formal rules be used to draw valid conclusions?
- How does the mind arise from a physical brain?
- Where does knowledge come from?
- How does knowledge lead to action?

Aristotle (384–322 BCE) was the first to formulate a precise set of laws governing the rational part of the mind. He developed an informal system of syllogisms for proper reasoning, which in principle allowed one to generate conclusions mechanically, given initial premises.

Ramon Llull (c. 1232–1315) devised a system of reasoning published as *Ars Magna* or *The Great Art* (1305). Llull tried to implement his system using an actual mechanical device: a set of paper wheels that could be rotated into different permutations.

Around 1500, Leonardo da Vinci (1452–1519) designed but did not build a mechanical calculator; recent reconstructions have shown the design to be functional. The first known calculating machine was constructed around 1623 by the German scientist Wilhelm Schickard (1592–1635). Blaise Pascal (1623–1662) built the Pascaline in 1642 and wrote that it "produces effects which appear nearer to thought than all the actions of animals." Gottfried Wilhelm Leibniz (1646–1716) built a mechanical device intended to carry out operations on concepts rather than numbers, but its scope was rather limited. In his 1651 book *Leviathan*, Thomas Hobbes (1588–1679) suggested the idea of a thinking machine, an "artificial animal" in his words, arguing "For what is the heart but a spring; and the nerves, but so many strings; and the joints, but so many wheels." He also suggested that reasoning was like numerical computation: "For 'reason' . . . is nothing but 'reckoning,' that is adding and subtracting."

It's one thing to say that the mind operates, at least in part, according to logical or numerical rules, and to build physical systems that emulate some of those rules. It's another to say that the mind itself *is* such a physical system. René Descartes (1596–1650) gave the first clear discussion of the distinction between mind and matter. He noted that a purely physical conception of the mind seems to leave little room for free will. If the mind is governed entirely by physical laws, then it has no more free will than a rock "deciding" to fall downward.

Dualism

Descartes was a proponent of **dualism**. He held that there is a part of the human mind (or soul or spirit) that is outside of nature, exempt from physical laws. Animals, on the other hand, did not possess this dual quality; they could be treated as machines.

An alternative to dualism is **materialism**, which holds that the brain's operation according to the laws of physics *constitutes* the mind. Free will is simply the way that the perception of available choices appears to the choosing entity. The terms **physicalism** and **naturalism** are also used to describe this view that stands in contrast to the supernatural.

Empiricism

Given a physical mind that manipulates knowledge, the next problem is to establish the source of knowledge. The **empiricism** movement, starting with Francis Bacon's (1561–1626) *Novum Organum*,[4] is characterized by a dictum of John Locke (1632–1704): "Nothing is in the understanding, which was not first in the senses."

Induction

David Hume's (1711–1776) *A Treatise of Human Nature* (Hume, 1739) proposed what is now known as the principle of **induction**: that general rules are acquired by exposure to repeated associations between their elements.

[4] The *Novum Organum* is an update of Aristotle's *Organon*, or instrument of thought.

Building on the work of Ludwig Wittgenstein (1889–1951) and Bertrand Russell (1872–1970), the famous Vienna Circle (Sigmund, 2017), a group of philosophers and mathematicians meeting in Vienna in the 1920s and 1930s, developed the doctrine of **logical positivism**. This doctrine holds that all knowledge can be characterized by logical theories connected, ultimately, to **observation sentences** that correspond to sensory inputs; thus logical positivism combines rationalism and empiricism.

Logical positivism

Observation sentence

The **confirmation theory** of Rudolf Carnap (1891–1970) and Carl Hempel (1905–1997) attempted to analyze the acquisition of knowledge from experience by quantifying the degree of belief that should be assigned to logical sentences based on their connection to observations that confirm or disconfirm them. Carnap's book *The Logical Structure of the World* (1928) was perhaps the first theory of mind as a computational process.

Confirmation theory

The final element in the philosophical picture of the mind is the connection between knowledge and action. This question is vital to AI because intelligence requires action as well as reasoning. Moreover, only by understanding how actions are justified can we understand how to build an agent whose actions are justifiable (or rational).

Aristotle argued (in *De Motu Animalium*) that actions are justified by a logical connection between goals and knowledge of the action's outcome:

> But how does it happen that thinking is sometimes accompanied by action and sometimes not, sometimes by motion, and sometimes not? It looks as if almost the same thing happens as in the case of reasoning and making inferences about unchanging objects. But in that case the end is a speculative proposition ... whereas here the conclusion which results from the two premises is an action. ... I need covering; a cloak is a covering. I need a cloak. What I need, I have to make; I need a cloak. I have to make a cloak. And the conclusion, the "I have to make a cloak," is an action.

In the *Nicomachean Ethics* (Book III. 3, 1112b), Aristotle further elaborates on this topic, suggesting an algorithm:

> We deliberate not about ends, but about means. For a doctor does not deliberate whether he shall heal, nor an orator whether he shall persuade, ... They assume the end and consider how and by what means it is attained, and if it seems easily and best produced thereby; while if it is achieved by one means only they consider *how* it will be achieved by this and by what means *this* will be achieved, till they come to the first cause, ... and what is last in the order of analysis seems to be first in the order of becoming. And if we come on an impossibility, we give up the search, e.g., if we need money and this cannot be got; but if a thing appears possible we try to do it.

Aristotle's algorithm was implemented 2300 years later by Newell and Simon in their **General Problem Solver** program. We would now call it a greedy regression planning system (see Chapter 11). Methods based on logical planning to achieve definite goals dominated the first few decades of theoretical research in AI.

Thinking purely in terms of actions achieving goals is often useful but sometimes inapplicable. For example, if there are several different ways to achieve a goal, there needs to be some way to choose among them. More importantly, it may not be possible to achieve a goal with certainty, but some action must still be taken. How then should one decide? Antoine Arnauld (1662), analyzing the notion of rational decisions in gambling, proposed a quantitative formula for maximizing the expected monetary value of the outcome. Later, Daniel Bernoulli (1738) introduced the more general notion of **utility** to capture the internal, subjective value

Utility

of an outcome. The modern notion of rational decision making under uncertainty involves maximizing expected utility, as explained in Chapter 15.

In matters of ethics and public policy, a decision maker must consider the interests of multiple individuals. Jeremy Bentham (1823) and John Stuart Mill (1863) promoted the idea of **utilitarianism**: that rational decision making based on maximizing utility should apply to all spheres of human activity, including public policy decisions made on behalf of many individuals. Utilitarianism is a specific kind of **consequentialism**: the idea that what is right and wrong is determined by the expected outcomes of an action.

Utilitarianism

In contrast, Immanuel Kant, in 1785, proposed a theory of rule-based or **deontological ethics**, in which "doing the right thing" is determined not by outcomes but by universal social laws that govern allowable actions, such as "don't lie" or "don't kill." Thus, a utilitarian could tell a white lie if the expected good outweighs the bad, but a Kantian would be bound not to, because lying is inherently wrong. Mill acknowledged the value of rules, but understood them as efficient decision procedures compiled from first-principles reasoning about consequences. Many modern AI systems adopt exactly this approach.

Deontological ethics

1.2.2 Mathematics

- What are the formal rules to draw valid conclusions?
- What can be computed?
- How do we reason with uncertain information?

Philosophers staked out some of the fundamental ideas of AI, but the leap to a formal science required the mathematization of logic and probability and the introduction of a new branch of mathematics: computation.

Formal logic

The idea of **formal logic** can be traced back to the philosophers of ancient Greece, India, and China, but its mathematical development really began with the work of George Boole (1815–1864), who worked out the details of propositional, or Boolean, logic (Boole, 1847). In 1879, Gottlob Frege (1848–1925) extended Boole's logic to include objects and relations, creating the first-order logic that is used today.[5] In addition to its central role in the early period of AI research, first-order logic motivated the work of Gödel and Turing that underpinned computation itself, as we explain below.

Probability

The theory of **probability** can be seen as generalizing logic to situations with uncertain information—a consideration of great importance for AI. Gerolamo Cardano (1501–1576) first framed the idea of probability, describing it in terms of the possible outcomes of gambling events. In 1654, Blaise Pascal (1623–1662), in a letter to Pierre Fermat (1601–1665), showed how to predict the future of an unfinished gambling game and assign average payoffs to the gamblers. Probability quickly became an invaluable part of the quantitative sciences, helping to deal with uncertain measurements and incomplete theories. Jacob Bernoulli (1654–1705, uncle of Daniel), Pierre Laplace (1749–1827), and others advanced the theory and introduced new statistical methods. Thomas Bayes (1702–1761) proposed a rule for updating probabilities in the light of new evidence; Bayes' rule is a crucial tool for AI systems.

Statistics

The formalization of probability, combined with the availability of data, led to the emergence of **statistics** as a field. One of the first uses was John Graunt's analysis of Lon-

[5] Frege's proposed notation for first-order logic—an arcane combination of textual and geometric features—never became popular.

don census data in 1662. Ronald Fisher is considered the first modern statistician (Fisher, 1922). He brought together the ideas of probability, experiment design, analysis of data, and computing—in 1919, he insisted that he couldn't do his work without a mechanical calculator called the MILLIONAIRE (the first calculator that could do multiplication), even though the cost of the calculator was more than his annual salary (Ross, 2012).

The history of computation is as old as the history of numbers, but the first nontrivial **algorithm** is thought to be Euclid's algorithm for computing greatest common divisors. The word *algorithm* comes from Muhammad ibn Musa al-Khwarizmi, a 9th century mathematician, whose writings also introduced Arabic numerals and algebra to Europe. Boole and others discussed algorithms for logical deduction, and, by the late 19th century, efforts were under way to formalize general mathematical reasoning as logical deduction.

Algorithm

Kurt Gödel (1906–1978) showed that there exists an effective procedure to prove any true statement in the first-order logic of Frege and Russell, but that first-order logic could not capture the principle of mathematical induction needed to characterize the natural numbers. In 1931, Gödel showed that limits on deduction do exist. His **incompleteness theorem** showed that in any formal theory as strong as Peano arithmetic (the elementary theory of natural numbers), there are necessarily true statements that have no proof within the theory.

Incompleteness theorem

This fundamental result can also be interpreted as showing that some functions on the integers cannot be represented by an algorithm—that is, they cannot be computed. This motivated Alan Turing (1912–1954) to try to characterize exactly which functions *are* **computable**—capable of being computed by an effective procedure. The Church–Turing thesis proposes to identify the general notion of computability with functions computed by a Turing machine (Turing, 1936). Turing also showed that there were some functions that no Turing machine can compute. For example, no machine can tell *in general* whether a given program will return an answer on a given input or run forever.

Computability

Although computability is important to an understanding of computation, the notion of **tractability** has had an even greater impact on AI. Roughly speaking, a problem is called intractable if the time required to solve instances of the problem grows exponentially with the size of the instances. The distinction between polynomial and exponential growth in complexity was first emphasized in the mid-1960s (Cobham, 1964; Edmonds, 1965). It is important because exponential growth means that even moderately large instances cannot be solved in any reasonable time.

Tractability

The theory of **NP-completeness**, pioneered by Cook (1971) and Karp (1972), provides a basis for analyzing the tractability of problems: any problem class to which the class of NP-complete problems can be reduced is likely to be intractable. (Although it has not been proved that NP-complete problems are necessarily intractable, most theoreticians believe it.) These results contrast with the optimism with which the popular press greeted the first computers—"Electronic Super-Brains" that were "Faster than Einstein!" Despite the increasing speed of computers, careful use of resources and necessary imperfection will characterize intelligent systems. Put crudely, the world is an *extremely* large problem instance!

NP-completeness

1.2.3 Economics

- How should we make decisions in accordance with our preferences?
- How should we do this when others may not go along?
- How should we do this when the payoff may be far in the future?

The science of economics originated in 1776, when Adam Smith (1723–1790) published *An Inquiry into the Nature and Causes of the Wealth of Nations*. Smith proposed to analyze economies as consisting of many individual agents attending to their own interests. Smith was not, however, advocating financial greed as a moral position: his earlier (1759) book *The Theory of Moral Sentiments* begins by pointing out that concern for the well-being of others is an essential component of the interests of every individual.

Most people think of economics as being about money, and indeed the first mathematical analysis of decisions under uncertainty, the maximum-expected-value formula of Arnauld (1662), dealt with the monetary value of bets. Daniel Bernoulli (1738) noticed that this formula didn't seem to work well for larger amounts of money, such as investments in maritime trading expeditions. He proposed instead a principle based on maximization of expected utility, and explained human investment choices by proposing that the marginal utility of an additional quantity of money diminished as one acquired more money.

Léon Walras (pronounced "Valrasse") (1834–1910) gave utility theory a more general foundation in terms of preferences between gambles on any outcomes (not just monetary outcomes). The theory was improved by Ramsey (1931) and later by John von Neumann and Oskar Morgenstern in their book *The Theory of Games and Economic Behavior* (1944). Economics is no longer the study of money; rather it is the study of desires and preferences.

Decision theory

Decision theory, which combines probability theory with utility theory, provides a formal and complete framework for individual decisions (economic or otherwise) made under uncertainty—that is, in cases where probabilistic descriptions appropriately capture the decision maker's environment. This is suitable for "large" economies where each agent need pay no attention to the actions of other agents as individuals. For "small" economies, the situation is much more like a **game**: the actions of one player can significantly affect the utility of another (either positively or negatively). Von Neumann and Morgenstern's development of **game theory** (see also Luce and Raiffa, 1957) included the surprising result that, for some games, a rational agent should adopt policies that are (or least appear to be) randomized. Unlike decision theory, game theory does not offer an unambiguous prescription for selecting actions. In AI, decisions involving multiple agents are studied under the heading of **multiagent systems** (Chapter 17).

Operations research

Economists, with some exceptions, did not address the third question listed above: how to make rational decisions when payoffs from actions are not immediate but instead result from several actions taken *in sequence*. This topic was pursued in the field of **operations research**, which emerged in World War II from efforts in Britain to optimize radar installations, and later found innumerable civilian applications. The work of Richard Bellman (1957) formalized a class of sequential decision problems called **Markov decision processes**, which we study in Chapter 16 and, under the heading of **reinforcement learning**, in Chapter 23.

Satisficing

Work in economics and operations research has contributed much to our notion of rational agents, yet for many years AI research developed along entirely separate paths. One reason was the apparent complexity of making rational decisions. The pioneering AI researcher Herbert Simon (1916–2001) won the Nobel Prize in economics in 1978 for his early work showing that models based on **satisficing**—making decisions that are "good enough," rather than laboriously calculating an optimal decision—gave a better description of actual human behavior (Simon, 1947). Since the 1990s, there has been a resurgence of interest in decision-theoretic techniques for AI.

1.2.4 Neuroscience

- How do brains process information?

Neuroscience

Neuroscience is the study of the nervous system, particularly the brain. Although the exact way in which the brain enables thought is one of the great mysteries of science, the fact that it *does* enable thought has been appreciated for thousands of years because of the evidence that strong blows to the head can lead to mental incapacitation. It has also long been known that human brains are somehow different; in about 335 BCE Aristotle wrote, "Of all the animals, man has the largest brain in proportion to his size."[6] Still, it was not until the middle of the 18th century that the brain was widely recognized as the seat of consciousness. Before then, candidate locations included the heart and the spleen.

Neuron

Paul Broca's (1824–1880) investigation of aphasia (speech deficit) in brain-damaged patients in 1861 initiated the study of the brain's functional organization by identifying a localized area in the left hemisphere—now called Broca's area—that is responsible for speech production.[7] By that time, it was known that the brain consisted largely of nerve cells, or **neurons**, but it was not until 1873 that Camillo Golgi (1843–1926) developed a staining technique allowing the observation of individual neurons (see Figure 1.1). This technique was used by Santiago Ramon y Cajal (1852–1934) in his pioneering studies of neuronal organization.[8] It is now widely accepted that cognitive functions result from the electrochemical operation of these structures. That is, *a collection of simple cells can lead to thought, action, and consciousness*. In the pithy words of John Searle (1992), *brains cause minds*.

We now have some data on the mapping between areas of the brain and the parts of the body that they control or from which they receive sensory input. Such mappings are able to change radically over the course of a few weeks, and some animals seem to have multiple maps. Moreover, we do not fully understand how other areas can take over functions when one area is damaged. There is almost no theory on how an individual memory is stored or on how higher-level cognitive functions operate.

Optogenetics

The measurement of intact brain activity began in 1929 with the invention by Hans Berger of the electroencephalograph (EEG). The development of functional magnetic resonance imaging (fMRI) (Ogawa *et al.*, 1990; Cabeza and Nyberg, 2001) is giving neuroscientists unprecedentedly detailed images of brain activity, enabling measurements that correspond in interesting ways to ongoing cognitive processes. These are augmented by advances in single-cell electrical recording of neuron activity and by the methods of **optogenetics** (Crick, 1999; Zemelman *et al.*, 2002; Han and Boyden, 2007), which allow both measurement and control of individual neurons modified to be light-sensitive.

Brain–machine interface

The development of **brain–machine interfaces** (Lebedev and Nicolelis, 2006) for both sensing and motor control not only promises to restore function to disabled individuals, but also sheds light on many aspects of neural systems. A remarkable finding from this work is that the brain is able to adjust itself to interface successfully with an external device, treating it in effect like another sensory organ or limb.

---

[6] It has since been discovered that the tree shrew and some bird species exceed the human brain/body ratio.

[7] Many cite Alexander Hood (1824) as a possible prior source.

[8] Golgi persisted in his belief that the brain's functions were carried out primarily in a continuous medium in which neurons were embedded, whereas Cajal propounded the "neuronal doctrine." The two shared the Nobel Prize in 1906 but gave mutually antagonistic acceptance speeches.

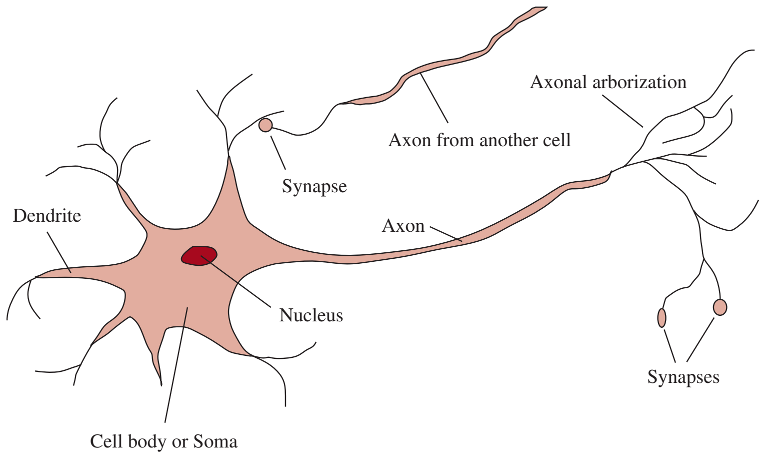

Figure 1.1 The parts of a nerve cell or neuron. Each neuron consists of a cell body, or soma, that contains a cell nucleus. Branching out from the cell body are a number of fibers called dendrites and a single long fiber called the axon. The axon stretches out for a long distance, much longer than the scale in this diagram indicates. Typically, an axon is 1 cm long (100 times the diameter of the cell body), but can reach up to 1 meter. A neuron makes connections with 10 to 100,000 other neurons at junctions called synapses. Signals are propagated from neuron to neuron by a complicated electrochemical reaction. The signals control brain activity in the short term and also enable long-term changes in the connectivity of neurons. These mechanisms are thought to form the basis for learning in the brain. Most information processing goes on in the cerebral cortex, the outer layer of the brain. The basic organizational unit appears to be a column of tissue about 0.5 mm in diameter, containing about 20,000 neurons and extending the full depth of the cortex (about 4 mm in humans).

Brains and digital computers have somewhat different properties. Figure 1.2 shows that computers have a cycle time that is a million times faster than a brain. The brain makes up for that with far more storage and interconnection than even a high-end personal computer, although the largest supercomputers match the brain on some metrics. Futurists make much of these numbers, pointing to an approaching **singularity** at which computers reach a superhuman level of performance (Vinge, 1993; Kurzweil, 2005; Doctorow and Stross, 2012), and then rapidly improve themselves even further. But the comparisons of raw numbers are not especially informative. Even with a computer of virtually unlimited capacity, we still require further conceptual breakthroughs in our understanding of intelligence (see Chapter 29). Crudely put, without the right theory, faster machines just give you the wrong answer faster.

Singularity

1.2.5 Psychology

- How do humans and animals think and act?

The origins of scientific psychology are usually traced to the work of the German physicist Hermann von Helmholtz (1821–1894) and his student Wilhelm Wundt (1832–1920). Helmholtz applied the scientific method to the study of human vision, and his *Handbook of Physiological Optics* has been described as “the single most important treatise on the physics and physiology of human vision” (Nalwa, 1993, p.15). In 1879, Wundt opened the first laboratory of experimental psychology, at the University of Leipzig. Wundt insisted on carefully

	Supercomputer	Personal Computer	Human Brain
Computational units	10^6 GPUs + CPUs	8 CPU cores	10^6 columns
	10^{15} transistors	10^{10} transistors	10^{11} neurons
Storage units	10^{16} bytes RAM	10^{10} bytes RAM	10^{11} neurons
	10^{17} bytes disk	10^{12} bytes disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{18}	10^{10}	10^{17}

Figure 1.2 A crude comparison of a leading supercomputer, Summit (Feldman, 2017); a typical personal computer of 2019; and the human brain. Human brain power has not changed much in thousands of years, whereas supercomputers have improved from megaFLOPs in the 1960s to gigaFLOPs in the 1980s, teraFLOPs in the 1990s, petaFLOPs in 2008, and exaFLOPs in 2018 (1 exaFLOP = 10^{18} floating point operations per second).

controlled experiments in which his workers would perform a perceptual or associative task while introspecting on their thought processes. The careful controls went a long way toward making psychology a science, but the subjective nature of the data made it unlikely that experimenters would ever disconfirm their own theories.

Biologists studying animal behavior, on the other hand, lacked introspective data and developed an objective methodology, as described by H. S. Jennings (1906) in his influential work *Behavior of the Lower Organisms*. Applying this viewpoint to humans, the **behaviorism** movement, led by John Watson (1878–1958), rejected *any* theory involving mental processes on the grounds that introspection could not provide reliable evidence. Behaviorists insisted on studying only objective measures of the percepts (or *stimulus*) given to an animal and its resulting actions (or *response*). Behaviorism discovered a lot about rats and pigeons but had less success at understanding humans.

Behaviorism

Cognitive psychology, which views the brain as an information-processing device, can be traced back at least to the works of William James (1842–1910). Helmholtz also insisted that perception involved a form of unconscious logical inference. The cognitive viewpoint was largely eclipsed by behaviorism in the United States, but at Cambridge's Applied Psychology Unit, directed by Frederic Bartlett (1886–1969), cognitive modeling was able to flourish. *The Nature of Explanation*, by Bartlett's student and successor Kenneth Craik (1943), forcefully reestablished the legitimacy of such "mental" terms as beliefs and goals, arguing that they are just as scientific as, say, using pressure and temperature to talk about gases, despite gasses being made of molecules that have neither.

Cognitive psychology

Craik specified the three key steps of a knowledge-based agent: (1) the stimulus must be translated into an internal representation, (2) the representation is manipulated by cognitive processes to derive new internal representations, and (3) these are in turn retranslated back into action. He clearly explained why this was a good design for an agent:

> If the organism carries a "small-scale model" of external reality and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of past events in dealing with the present and future, and in every way to react in a much fuller, safer, and more competent manner to the emergencies which face it. (Craik, 1943)

After Craik's death in a bicycle accident in 1945, his work was continued by Donald Broadbent, whose book *Perception and Communication* (1958) was one of the first works to model psychological phenomena as information processing. Meanwhile, in the United States, the development of computer modeling led to the creation of the field of **cognitive science**. The field can be said to have started at a workshop in September 1956 at MIT—just two months after the conference at which AI itself was "born."

At the workshop, George Miller presented *The Magic Number Seven*, Noam Chomsky presented *Three Models of Language*, and Allen Newell and Herbert Simon presented *The Logic Theory Machine*. These three influential papers showed how computer models could be used to address the psychology of memory, language, and logical thinking, respectively. It is now a common (although far from universal) view among psychologists that "a cognitive theory should be like a computer program" (Anderson, 1980); that is, it should describe the operation of a cognitive function in terms of the processing of information.

For purposes of this review, we will count the field of **human–computer interaction** (HCI) under psychology. Doug Engelbart, one of the pioneers of HCI, championed the idea of **intelligence augmentation**—IA rather than AI. He believed that computers should augment human abilities rather than automate away human tasks. In 1968, Engelbart's "mother of all demos" showed off for the first time the computer mouse, a windowing system, hypertext, and video conferencing—all in an effort to demonstrate what human knowledge workers could collectively accomplish with some intelligence augmentation.

Intelligence augmentation

Today we are more likely to see IA and AI as two sides of the same coin, with the former emphasizing human control and the latter emphasizing intelligent behavior on the part of the machine. Both are needed for machines to be useful to humans.

1.2.6 Computer engineering

- How can we build an efficient computer?

The modern digital electronic computer was invented independently and almost simultaneously by scientists in three countries embattled in World War II. The first *operational* computer was the electromechanical Heath Robinson,[9] built in 1943 by Alan Turing's team for a single purpose: deciphering German messages. In 1943, the same group developed the Colossus, a powerful general-purpose machine based on vacuum tubes.[10] The first operational *programmable* computer was the Z-3, the invention of Konrad Zuse in Germany in 1941. Zuse also invented floating-point numbers and the first high-level programming language, Plankalkül. The first *electronic* computer, the ABC, was assembled by John Atanasoff and his student Clifford Berry between 1940 and 1942 at Iowa State University. Atanasoff's research received little support or recognition; it was the ENIAC, developed as part of a secret military project at the University of Pennsylvania by a team including John Mauchly and J. Presper Eckert, that proved to be the most influential forerunner of modern computers.

Since that time, each generation of computer hardware has brought an increase in speed and capacity and a decrease in price—a trend captured in **Moore's law**. Performance doubled every 18 months or so until around 2005, when power dissipation problems led manufacturers

Moore's law

[9] A complex machine named after a British cartoonist who depicted whimsical and absurdly complicated contraptions for everyday tasks such as buttering toast.

[10] In the postwar period, Turing wanted to use these computers for AI research—for example, he created an outline of the first chess program (Turing *et al.*, 1953)—but the British government blocked this research.

to start multiplying the number of CPU cores rather than the clock speed. Current expectations are that future increases in functionality will come from massive parallelism—a curious convergence with the properties of the brain. We also see new hardware designs based on the idea that in dealing with an uncertain world, we don't need 64 bits of precision in our numbers; just 16 bits (as in the `bfloat16` format) or even 8 bits will be enough, and will enable faster processing.

We are just beginning to see hardware tuned for AI applications, such as the graphics processing unit (GPU), tensor processing unit (TPU), and wafer scale engine (WSE). From the 1960s to about 2012, the amount of computing power used to train top machine learning applications followed Moore's law. Beginning in 2012, things changed: from 2012 to 2018 there was a 300,000-fold increase, which works out to a doubling every 100 days or so (Amodei and Hernandez, 2018). A machine learning model that took a full day to train in 2014 takes only two minutes in 2018 (Ying *et al.*, 2018). Although it is not yet practical, **quantum computing** holds out the promise of far greater accelerations for some important subclasses of AI algorithms. Quantum computing

Of course, there were calculating devices before the electronic computer. The earliest automated machines, dating from the 17th century, were discussed on page 24. The first *programmable* machine was a loom, devised in 1805 by Joseph Marie Jacquard (1752–1834), that used punched cards to store instructions for the pattern to be woven.

In the mid-19th century, Charles Babbage (1792–1871) designed two computing machines, neither of which he completed. The Difference Engine was intended to compute mathematical tables for engineering and scientific projects. It was finally built and shown to work in 1991 (Swade, 2000). Babbage's Analytical Engine was far more ambitious: it included addressable memory, stored programs based on Jacquard's punched cards, and conditional jumps. It was the first machine capable of universal computation.

Babbage's colleague Ada Lovelace, daughter of the poet Lord Byron, understood its potential, describing it as "a thinking or . . . a reasoning machine," one capable of reasoning about "all subjects in the universe" (Lovelace, 1843). She also anticipated AI's hype cycles, writing, "It is desirable to guard against the possibility of exaggerated ideas that might arise as to the powers of the Analytical Engine." Unfortunately, Babbage's machines and Lovelace's ideas were largely forgotten.

AI also owes a debt to the software side of computer science, which has supplied the operating systems, programming languages, and tools needed to write modern programs (and papers about them). But this is one area where the debt has been repaid: work in AI has pioneered many ideas that have made their way back to mainstream computer science, including time sharing, interactive interpreters, personal computers with windows and mice, rapid development environments, the linked-list data type, automatic storage management, and key concepts of symbolic, functional, declarative, and object-oriented programming.

1.2.7 Control theory and cybernetics

- How can artifacts operate under their own control?

Ktesibios of Alexandria (c. 250 BCE) built the first self-controlling machine: a water clock with a regulator that maintained a constant flow rate. This invention changed the definition of what an artifact could do. Previously, only living things could modify their behavior in response to changes in the environment. Other examples of self-regulating feedback control systems include the steam engine governor, created by James Watt (1736–1819), and the thermostat, invented by Cornelis Drebbel (1572–1633), who also invented the submarine. James Clerk Maxwell (1868) initiated the mathematical theory of control systems.

Control theory

A central figure in the post-war development of **control theory** was Norbert Wiener (1894–1964). Wiener was a brilliant mathematician who worked with Bertrand Russell, among others, before developing an interest in biological and mechanical control systems and their connection to cognition. Like Craik (who also used control systems as psychological models), Wiener and his colleagues Arturo Rosenblueth and Julian Bigelow challenged the behaviorist orthodoxy (Rosenblueth *et al.*, 1943). They viewed purposive behavior as arising from a regulatory mechanism trying to minimize "error"—the difference between current state and goal state. In the late 1940s, Wiener, along with Warren McCulloch, Walter Pitts, and John von Neumann, organized a series of influential conferences that explored the new mathematical and computational models of cognition. Wiener's book *Cybernetics* (1948) became a bestseller and awoke the public to the possibility of artificially intelligent machines.

Cybernetics

Meanwhile, in Britain, W. Ross Ashby pioneered similar ideas (Ashby, 1940). Ashby, Alan Turing, Grey Walter, and others formed the Ratio Club for "those who had Wiener's ideas before Wiener's book appeared." Ashby's *Design for a Brain* (1948, 1952) elaborated on his idea that intelligence could be created by the use of **homeostatic** devices containing appropriate feedback loops to achieve stable adaptive behavior.

Homeostatic

Modern control theory, especially the branch known as stochastic optimal control, has as its goal the design of systems that minimize a **cost function** over time. This roughly matches the standard model of AI: designing systems that behave optimally. Why, then, are AI and control theory two different fields, despite the close connections among their founders? The answer lies in the close coupling between the mathematical techniques that were familiar to the participants and the corresponding sets of problems that were encompassed in each world view. Calculus and matrix algebra, the tools of control theory, lend themselves to systems that are describable by fixed sets of continuous variables, whereas AI was founded in part as a way to escape from these perceived limitations. The tools of logical inference and computation allowed AI researchers to consider problems such as language, vision, and symbolic planning that fell completely outside the control theorist's purview.

Cost function

1.2.8 Linguistics

- How does language relate to thought?

In 1957, B. F. Skinner published *Verbal Behavior*. This was a comprehensive, detailed account of the behaviorist approach to language learning, written by the foremost expert in the field. But curiously, a review of the book became as well known as the book itself, and served to almost kill off interest in behaviorism. The author of the review was the linguist Noam Chomsky, who had just published a book on his own theory, *Syntactic Structures*. Chomsky pointed out that the behaviorist theory did not address the notion of creativity in language—it did not explain how children could understand and make up sentences that they had never heard before. Chomsky's theory—based on syntactic models going back to the Indian linguist Panini (c. 350 BCE)—could explain this, and unlike previous theories, it was formal enough that it could in principle be programmed.

Computational linguistics

Modern linguistics and AI, then, were "born" at about the same time, and grew up together, intersecting in a hybrid field called **computational linguistics** or **natural language**

processing. The problem of understanding language turned out to be considerably more complex than it seemed in 1957. Understanding language requires an understanding of the subject matter and context, not just an understanding of the structure of sentences. This might seem obvious, but it was not widely appreciated until the 1960s. Much of the early work in **knowledge representation** (the study of how to put knowledge into a form that a computer can reason with) was tied to language and informed by research in linguistics, which was connected in turn to decades of work on the philosophical analysis of language.

1.3 The History of Artificial Intelligence

One quick way to summarize the milestones in AI history is to list the Turing Award winners: Marvin Minsky (1969) and John McCarthy (1971) for defining the foundations of the field based on representation and reasoning; Allen Newell and Herbert Simon (1975) for symbolic models of problem solving and human cognition; Ed Feigenbaum and Raj Reddy (1994) for developing expert systems that encode human knowledge to solve real-world problems; Judea Pearl (2011) for developing probabilistic reasoning techniques that deal with uncertainty in a principled manner; and finally Yoshua Bengio, Geoffrey Hinton, and Yann LeCun (2019) for making "deep learning" (multilayer neural networks) a critical part of modern computing. The rest of this section goes into more detail on each phase of AI history.

1.3.1 The inception of artificial intelligence (1943–1956)

The first work that is now generally recognized as AI was done by Warren McCulloch and Walter Pitts (1943). Inspired by the mathematical modeling work of Pitts's advisor Nicolas Rashevsky (1936, 1938), they drew on three sources: knowledge of the basic physiology and function of neurons in the brain; a formal analysis of propositional logic due to Russell and Whitehead; and Turing's theory of computation. They proposed a model of artificial neurons in which each neuron is characterized as being "on" or "off," with a switch to "on" occurring in response to stimulation by a sufficient number of neighboring neurons. The state of a neuron was conceived of as "factually equivalent to a proposition which proposed its adequate stimulus." They showed, for example, that any computable function could be computed by some network of connected neurons, and that all the logical connectives (AND, OR, NOT, etc.) could be implemented by simple network structures. McCulloch and Pitts also suggested that suitably defined networks could learn. Donald Hebb (1949) demonstrated a simple updating rule for modifying the connection strengths between neurons. His rule, now called **Hebbian learning**, remains an influential model to this day.

Hebbian learning

Two undergraduate students at Harvard, Marvin Minsky (1927–2016) and Dean Edmonds, built the first neural network computer in 1950. The SNARC, as it was called, used 3000 vacuum tubes and a surplus automatic pilot mechanism from a B-24 bomber to simulate a network of 40 neurons. Later, at Princeton, Minsky studied universal computation in neural networks. His Ph.D. committee was skeptical about whether this kind of work should be considered mathematics, but von Neumann reportedly said, "If it isn't now, it will be someday."

There were a number of other examples of early work that can be characterized as AI, including two checkers-playing programs developed independently in 1952 by Christopher Strachey at the University of Manchester and by Arthur Samuel at IBM. However, Alan Turing's vision was the most influential. He gave lectures on the topic as early as 1947 at the London Mathematical Society and articulated a persuasive agenda in his 1950 article "Computing Machinery and Intelligence." Therein, he introduced the Turing test, machine learning, genetic algorithms, and reinforcement learning. He dealt with many of the objections raised to the possibility of AI, as described in Chapter 28. He also suggested that it would be easier to create human-level AI by developing learning algorithms and then teaching the machine rather than by programming its intelligence by hand. In subsequent lectures he warned that achieving this goal might not be the best thing for the human race.

In 1955, John McCarthy of Dartmouth College convinced Minsky, Claude Shannon, and Nathaniel Rochester to help him bring together U.S. researchers interested in automata theory, neural nets, and the study of intelligence. They organized a two-month workshop at Dartmouth in the summer of 1956. There were 10 attendees in all, including Allen Newell and Herbert Simon from Carnegie Tech,[11] Trenchard More from Princeton, Arthur Samuel from IBM, and Ray Solomonoff and Oliver Selfridge from MIT. The proposal states:[12]

> We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

Despite this optimistic prediction, the Dartmouth workshop did not lead to any breakthroughs. Newell and Simon presented perhaps the most mature work, a mathematical theorem-proving system called the Logic Theorist (LT). Simon claimed, "We have invented a computer program capable of thinking non-numerically, and thereby solved the venerable mind–body problem."[13] Soon after the workshop, the program was able to prove most of the theorems in Chapter 2 of Russell and Whitehead's *Principia Mathematica*. Russell was reportedly delighted when told that LT had come up with a proof for one theorem that was shorter than the one in *Principia*. The editors of the *Journal of Symbolic Logic* were less impressed; they rejected a paper coauthored by Newell, Simon, and Logic Theorist.

1.3.2 Early enthusiasm, great expectations (1952–1969)

The intellectual establishment of the 1950s, by and large, preferred to believe that "a machine can never do X." (See Chapter 28 for a long list of X's gathered by Turing.) AI researchers naturally responded by demonstrating one X after another. They focused in particular on tasks considered indicative of intelligence in humans, including games, puzzles, mathematics, and IQ tests. John McCarthy referred to this period as the "Look, Ma, no hands!" era.

---

[11] Now Carnegie Mellon University (CMU).

[12] This was the first official usage of McCarthy's term *artificial intelligence*. Perhaps "computational rationality" would have been more precise and less threatening, but "AI" has stuck. At the 50th anniversary of the Dartmouth conference, McCarthy stated that he resisted the terms "computer" or "computational" in deference to Norbert Wiener, who was promoting analog cybernetic devices rather than digital computers.

[13] Newell and Simon also invented a list-processing language, IPL, to write LT. They had no compiler and translated it into machine code by hand. To avoid errors, they worked in parallel, calling out binary numbers to each other as they wrote each instruction to make sure they agreed.

Newell and Simon followed up their success with LT with the General Problem Solver, or GPS. Unlike LT, this program was designed from the start to imitate human problem-solving protocols. Within the limited class of puzzles it could handle, it turned out that the order in which the program considered subgoals and possible actions was similar to that in which humans approached the same problems. Thus, GPS was probably the first program to embody the "thinking humanly" approach. The success of GPS and subsequent programs as models of cognition led Newell and Simon (1976) to formulate the famous **physical symbol system** hypothesis, which states that "a physical symbol system has the necessary and sufficient means for general intelligent action." What they meant is that any system (human or machine) exhibiting intelligence must operate by manipulating data structures composed of symbols. We will see later that this hypothesis has been challenged from many directions.

Physical symbol system

At IBM, Nathaniel Rochester and his colleagues produced some of the first AI programs. Herbert Gelernter (1959) constructed the Geometry Theorem Prover, which was able to prove theorems that many students of mathematics would find quite tricky. This work was a precursor of modern mathematical theorem provers.

Of all the exploratory work done during this period, perhaps the most influential in the long run was that of Arthur Samuel on checkers (draughts). Using methods that we now call reinforcement learning (see Chapter 23), Samuel's programs learned to play at a strong amateur level. He thereby disproved the idea that computers can do only what they are told to: his program quickly learned to play a better game than its creator. The program was demonstrated on television in 1956, creating a strong impression. Like Turing, Samuel had trouble finding computer time. Working at night, he used machines that were still on the testing floor at IBM's manufacturing plant. Samuel's program was the precursor of later systems such as TD-GAMMON (Tesauro, 1992), which was among the world's best backgammon players, and ALPHAGO (Silver *et al.*, 2016), which shocked the world by defeating the human world champion at Go (see Chapter 6).

In 1958, John McCarthy made two important contributions to AI. In MIT AI Lab Memo No. 1, he defined the high-level language **Lisp**, which was to become the dominant AI programming language for the next 30 years. In a paper entitled *Programs with Common Sense*, he advanced a conceptual proposal for AI systems based on knowledge and reasoning. The paper describes the Advice Taker, a hypothetical program that would embody general knowledge of the world and could use it to derive plans of action. The concept was illustrated with simple logical axioms that suffice to generate a plan to drive to the airport. The program was also designed to accept new axioms in the normal course of operation, thereby allowing it to achieve competence in new areas *without being reprogrammed*. The Advice Taker thus embodied the central principles of knowledge representation and reasoning: that it is useful to have a formal, explicit representation of the world and its workings and to be able to manipulate that representation with deductive processes. The paper influenced the course of AI and remains relevant today.

Lisp

1958 also marked the year that Marvin Minsky moved to MIT. His initial collaboration with McCarthy did not last, however. McCarthy stressed representation and reasoning in formal logic, whereas Minsky was more interested in getting programs to work and eventually developed an anti-logic outlook. In 1963, McCarthy started the AI lab at Stanford. His plan to use logic to build the ultimate Advice Taker was advanced by J. A. Robinson's discovery in 1965 of the resolution method (a complete theorem-proving algorithm for first-order

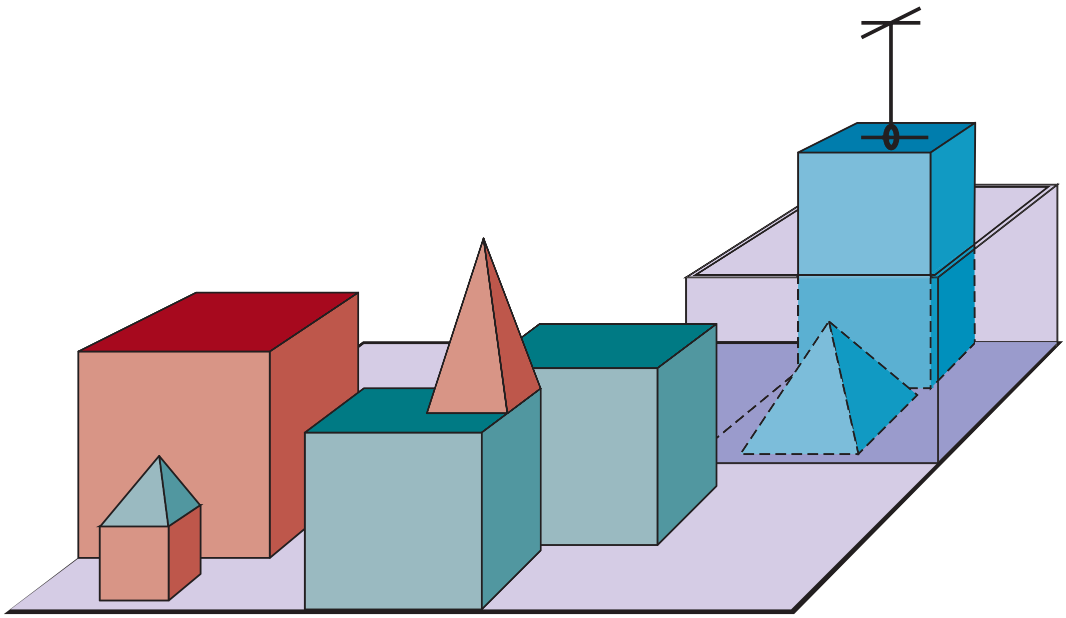

Figure 1.3 A scene from the blocks world. SHRDLU (Winograd, 1972) has just completed the command "Find a block which is taller than the one you are holding and put it in the box."

logic; see Chapter 9). Work at Stanford emphasized general-purpose methods for logical reasoning. Applications of logic included Cordell Green's question-answering and planning systems (Green, 1969b) and the Shakey robotics project at the Stanford Research Institute (SRI). The latter project, discussed further in Chapter 26, was the first to demonstrate the complete integration of logical reasoning and physical activity.

Microworld

At MIT, Minsky supervised a series of students who chose limited problems that appeared to require intelligence to solve. These limited domains became known as **microworlds**. James Slagle's SAINT program (1963) was able to solve closed-form calculus integration problems typical of first-year college courses. Tom Evans's ANALOGY program (1968) solved geometric analogy problems that appear in IQ tests. Daniel Bobrow's STUDENT program (1967) solved algebra story problems, such as the following:

> If the number of customers Tom gets is twice the square of 20 percent of the number of advertisements he runs, and the number of advertisements he runs is 45, what is the number of customers Tom gets?

Blocks world

The most famous microworld is the **blocks world**, which consists of a set of solid blocks placed on a tabletop (or more often, a simulation of a tabletop), as shown in Figure 1.3. A typical task in this world is to rearrange the blocks in a certain way, using a robot hand that can pick up one block at a time. The blocks world was home to the vision project of David Huffman (1971), the vision and constraint-propagation work of David Waltz (1975), the learning theory of Patrick Winston (1970), the natural-language-understanding program of Terry Winograd (1972), and the planner of Scott Fahlman (1974).

Early work building on the neural networks of McCulloch and Pitts also flourished. The work of Shmuel Winograd and Jack Cowan (1963) showed how a large number of elements could collectively represent an individual concept, with a corresponding increase in robustness and parallelism. Hebb's learning methods were enhanced by Bernie Widrow (Widrow and Hoff, 1960; Widrow, 1962), who called his networks **adalines**, and by Frank Rosenblatt (1962) with his **perceptrons**. The **perceptron convergence theorem** (Block *et al.*, 1962) says that the learning algorithm can adjust the connection strengths of a perceptron to match any input data, provided such a match exists.

1.3.3 A dose of reality (1966–1973)

From the beginning, AI researchers were not shy about making predictions of their coming successes. The following statement by Herbert Simon in 1957 is often quoted:

> It is not my aim to surprise or shock you—but the simplest way I can summarize is to say that there are now in the world machines that think, that learn and that create. Moreover, their ability to do these things is going to increase rapidly until—in a visible future—the range of problems they can handle will be coextensive with the range to which the human mind has been applied.

The term "visible future" is vague, but Simon also made more concrete predictions: that within 10 years a computer would be chess champion and a significant mathematical theorem would be proved by machine. These predictions came true (or approximately true) within 40 years rather than 10. Simon's overconfidence was due to the promising performance of early AI systems on simple examples. In almost all cases, however, these early systems failed on more difficult problems.

There were two main reasons for this failure. The first was that many early AI systems were based primarily on "informed introspection" as to how humans perform a task, rather than on a careful analysis of the task, what it means to be a solution, and what an algorithm would need to do to reliably produce such solutions.

The second reason for failure was a lack of appreciation of the intractability of many of the problems that AI was attempting to solve. Most of the early problem-solving systems worked by trying out different combinations of steps until the solution was found. This strategy worked initially because microworlds contained very few objects and hence very few possible actions and very short solution sequences. Before the theory of computational complexity was developed, it was widely thought that "scaling up" to larger problems was simply a matter of faster hardware and larger memories. The optimism that accompanied the development of resolution theorem proving, for example, was soon dampened when researchers failed to prove theorems involving more than a few dozen facts. *The fact that a program can find a solution in principle does not mean that the program contains any of the mechanisms needed to find it in practice.*

The illusion of unlimited computational power was not confined to problem-solving programs. Early experiments in **machine evolution** (now called **genetic programming**) (Friedberg, 1958; Friedberg *et al.*, 1959) were based on the undoubtedly correct belief that by making an appropriate series of small mutations to a machine-code program, one can generate a program with good performance for any particular task. The idea, then, was to try random mutations with a selection process to preserve mutations that seemed useful. Despite thousands of hours of CPU time, almost no progress was demonstrated.

Machine evolution

Failure to come to grips with the "combinatorial explosion" was one of the main criticisms of AI contained in the Lighthill report (Lighthill, 1973), which formed the basis for the decision by the British government to end support for AI research in all but two universities. (Oral tradition paints a somewhat different and more colorful picture, with political ambitions and personal animosities whose description is beside the point.)

A third difficulty arose because of some fundamental limitations on the basic structures being used to generate intelligent behavior. For example, Minsky and Papert's book *Perceptrons* (1969) proved that, although perceptrons (a simple form of neural network) could be shown to learn anything they were capable of representing, they could represent very little. In particular, a two-input perceptron could not be trained to recognize when its two inputs were different. Although their results did not apply to more complex, multilayer networks, research funding for neural-net research soon dwindled to almost nothing. Ironically, the new back-propagation learning algorithms that were to cause an enormous resurgence in neural-net research in the late 1980s and again in the 2010s had already been developed in other contexts in the early 1960s (Kelley, 1960; Bryson, 1962).

1.3.4 Expert systems (1969–1986)

The picture of problem solving that had arisen during the first decade of AI research was of a general-purpose search mechanism trying to string together elementary reasoning steps to find complete solutions. Such approaches have been called **weak methods** because, although general, they do not scale up to large or difficult problem instances. The alternative to weak methods is to use more powerful, domain-specific knowledge that allows larger reasoning steps and can more easily handle typically occurring cases in narrow areas of expertise. One might say that to solve a hard problem, you have to almost know the answer already.

Weak method

The DENDRAL program (Buchanan *et al.*, 1969) was an early example of this approach. It was developed at Stanford, where Ed Feigenbaum (a former student of Herbert Simon), Bruce Buchanan (a philosopher turned computer scientist), and Joshua Lederberg (a Nobel laureate geneticist) teamed up to solve the problem of inferring molecular structure from the information provided by a mass spectrometer. The input to the program consists of the elementary formula of the molecule (e.g., $C_6H_{13}NO_2$) and the mass spectrum giving the masses of the various fragments of the molecule generated when it is bombarded by an electron beam. For example, the mass spectrum might contain a peak at $m = 15$, corresponding to the mass of a methyl (CH_3) fragment.

The naive version of the program generated all possible structures consistent with the formula, and then predicted what mass spectrum would be observed for each, comparing this with the actual spectrum. As one might expect, this is intractable for even moderate-sized molecules. The DENDRAL researchers consulted analytical chemists and found that they worked by looking for well-known patterns of peaks in the spectrum that suggested common substructures in the molecule. For example, the following rule is used to recognize a ketone (C=O) subgroup (which weighs 28):

> **if** M is the mass of the whole molecule and there are two peaks at x_1 and x_2 such that
> (a) $x_1 + x_2 = M + 28$; (b) $x_1 - 28$ is a high peak; (c) $x_2 - 28$ is a high peak; and
> (d) At least one of x_1 and x_2 is high
> **then** there is a ketone subgroup.

Recognizing that the molecule contains a particular substructure reduces the number of possible candidates enormously. According to its authors, DENDRAL was powerful because it embodied the relevant knowledge of mass spectroscopy not in the form of first principles but in efficient "cookbook recipes" (Feigenbaum *et al.*, 1971). The significance of DENDRAL was that it was the first successful *knowledge-intensive* system: its expertise derived from large numbers of special-purpose rules. In 1971, Feigenbaum and others at Stanford began the Heuristic Programming Project (HPP) to investigate the extent to which the new methodology of **expert systems** could be applied to other areas.

Expert systems

The next major effort was the MYCIN system for diagnosing blood infections. With about 450 rules, MYCIN was able to perform as well as some experts, and considerably better than junior doctors. It also contained two major differences from DENDRAL. First, unlike the DENDRAL rules, no general theoretical model existed from which the MYCIN rules could be deduced. They had to be acquired from extensive interviewing of experts. Second, the rules had to reflect the uncertainty associated with medical knowledge. MYCIN incorporated a calculus of uncertainty called **certainty factors** (see Chapter 13), which seemed (at the time) to fit well with how doctors assessed the impact of evidence on the diagnosis.

Certainty factor

The first successful commercial expert system, R1, began operation at the Digital Equipment Corporation (McDermott, 1982). The program helped configure orders for new computer systems; by 1986, it was saving the company an estimated $40 million a year. By 1988, DEC's AI group had 40 expert systems deployed, with more on the way. DuPont had 100 in use and 500 in development. Nearly every major U.S. corporation had its own AI group and was either using or investigating expert systems.

The importance of domain knowledge was also apparent in the area of natural language understanding. Despite the success of Winograd's SHRDLU system, its methods did not extend to more general tasks: for problems such as ambiguity resolution it used simple rules that relied on the tiny scope of the blocks world.

Several researchers, including Eugene Charniak at MIT and Roger Schank at Yale, suggested that robust language understanding would require general knowledge about the world and a general method for using that knowledge. (Schank went further, claiming, "There is no such thing as syntax," which upset a lot of linguists but did serve to start a useful discussion.) Schank and his students built a series of programs (Schank and Abelson, 1977; Wilensky, 1978; Schank and Riesbeck, 1981) that all had the task of understanding natural language. The emphasis, however, was less on language *per se* and more on the problems of representing and reasoning with the knowledge required for language understanding.

The widespread growth of applications to real-world problems led to the development of a wide range of representation and reasoning tools. Some were based on logic—for example, the Prolog language became popular in Europe and Japan, and the PLANNER family in the United States. Others, following Minsky's idea of **frames** (1975), adopted a more structured approach, assembling facts about particular object and event types and arranging the types into a large taxonomic hierarchy analogous to a biological taxonomy.

Frames

In 1981, the Japanese government announced the "Fifth Generation" project, a 10-year plan to build massively parallel, intelligent computers running Prolog. The budget was to exceed a $1.3 billion in today's money. In response, the United States formed the Microelectronics and Computer Technology Corporation (MCC), a consortium designed to assure national competitiveness. In both cases, AI was part of a broad effort, including chip design and human-interface research. In Britain, the Alvey report reinstated the funding removed by the Lighthill report. However, none of these projects ever met its ambitious goals in terms of new AI capabilities or economic impact.

Overall, the AI industry boomed from a few million dollars in 1980 to billions of dollars in 1988, including hundreds of companies building expert systems, vision systems, robots, and software and hardware specialized for these purposes.

Soon after that came a period called the "AI winter," in which many companies fell by the wayside as they failed to deliver on extravagant promises. It turned out to be difficult to build and maintain expert systems for complex domains, in part because the reasoning methods used by the systems broke down in the face of uncertainty and in part because the systems could not learn from experience.

1.3.5 The return of neural networks (1986–present)

In the mid-1980s at least four different groups reinvented the **back-propagation** learning algorithm first developed in the early 1960s. The algorithm was applied to many learning problems in computer science and psychology, and the widespread dissemination of the results in the collection *Parallel Distributed Processing* (Rumelhart and McClelland, 1986) caused great excitement.

Connectionist

These so-called **connectionist** models were seen by some as direct competitors both to the symbolic models promoted by Newell and Simon and to the logicist approach of McCarthy and others. It might seem obvious that at some level humans manipulate symbols—in fact, the anthropologist Terrence Deacon's book *The Symbolic Species* (1997) suggests that this is the *defining characteristic* of humans. Against this, Geoff Hinton, a leading figure in the resurgence of neural networks in the 1980s and 2010s, has described symbols as the "luminiferous aether of AI"—a reference to the non-existent medium through which many 19th-century physicists believed that electromagnetic waves propagated. Certainly, many concepts that we name in language fail, on closer inspection, to have the kind of logically defined necessary and sufficient conditions that early AI researchers hoped to capture in axiomatic form. It may be that connectionist models form internal concepts in a more fluid and imprecise way that is better suited to the messiness of the real world. They also have the capability to learn from examples—they can compare their predicted output value to the true value on a problem and modify their parameters to decrease the difference, making them more likely to perform well on future examples.

1.3.6 Probabilistic reasoning and machine learning (1987–present)

The brittleness of expert systems led to a new, more scientific approach incorporating probability rather than Boolean logic, machine learning rather than hand-coding, and experimental results rather than philosophical claims.[14] It became more common to build on existing theories than to propose brand-new ones, to base claims on rigorous theorems or solid experimental methodology (Cohen, 1995) rather than on intuition, and to show relevance to real-world applications rather than toy examples.

Shared benchmark problem sets became the norm for demonstrating progress, including the UC Irvine repository for machine learning data sets, the International Planning Compe-

[14] Some have characterized this change as a victory of the **neats**—those who think that AI theories should be grounded in mathematical rigor—over the **scruffies**—those who would rather try out lots of ideas, write some programs, and then assess what seems to be working. Both approaches are important. A shift toward neatness implies that the field has reached a level of stability and maturity. The present emphasis on deep learning may represent a resurgence of the scruffies.

tition for planning algorithms, the LibriSpeech corpus for speech recognition, the MNIST data set for handwritten digit recognition, ImageNet and COCO for image object recognition, SQuAD for natural language question answering, the WMT competition for machine translation, and the International SAT Competitions for Boolean satisfiability solvers.

AI was founded in part as a rebellion against the limitations of existing fields like control theory and statistics, but in this period it embraced the positive results of those fields. As David McAllester (1998) put it:

> In the early period of AI it seemed plausible that new forms of symbolic computation, e.g., frames and semantic networks, made much of classical theory obsolete. This led to a form of isolationism in which AI became largely separated from the rest of computer science. This isolationism is currently being abandoned. There is a recognition that machine learning should not be isolated from information theory, that uncertain reasoning should not be isolated from stochastic modeling, that search should not be isolated from classical optimization and control, and that automated reasoning should not be isolated from formal methods and static analysis.

The field of speech recognition illustrates the pattern. In the 1970s, a wide variety of different architectures and approaches were tried. Many of these were rather ad hoc and fragile, and worked on only a few carefully selected examples. In the 1980s, approaches using **hidden Markov models** (HMMs) came to dominate the area. Two aspects of HMMs are relevant. First, they are based on a rigorous mathematical theory. This allowed speech researchers to build on several decades of mathematical results developed in other fields. Second, they are generated by a process of training on a large corpus of real speech data. This ensures that the performance is robust, and in rigorous blind tests HMMs improved their scores steadily. As a result, speech technology and the related field of handwritten character recognition made the transition to widespread industrial and consumer applications. Note that there was no scientific claim that humans use HMMs to recognize speech; rather, HMMs provided a mathematical framework for understanding and solving the problem. We will see in Section 1.3.8, however, that deep learning has rather upset this comfortable narrative.

Hidden Markov models

1988 was an important year for the connection between AI and other fields, including statistics, operations research, decision theory, and control theory. Judea Pearl's (1988) *Probabilistic Reasoning in Intelligent Systems* led to a new acceptance of probability and decision theory in AI. Pearl's development of **Bayesian networks** yielded a rigorous and efficient formalism for representing uncertain knowledge as well as practical algorithms for probabilistic reasoning. Chapters 12, 13, 14, 15, and 18 cover this area, in addition to more recent developments that have greatly increased the expressive power of probabilistic formalisms; Chapter 21 describes methods for learning Bayesian networks and related models from data.

Bayesian network

A second major contribution in 1988 was Rich Sutton's work connecting reinforcement learning—which had been used in Arthur Samuel's checker-playing program in the 1950s—to the theory of Markov decision processes (MDPs) developed in the field of operations research. A flood of work followed connecting AI planning research to MDPs, and the field of reinforcement learning found applications in robotics and process control as well as acquiring deep theoretical foundations.

One consequence of AI's newfound appreciation for data, statistical modeling, optimization, and machine learning was the gradual reunification of subfields such as computer vision, robotics, speech recognition, multiagent systems, and natural language processing that had become somewhat separate from core AI. The process of reintegration has yielded significant benefits both in terms of applications—for example, the deployment of practical robots expanded greatly during this period—and in a better theoretical understanding of the core problems of AI.

1.3.7 Big data (2001–present)

Big data

Remarkable advances in computing power and the creation of the World Wide Web have facilitated the creation of very large data sets—a phenomenon sometimes known as **big data**. These data sets include trillions of words of text, billions of images, and billions of hours of speech and video, as well as vast amounts of genomic data, vehicle tracking data, clickstream data, social network data, and so on.

This has led to the development of learning algorithms specially designed to take advantage of very large data sets. Often, the vast majority of examples in such data sets are *unlabeled*; for example, in Yarowsky's (1995) influential work on word-sense disambiguation, occurrences of a word such as "plant" are not labeled in the data set to indicate whether they refer to flora or factory. With large enough data sets, however, suitable learning algorithms can achieve an accuracy of over 96% on the task of identifying which sense was intended in a sentence. Moreover, Banko and Brill (2001) argued that the improvement in performance obtained from increasing the size of the data set by two or three orders of magnitude outweighs any improvement that can be obtained from tweaking the algorithm.

A similar phenomenon seems to occur in computer vision tasks such as filling in holes in photographs—holes caused either by damage or by the removal of ex-friends. Hays and Efros (2007) developed a clever method for doing this by blending in pixels from similar images; they found that the technique worked poorly with a database of only thousands of images but crossed a threshold of quality with millions of images. Soon after, the availability of tens of millions of images in the ImageNet database (Deng *et al.*, 2009) sparked a revolution in the field of computer vision.

The availability of big data and the shift towards machine learning helped AI recover commercial attractiveness (Havenstein, 2005; Halevy *et al.*, 2009). Big data was a crucial factor in the 2011 victory of IBM's Watson system over human champions in the Jeopardy! quiz game, an event that had a major impact on the public's perception of AI.

1.3.8 Deep learning (2011–present)

Deep learning

The term **deep learning** refers to machine learning using multiple layers of simple, adjustable computing elements. Experiments were carried out with such networks as far back as the 1970s, and in the form of **convolutional neural networks** they found some success in handwritten digit recognition in the 1990s (LeCun *et al.*, 1995). It was not until 2011, however, that deep learning methods really took off. This occurred first in speech recognition and then in visual object recognition.

In the 2012 ImageNet competition, which required classifying images into one of a thousand categories (armadillo, bookshelf, corkscrew, etc.), a deep learning system created in Geoffrey Hinton's group at the University of Toronto (Krizhevsky *et al.*, 2013) demonstrated a dramatic improvement over previous systems, which were based largely on handcrafted features. Since then, deep learning systems have exceeded human performance on some vision tasks (and lag behind in some other tasks). Similar gains have been reported in speech recognition, machine translation, medical diagnosis, and game playing. The use of a deep network to represent the evaluation function contributed to ALPHAGO's victories over the leading human Go players (Silver *et al.*, 2016, 2017, 2018).

These remarkable successes have led to a resurgence of interest in AI among students, companies, investors, governments, the media, and the general public. It seems that every week there is news of a new AI application approaching or exceeding human performance, often accompanied by speculation of either accelerated success or a new AI winter.

Deep learning relies heavily on powerful hardware. Whereas a standard computer CPU can do 10^9 or 10^{10} operations per second. a deep learning algorithm running on specialized hardware (e.g., GPU, TPU, or FPGA) might consume between 10^{14} and 10^{17} operations per second, mostly in the form of highly parallelized matrix and vector operations. Of course, deep learning also depends on the availability of large amounts of training data, and on a few algorithmic tricks (see Chapter 22).

1.4 The State of the Art

Stanford University's One Hundred Year Study on AI (also known as AI100) convenes panels of experts to provide reports on the state of the art in AI. Their 2016 report (Stone *et al.*, 2016; Grosz and Stone, 2018) concludes that "Substantial increases in the future uses of AI applications, including more self-driving cars, healthcare diagnostics and targeted treatment, and physical assistance for elder care can be expected" and that "Society is now at a crucial juncture in determining how to deploy AI-based technologies in ways that promote rather than hinder democratic values such as freedom, equality, and transparency." AI100 also produces an **AI Index** at `aiindex.org` to help track progress. Some highlights from the 2018 and 2019 reports (comparing to a year 2000 baseline unless otherwise stated): AI Index

- Publications: AI papers increased 20-fold between 2010 and 2019 to about 20,000 a year. The most popular category was machine learning. (Machine learning papers in arXiv.org doubled every year from 2009 to 2017.) Computer vision and natural language processing were the next most popular.
- Sentiment: About 70% of news articles on AI are neutral, but articles with positive tone increased from 12% in 2016 to 30% in 2018. The most common issues are ethical: data privacy and algorithm bias.
- Students: Course enrollment increased 5-fold in the U.S. and 16-fold internationally from a 2010 baseline. AI is the most popular specialization in Computer Science.
- Diversity: AI Professors worldwide are about 80% male, 20% female. Similar numbers hold for Ph.D. students and industry hires.
- Conferences: Attendance at NeurIPS increased 800% since 2012 to 13,500 attendees. Other conferences are seeing annual growth of about 30%.
- Industry: AI startups in the U.S. increased 20-fold to over 800.
- Internationalization: China publishes more papers per year than the U.S. and about as many as all of Europe. However, in citation-weighted impact, U.S. authors are 50% ahead of Chinese authors. Singapore, Brazil, Australia, Canada, and India are the fastest growing countries in terms of the number of AI hires.

- Vision: Error rates for object detection (as achieved in LSVRC, the Large-Scale Visual Recognition Challenge) improved from 28% in 2010 to 2% in 2017, exceeding human performance. Accuracy on open-ended visual question answering (VQA) improved from 55% to 68% since 2015, but lags behind human performance at 83%.
- Speed: Training time for the image recognition task dropped by a factor of 100 in just the past two years. The amount of computing power used in top AI applications is doubling every 3.4 months.
- Language: Accuracy on question answering, as measured by F1 score on the Stanford Question Answering Dataset (SQuAD), increased from 60 to 95 from 2015 to 2019; on the SQuAD 2 variant, progress was faster, going from 62 to 90 in just one year. Both scores exceed human-level performance.
- Human benchmarks: By 2019, AI systems had reportedly met or exceeded human-level performance in chess, Go, poker, Pac-Man, Jeopardy!, ImageNet object detection, speech recognition in a limited domain, Chinese-to-English translation in a restricted domain, Quake III, Dota 2, StarCraft II, various Atari games, skin cancer detection, prostate cancer detection, protein folding, and diabetic retinopathy diagnosis.

When (if ever) will AI systems achieve human-level performance across a broad variety of tasks? Ford (2018) interviews AI experts and finds a wide range of target years, from 2029 to 2200, with a mean of 2099. In a similar survey (Grace *et al.*, 2017) 50% of respondents thought this could happen by 2066, although 10% thought it could happen as early as 2025, and a few said "never." The experts were also split on whether we need fundamental new breakthroughs or just refinements on current approaches. But don't take their predictions too seriously; as Philip Tetlock (2017) demonstrates in the area of predicting world events, experts are no better than amateurs.

How will future AI systems operate? We can't yet say. As detailed in this section, the field has adopted several stories about itself—first the bold idea that intelligence by a machine was even possible, then that it could be achieved by encoding expert knowledge into logic, then that probabilistic models of the world would be the main tool, and most recently that machine learning would induce models that might not be based on any well-understood theory at all. The future will reveal what model comes next.

What can AI do today? Perhaps not as much as some of the more optimistic media articles might lead one to believe, but still a great deal. Here are some examples:

Robotic vehicles: The history of robotic vehicles stretches back to radio-controlled cars of the 1920s, but the first demonstrations of autonomous road driving without special guides occurred in the 1980s (Kanade *et al.*, 1986; Dickmanns and Zapp, 1987). After successful demonstrations of driving on dirt roads in the 132-mile DARPA Grand Challenge in 2005 (Thrun, 2006) and on streets with traffic in the 2007 Urban Challenge, the race to develop self-driving cars began in earnest. In 2018, Waymo test vehicles passed the landmark of 10 million miles driven on public roads without a serious accident, with the human driver stepping in to take over control only once every 6,000 miles. Soon after, the company began offering a commercial robotic taxi service.

In the air, autonomous fixed-wing drones have been providing cross-country blood deliveries in Rwanda since 2016. Quadcopters perform remarkable aerobatic maneuvers, explore buildings while constructing 3-D maps, and self-assemble into autonomous formations.

Legged locomotion: BigDog, a quadruped robot by Raibert *et al.* (2008), upended our notions of how robots move—no longer the slow, stiff-legged, side-to-side gait of Hollywood movie robots, but something closely resembling an animal and able to recover when shoved or when slipping on an icy puddle. Atlas, a humanoid robot, not only walks on uneven terrain but jumps onto boxes and does backflips (Ackerman and Guizzo, 2016).

Autonomous planning and scheduling: A hundred million miles from Earth, NASA's Remote Agent program became the first on-board autonomous planning program to control the scheduling of operations for a spacecraft (Jonsson *et al.*, 2000). Remote Agent generated plans from high-level goals specified from the ground and monitored the execution of those plans—detecting, diagnosing, and recovering from problems as they occurred. Today, the EUROPA planning toolkit (Barreiro *et al.*, 2012) is used for daily operations of NASA's Mars rovers and the SEXTANT system (Winternitz, 2017) allows autonomous navigation in deep space, beyond the global GPS system.

During the Persian Gulf crisis of 1991, U.S. forces deployed a Dynamic Analysis and Replanning Tool, DART (Cross and Walker, 1994), to do automated logistics planning and scheduling for transportation. This involved up to 50,000 vehicles, cargo, and people at a time, and had to account for starting points, destinations, routes, transport capacities, port and airfield capacities, and conflict resolution among all parameters. The Defense Advanced Research Project Agency (DARPA) stated that this single application more than paid back DARPA's 30-year investment in AI.

Every day, ride hailing companies such as Uber and mapping services such as Google Maps provide driving directions for hundreds of millions of users, quickly plotting an optimal route taking into account current and predicted future traffic conditions.

Machine translation: Online machine translation systems now enable the reading of documents in over 100 languages, including the native languages of over 99% of humans, and render hundreds of billions of words per day for hundreds of millions of users. While not perfect, they are generally adequate for understanding. For closely related languages with a great deal of training data (such as French and English) translations within a narrow domain are close to the level of a human (Wu *et al.*, 2016b).

Speech recognition: In 2017, Microsoft showed that its Conversational Speech Recognition System had reached a word error rate of 5.1%, matching human performance on the Switchboard task, which involves transcribing telephone conversations (Xiong *et al.*, 2017). About a third of computer interaction worldwide is now done by voice rather than keyboard; Skype provides real-time speech-to-speech translation in ten languages. Alexa, Siri, Cortana, and Google offer assistants that can answer questions and carry out tasks for the user; for example the Google Duplex service uses speech recognition and speech synthesis to make restaurant reservations for users, carrying out a fluent conversation on their behalf.

Recommendations: Companies such as Amazon, Facebook, Netflix, Spotify, YouTube, Walmart, and others use machine learning to recommend what you might like based on your past experiences and those of others like you. The field of recommender systems has a long history (Resnick and Varian, 1997) but is changing rapidly due to new deep learning methods that analyze content (text, music, video) as well as history and metadata (van den Oord *et al.*, 2014; Zhang *et al.*, 2017). Spam filtering can also be considered a form of recommendation (or dis-recommendation); current AI techniques filter out over 99.9% of spam, and email services can also recommend potential recipients, as well as possible response text.

Game playing: When Deep Blue defeated world chess champion Garry Kasparov in 1997, defenders of human supremacy placed their hopes on Go. Piet Hut, an astrophysicist and Go enthusiast, predicted that it would take “a hundred years before a computer beats humans at Go—maybe even longer.” But just 20 years later, ALPHAGO surpassed all human players (Silver *et al.*, 2017). Ke Jie, the world champion, said, “Last year, it was still quite human-like when it played. But this year, it became like a god of Go.” ALPHAGO benefited from studying hundreds of thousands of past games by human Go players, and from the distilled knowledge of expert Go players that worked on the team.

A followup program, ALPHAZERO, used no input from humans (except for the rules of the game), and was able to learn through self-play alone to defeat all opponents, human and machine, at Go, chess, and shogi (Silver *et al.*, 2018). Meanwhile, human champions have been beaten by AI systems at games as diverse as Jeopardy! (Ferrucci *et al.*, 2010), poker (Bowling *et al.*, 2015; Moravčík *et al.*, 2017; Brown and Sandholm, 2019), and the video games Dota 2 (Fernandez and Mahlmann, 2018), StarCraft II (Vinyals *et al.*, 2019), and Quake III (Jaderberg *et al.*, 2019).

Image understanding: Not content with exceeding human accuracy on the challenging ImageNet object recognition task, computer vision researchers have taken on the more difficult problem of image captioning. Some impressive examples include “A person riding a motorcycle on a dirt road,” “Two pizzas sitting on top of a stove top oven,” and “A group of young people playing a game of frisbee” (Vinyals *et al.*, 2017b). Current systems are far from perfect, however: a “refrigerator filled with lots of food and drinks” turns out to be a no-parking sign partially obscured by lots of small stickers.

Medicine: AI algorithms now equal or exceed expert doctors at diagnosing many conditions, particularly when the diagnosis is based on images. Examples include Alzheimer’s disease (Ding *et al.*, 2018), metastatic cancer (Liu *et al.*, 2017; Esteva *et al.*, 2017), ophthalmic disease (Gulshan *et al.*, 2016), and skin diseases (Liu *et al.*, 2019c). A systematic review and meta-analysis (Liu *et al.*, 2019a) found that the performance of AI programs, on average, was equivalent to health care professionals. One current emphasis in medical AI is in facilitating human–machine partnerships. For example, the LYNA system achieves 99.6% overall accuracy in diagnosing metastatic breast cancer—better than an unaided human expert—but the combination does better still (Liu *et al.*, 2018; Steiner *et al.*, 2018).

The widespread adoption of these techniques is now limited not by diagnostic accuracy but by the need to demonstrate improvement in clinical outcomes and to ensure transparency, lack of bias, and data privacy (Topol, 2019). In 2017, only two medical AI applications were approved by the FDA, but that increased to 12 in 2018, and continues to rise.

Climate science: A team of scientists won the 2018 Gordon Bell Prize for a deep learning model that discovers detailed information about extreme weather events that were previously buried in climate data. They used a supercomputer with specialized GPU hardware to exceed the exaop level (10^{18} operations per second), the first machine learning program to do so (Kurth *et al.*, 2018). Rolnick *et al.* (2019) present a 60-page catalog of ways in which machine learning can be used to tackle climate change.

These are just a few examples of artificial intelligence systems that exist today. Not magic or science fiction—but rather science, engineering, and mathematics, to which this book provides an introduction.

1.5 Risks and Benefits of AI

Francis Bacon, a philosopher credited with creating the scientific method, noted in *The Wisdom of the Ancients* (1609) that the "mechanical arts are of ambiguous use, serving as well for hurt as for remedy." As AI plays an increasingly important role in the economic, social, scientific, medical, financial, and military spheres, we would do well to consider the hurts and remedies—in modern parlance, the risks and benefits—that it can bring. The topics summarized here are covered in greater depth in Chapters 28 and 29.

To begin with the benefits: put simply, our entire civilization is the product of our human intelligence. If we have access to substantially greater machine intelligence, the ceiling on our ambitions is raised substantially. The potential for AI and robotics to free humanity from menial repetitive work and to dramatically increase the production of goods and services could presage an era of peace and plenty. The capacity to accelerate scientific research could result in cures for disease and solutions for climate change and resource shortages. As Demis Hassabis, CEO of Google DeepMind, has suggested: "First solve AI, then use AI to solve everything else."

Long before we have an opportunity to "solve AI," however, we will incur risks from the misuse of AI, inadvertent or otherwise. Some of these are already apparent, while others seem likely based on current trends:

- *Lethal autonomous weapons*: These are defined by the United Nations as weapons that can locate, select, and eliminate human targets without human intervention. A primary concern with such weapons is their *scalability*: the absence of a requirement for human supervision means that a small group can deploy an arbitrarily large number of weapons against human targets defined by any feasible recognition criterion. The technologies needed for autonomous weapons are similar to those needed for self-driving cars. Informal expert discussions on the potential risks of lethal autonomous weapons began at the UN in 2014, moving to the formal pre-treaty stage of a Group of Governmental Experts in 2017.
- *Surveillance and persuasion*: While it is expensive, tedious, and sometimes legally questionable for security personnel to monitor phone lines, video camera feeds, emails, and other messaging channels, AI (speech recognition, computer vision, and natural language understanding) can be used in a scalable fashion to perform mass surveillance of individuals and detect activities of interest. By tailoring information flows to individuals through social media, based on machine learning techniques, political behavior can be modified and controlled to some extent—a concern that became apparent in elections beginning in 2016.
- *Biased decision making*: Careless or deliberate misuse of machine learning algorithms for tasks such as evaluating parole and loan applications can result in decisions that are biased by race, gender, or other protected categories. Often, the data themselves reflect pervasive bias in society.
- *Impact on employment*: Concerns about machines eliminating jobs are centuries old. The story is never simple: machines do some of the tasks that humans might otherwise do, but they also make humans more productive and therefore more employable, and make companies more profitable and therefore able to pay higher wages. They may render some activities economically viable that would otherwise be impractical. Their

use generally results in increasing wealth but tends to have the effect of shifting wealth from labor to capital, further exacerbating increases in inequality. Previous advances in technology—such as the invention of mechanical looms—have resulted in serious disruptions to employment, but eventually people find new kinds of work to do. On the other hand, it is possible that AI will be doing those new kinds of work too. This topic is rapidly becoming a major focus for economists and governments around the world.

- *Safety-critical applications*: As AI techniques advance, they are increasingly used in high-stakes, safety-critical applications such as driving cars and managing the water supplies of cities. Fatal accidents have already occurred and highlight the difficulty of formal verification and statistical risk analysis for systems developed using machine learning techniques. The field of AI will need to develop technical and ethical standards at least comparable to those prevalent in other engineering and healthcare disciplines where people's lives are at stake.
- *Cybersecurity*: AI techniques are useful in defending against cyberattack, for example by detecting unusual patterns of behavior, but they will also contribute to the potency, survivability, and proliferation capability of malware. For example, reinforcement learning methods have been used to create highly effective tools for automated, personalized blackmail and phishing attacks.

We will revisit these topics in more depth in Section 28.3. As AI systems become more capable, they will take on more of the societal roles previously played by humans. Just as humans have used these roles in the past to perpetrate mischief, we can expect that humans may misuse AI systems in these roles to perpetrate even more mischief. All of the examples given above point to the importance of governance and, eventually, regulation. At present, the research community and the major corporations involved in AI research have developed voluntary self-governance principles for AI-related activities (see Section 28.3). Governments and international organizations are setting up advisory bodies to devise appropriate regulations for each specific use case, to prepare for the economic and social impacts, and to take advantage of AI capabilities to address major societal problems.

What of the longer term? Will we achieve the long-standing goal: the creation of intelligence comparable to or more capable than human intelligence? And, if we do, what then?

For much of AI's history, these questions have been overshadowed by the daily grind of getting AI systems to do anything even remotely intelligent. As with any broad discipline, the great majority of AI researchers have specialized in a specific subfield such as game-playing, knowledge representation, vision, or natural language understanding—often on the assumption that progress in these subfields would contribute to the broader goals of AI. Nils Nilsson (1995), one of the original leaders of the Shakey project at SRI, reminded the field of those broader goals and warned that the subfields were in danger of becoming ends in themselves. Later, some influential founders of AI, including John McCarthy (2007), Marvin Minsky (2007), and Patrick Winston (Beal and Winston, 2009), concurred with Nilsson's warnings, suggesting that instead of focusing on measurable performance in specific applications, AI should return to its roots of striving for, in Herb Simon's words, "machines that think, that learn and that create." They called the effort **human-level AI** or HLAI—a machine should be able to learn to do anything a human can do. Their first symposium was in 2004 (Minsky *et al.*, 2004). Another effort with similar goals, the **artificial general intelligence (AGI)**

Human-level AI

Artificial general intelligence (AGI)

movement (Goertzel and Pennachin, 2007), held its first conference and organized the *Journal of Artificial General Intelligence* in 2008.

Artificial superintelligence (ASI)

At around the same time, concerns were raised that creating **artificial superintelligence** or **ASI**—intelligence that far surpasses human ability—might be a bad idea (Yudkowsky, 2008; Omohundro, 2008). Turing (1996) himself made the same point in a lecture given in Manchester in 1951, drawing on earlier ideas from Samuel Butler (1863):[15]

> It seems probable that once the machine thinking method had started, it would not take long to outstrip our feeble powers. ... At some stage therefore we should have to expect the machines to take control, in the way that is mentioned in Samuel Butler's *Erewhon*.

These concerns have only become more widespread with recent advances in deep learning, the publication of books such as *Superintelligence* by Nick Bostrom (2014), and public pronouncements from Stephen Hawking, Bill Gates, Martin Rees, and Elon Musk.

Gorilla problem

Experiencing a general sense of unease with the idea of creating superintelligent machines is only natural. We might call this the **gorilla problem**: about seven million years ago, a now-extinct primate evolved, with one branch leading to gorillas and one to humans. Today, the gorillas are not too happy about the human branch; they have essentially no control over their future. If this is the result of success in creating superhuman AI—that humans cede control over their future—then perhaps we should stop work on AI, and, as a corollary, give up the benefits it might bring. This is the essence of Turing's warning: it is not obvious that we can control machines that are more intelligent than us.

If superhuman AI were a black box that arrived from outer space, then indeed it would be wise to exercise caution in opening the box. But it is not: *we* design the AI systems, so if they do end up "taking control," as Turing suggests, it would be the result of a design failure.

To avoid such an outcome, we need to understand the source of potential failure. Norbert Wiener (1960), who was motivated to consider the long-term future of AI after seeing Arthur Samuel's checker-playing program learn to beat its creator, had this to say:

> If we use, to achieve our purposes, a mechanical agency with whose operation we cannot interfere effectively ... we had better be quite sure that the purpose put into the machine is the purpose which we really desire.

King Midas problem

Many cultures have myths of humans who ask gods, genies, magicians, or devils for something. Invariably, in these stories, they get what they literally ask for, and then regret it. The third wish, if there is one, is to undo the first two. We will call this the **King Midas problem**: Midas, a legendary King in Greek mythology, asked that everything he touched should turn to gold, but then regretted it after touching his food, drink, and family members.[16]

We touched on this issue in Section 1.1.5, where we pointed out the need for a significant modification to the standard model of putting fixed objectives into the machine. The solution to Wiener's predicament is not to have a definite "purpose put into the machine" at all. Instead, we want machines that strive to achieve human objectives but know that they don't know for certain exactly what those objectives are.

---

[15] Even earlier, in 1847, Richard Thornton, editor of the *Primitive Expounder*, railed against mechanical calculators: "Mind ... outruns itself and does away with the necessity of its own existence by inventing machines to do its own thinking. ... But who knows that such machines when brought to greater perfection, may not think of a plan to remedy all their own defects and then grind out ideas beyond the ken of mortal mind!"

[16] Midas would have done better if he had followed basic principles of safety and included an "undo" button and a "pause" button in his wish.

It is perhaps unfortunate that almost all AI research to date has been carried out within the standard model, which means that almost all of the technical material in this edition reflects that intellectual framework. There are, however, some early results within the new framework. In Chapter 15, we show that a machine has a positive incentive to allow itself to be switched off if and only if it is uncertain about the human objective. In Chapter 17, we formulate and study **assistance games**, which describe mathematically the situation in which a human has an objective and a machine tries to achieve it, but is initially uncertain about what it is. In Chapter 23, we explain the methods of **inverse reinforcement learning** that allow machines to learn more about human preferences from observations of the choices that humans make. In Chapter 28, we explore two of the principal difficulties: first, that our choices depend on our preferences through a very complex cognitive architecture that is hard to invert; and, second, that we humans may not have consistent preferences in the first place—either individually or as a group—so it may not be clear what AI systems *should* be doing for us.

Assistance game

Inverse reinforcement learning

Summary

This chapter defines AI and establishes the cultural background against which it has developed. Some of the important points are as follows:

- Different people approach AI with different goals in mind. Two important questions to ask are: Are you concerned with thinking, or behavior? Do you want to model humans, or try to achieve the optimal results?
- According to what we have called the standard model, AI is concerned mainly with **rational action**. An ideal **intelligent agent** takes the best possible action in a situation. We study the problem of building agents that are intelligent in this sense.
- Two refinements to this simple idea are needed: first, the ability of any agent, human or otherwise, to choose rational actions is limited by the computational intractability of doing so; second, the concept of a machine that pursues a definite objective needs to be replaced with that of a machine pursuing objectives to benefit humans, but uncertain as to what those objectives are.
- Philosophers (going back to 400 BCE) made AI conceivable by suggesting that the mind is in some ways like a machine, that it operates on knowledge encoded in some internal language, and that thought can be used to choose what actions to take.
- Mathematicians provided the tools to manipulate statements of logical certainty as well as uncertain, probabilistic statements. They also set the groundwork for understanding computation and reasoning about algorithms.
- Economists formalized the problem of making decisions that maximize the expected utility to the decision maker.
- Neuroscientists discovered some facts about how the brain works and the ways in which it is similar to and different from computers.
- Psychologists adopted the idea that humans and animals can be considered information-processing machines. Linguists showed that language use fits into this model.
- Computer engineers provided the ever-more-powerful machines that make AI applications possible, and software engineers made them more usable.

- Control theory deals with designing devices that act optimally on the basis of feedback from the environment. Initially, the mathematical tools of control theory were quite different from those used in AI, but the fields are coming closer together.
- The history of AI has had cycles of success, misplaced optimism, and resulting cutbacks in enthusiasm and funding. There have also been cycles of introducing new, creative approaches and systematically refining the best ones.
- AI has matured considerably compared to its early decades, both theoretically and methodologically. As the problems that AI deals with became more complex, the field moved from Boolean logic to probabilistic reasoning, and from hand-crafted knowledge to machine learning from data. This has led to improvements in the capabilities of real systems and greater integration with other disciplines.
- As AI systems find application in the real world, it has become necessary to consider a wide range of risks and ethical consequences.
- In the longer term, we face the difficult problem of controlling superintelligent AI systems that may evolve in unpredictable ways. Solving this problem seems to necessitate a change in our conception of AI.

Bibliographical and Historical Notes

A comprehensive history of AI is given by Nils Nilsson (2009), one of the early pioneers of the field. Pedro Domingos (2015) and Melanie Mitchell (2019) give overviews of machine learning for a general audience, and Kai-Fu Lee (2018) describes the race for international leadership in AI. Martin Ford (2018) interviews 23 leading AI researchers.

The main professional societies for AI are the Association for the Advancement of Artificial Intelligence (AAAI), the ACM Special Interest Group in Artificial Intelligence (SIGAI, formerly SIGART), the European Association for AI, and the Society for Artificial Intelligence and Simulation of Behaviour (AISB). The Partnership on AI brings together many commercial and nonprofit organizations concerned with the ethical and social impacts of AI. AAAI's *AI Magazine* contains many topical and tutorial articles, and its Web site, `aaai.org`, contains news, tutorials, and background information.

The most recent work appears in the proceedings of the major AI conferences: the International Joint Conference on AI (IJCAI), the annual European Conference on AI (ECAI), and the AAAI Conference. Machine learning is covered by the International Conference on Machine Learning and the Neural Information Processing Systems (NeurIPS) meeting. The major journals for general AI are *Artificial Intelligence*, *Computational Intelligence*, the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Intelligent Systems*, and the *Journal of Artificial Intelligence Research*. There are also many conferences and journals devoted to specific areas, which we cover in the appropriate chapters.

CHAPTER 2

INTELLIGENT AGENTS

In which we discuss the nature of agents, perfect or otherwise, the diversity of environments, and the resulting menagerie of agent types.

Chapter 1 identified the concept of **rational agents** as central to our approach to artificial intelligence. In this chapter, we make this notion more concrete. We will see that the concept of rationality can be applied to a wide variety of agents operating in any imaginable environment. Our plan in this book is to use this concept to develop a small set of design principles for building successful agents—systems that can reasonably be called **intelligent**.

We begin by examining agents, environments, and the coupling between them. The observation that some agents behave better than others leads naturally to the idea of a rational agent—one that behaves as well as possible. How well an agent can behave depends on the nature of the environment; some environments are more difficult than others. We give a crude categorization of environments and show how properties of an environment influence the design of suitable agents for that environment. We describe a number of basic "skeleton" agent designs, which we flesh out in the rest of the book.

2.1 Agents and Environments

Environment
Sensor
Actuator

An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**. This simple idea is illustrated in Figure 2.1. A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators. A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators. A software agent receives file contents, network packets, and human input (keyboard/mouse/touchscreen/voice) as sensory inputs and acts on the environment by writing files, sending network packets, and displaying information or generating sounds. The environment could be everything—the entire universe! In practice it is just that part of the universe whose state we care about when designing this agent—the part that affects what the agent perceives and that is affected by the agent's actions.

Percept
Percept sequence

We use the term **percept** to refer to the content an agent's sensors are perceiving. An agent's **percept sequence** is the complete history of everything the agent has ever perceived. In general, *an agent's choice of action at any given instant can depend on its built-in knowledge and on the entire percept sequence observed to date, but not on anything it hasn't perceived.* By specifying the agent's choice of action for every possible percept sequence, we have said more or less everything there is to say about the agent. Mathematically speaking, we say that an agent's behavior is described by the **agent function** that maps any given percept sequence to an action.

Agent function

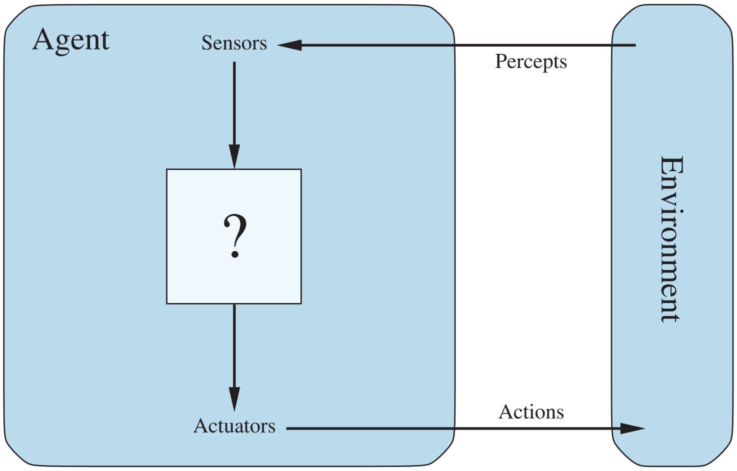

Figure 2.1 Agents interact with environments through sensors and actuators.

We can imagine *tabulating* the agent function that describes any given agent; for most agents, this would be a very large table—infinite, in fact, unless we place a bound on the length of percept sequences we want to consider. Given an agent to experiment with, we can, in principle, construct this table by trying out all possible percept sequences and recording which actions the agent does in response.[1] The table is, of course, an *external* characterization of the agent. *Internally*, the agent function for an artificial agent will be implemented by an **agent program**. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.

Agent program

To illustrate these ideas, we use a simple example—the vacuum-cleaner world, which consists of a robotic vacuum-cleaning agent in a world consisting of squares that can be either dirty or clean. Figure 2.2 shows a configuration with just two squares, A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. The agent starts in square A. The available actions are to move to the right, move to the left, suck up the dirt, or do nothing.[2] One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square. A partial tabulation of this agent function is shown in Figure 2.3 and an agent program that implements it appears in Figure 2.8 on page 67.

Looking at Figure 2.3, we see that various vacuum-world agents can be defined simply by filling in the right-hand column in various ways. The obvious question, then, is this: *What is the right way to fill out the table?* In other words, what makes an agent good or bad, intelligent or stupid? We answer these questions in the next section.

[1] If the agent uses some randomization to choose its actions, then we would have to try each sequence many times to identify the probability of each action. One might imagine that acting randomly is rather silly, but we show later in this chapter that it can be very intelligent.

[2] In a real robot, it would be unlikely to have an actions like "move right" and "move left." Instead the actions would be "spin wheels forward" and "spin wheels backward." We have chosen the actions to be easier to follow on the page, not for ease of implementation in an actual robot.

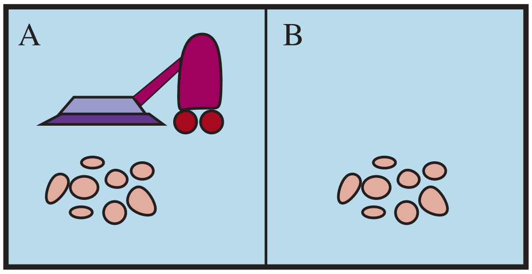

Figure 2.2 A vacuum-cleaner world with just two locations. Each location can be clean or dirty, and the agent can move left or right and can clean the square that it occupies. Different versions of the vacuum world allow for different rules about what the agent can perceive, whether its actions always succeed, and so on.

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean]$, $[A, Clean]$	*Right*
$[A, Clean]$, $[A, Dirty]$	*Suck*
⋮	⋮
$[A, Clean]$, $[A, Clean]$, $[A, Clean]$	*Right*
$[A, Clean]$, $[A, Clean]$, $[A, Dirty]$	*Suck*
⋮	⋮

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2. The agent cleans the current square if it is dirty, otherwise it moves to the other square. Note that the table is of unbounded size unless there is a restriction on the length of possible percept sequences.

Before closing this section, we should emphasize that the notion of an agent is meant to be a tool for analyzing systems, not an absolute characterization that divides the world into agents and non-agents. One could view a hand-held calculator as an agent that chooses the action of displaying “4” when given the percept sequence “2 + 2 =,” but such an analysis would hardly aid our understanding of the calculator. In a sense, all areas of engineering can be seen as designing artifacts that interact with the world; AI operates at (what the authors consider to be) the most interesting end of the spectrum, where the artifacts have significant computational resources and the task environment requires nontrivial decision making.

2.2 Good Behavior: The Concept of Rationality

A **rational agent** is one that does the right thing. Obviously, doing the right thing is better than doing the wrong thing, but what does it mean to do the right thing?

Rational agent

2.2.1 Performance measures

Moral philosophy has developed several different notions of the "right thing," but AI has generally stuck to one notion called **consequentialism**: we evaluate an agent's behavior by its consequences. When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well. This notion of desirability is captured by a **performance measure** that evaluates any given sequence of environment states.

Consequentialism

Performance measure

Humans have desires and preferences of their own, so the notion of rationality as applied to humans has to do with their success in choosing actions that produce sequences of environment states that are desirable *from their point of view*. Machines, on the other hand, do *not* have desires and preferences of their own; the performance measure is, initially at least, in the mind of the designer of the machine, or in the mind of the users the machine is designed for. We will see that some agent designs have an explicit representation of (a version of) the performance measure, while in other designs the performance measure is entirely implicit—the agent may do the right thing, but it doesn't know why.

Recalling Norbert Wiener's warning to ensure that "the purpose put into the machine is the purpose which we really desire" (page 51), notice that it can be quite hard to formulate a performance measure correctly. Consider, for example, the vacuum-cleaner agent from the preceding section. We might propose to measure performance by the amount of dirt cleaned up in a single eight-hour shift. With a rational agent, of course, what you ask for is what you get. A rational agent can maximize this performance measure by cleaning up the dirt, then dumping it all on the floor, then cleaning it up again, and so on. A more suitable performance measure would reward the agent for having a clean floor. For example, one point could be awarded for each clean square at each time step (perhaps with a penalty for electricity consumed and noise generated). *As a general rule, it is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave.*

Even when the obvious pitfalls are avoided, some knotty problems remain. For example, the notion of "clean floor" in the preceding paragraph is based on average cleanliness over time. Yet the same average cleanliness can be achieved by two different agents, one of which does a mediocre job all the time while the other cleans energetically but takes long breaks. Which is preferable might seem to be a fine point of janitorial science, but in fact it is a deep philosophical question with far-reaching implications. Which is better—a reckless life of highs and lows, or a safe but humdrum existence? Which is better—an economy where everyone lives in moderate poverty, or one in which some live in plenty while others are very poor? We leave these questions as an exercise for the diligent reader.

For most of the book, we will assume that the performance measure can be specified correctly. For the reasons given above, however, we must accept the possibility that we might put the wrong purpose into the machine—precisely the King Midas problem described on page 51. Moreover, when designing one piece of software, copies of which will belong to different users, we cannot anticipate the exact preferences of each individual user. Thus, we may need to build agents that reflect initial uncertainty about the true performance measure and learn more about it as time goes by; such agents are described in Chapters 15, 17, and 23.

2.2.2 Rationality

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.

Definition of a rational agent

This leads to a **definition of a rational agent**:

> *For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

Consider the simple vacuum-cleaner agent that cleans a square if it is dirty and moves to the other square if not; this is the agent function tabulated in Figure 2.3. Is this a rational agent? That depends! First, we need to say what the performance measure is, what is known about the environment, and what sensors and actuators the agent has. Let us assume the following:

- The performance measure awards one point for each clean square at each time step, over a "lifetime" of 1000 time steps.
- The "geography" of the environment is known *a priori* (Figure 2.2) but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The *Right* and *Left* actions move the agent one square except when this would take the agent outside the environment, in which case the agent remains where it is.
- The only available actions are *Right*, *Left*, and *Suck*.
- The agent correctly perceives its location and whether that location contains dirt.

Under these circumstances the agent is indeed rational; its expected performance is at least as good as any other agent's.

One can see easily that the same agent would be irrational under different circumstances. For example, once all the dirt is cleaned up, the agent will oscillate needlessly back and forth; if the performance measure includes a penalty of one point for each movement, the agent will fare poorly. A better agent for this case would do nothing once it is sure that all the squares are clean. If clean squares can become dirty again, the agent should occasionally check and re-clean them if needed. If the geography of the environment is unknown, the agent will need to **explore** it. Exercise 2.VACR asks you to design agents for these cases.

2.2.3 Omniscience, learning, and autonomy

Omniscience

We need to be careful to distinguish between rationality and **omniscience**. An omniscient agent knows the *actual* outcome of its actions and can act accordingly; but omniscience is impossible in reality. Consider the following example: I am walking along the Champs Elysées one day and I see an old friend across the street. There is no traffic nearby and I'm not otherwise engaged, so, being rational, I start to cross the street. Meanwhile, at 33,000 feet, a cargo door falls off a passing airliner,[3] and before I make it to the other side of the street I am flattened. Was I irrational to cross the street? It is unlikely that my obituary would read "Idiot attempts to cross street."

This example shows that rationality is not the same as perfection. Rationality maximizes *expected* performance, while perfection maximizes *actual* performance. Retreating from a requirement of perfection is not just a question of being fair to agents. The point is that if we expect an agent to do what turns out after the fact to be the best action, it will be impossible to design an agent to fulfill this specification—unless we improve the performance of crystal balls or time machines.

Our definition of rationality does not require omniscience, then, because the rational choice depends only on the percept sequence *to date*. We must also ensure that we haven't inadvertently allowed the agent to engage in decidedly underintelligent activities. For example, if an agent does not look both ways before crossing a busy road, then its percept sequence will not tell it that there is a large truck approaching at high speed. Does our definition of rationality say that it's now OK to cross the road? Far from it!

First, it would not be rational to cross the road given this uninformative percept sequence: the risk of accident from crossing without looking is too great. Second, a rational agent should choose the "looking" action before stepping into the street, because looking helps maximize the expected performance. Doing actions *in order to modify future percepts*—sometimes called **information gathering**—is an important part of rationality and is covered in depth in Chapter 15. A second example of information gathering is provided by the **exploration** that must be undertaken by a vacuum-cleaning agent in an initially unknown environment.

Information gathering

Learning

Our definition requires a rational agent not only to gather information but also to **learn** as much as possible from what it perceives. The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented. There are extreme cases in which the environment is completely known *a priori* and completely predictable. In such cases, the agent need not perceive or learn; it simply acts correctly.

Of course, such agents are fragile. Consider the lowly dung beetle. After digging its nest and laying its eggs, it fetches a ball of dung from a nearby heap to plug the entrance. If the ball of dung is removed from its grasp *en route*, the beetle continues its task and pantomimes plugging the nest with the nonexistent dung ball, never noticing that it is missing. Evolution has built an assumption into the beetle's behavior, and when it is violated, unsuccessful behavior results.

Slightly more intelligent is the sphex wasp. The female sphex will dig a burrow, go out and sting a caterpillar and drag it to the burrow, enter the burrow again to check all is well, drag the caterpillar inside, and lay its eggs. The caterpillar serves as a food source when the eggs hatch. So far so good, but if an entomologist moves the caterpillar a few inches away while the sphex is doing the check, it will revert to the "drag the caterpillar" step of its plan and will continue the plan without modification, re-checking the burrow, even after dozens of caterpillar-moving interventions. The sphex is unable to learn that its innate plan is failing, and thus will not change it.

---

[3] See N. Henderson, "New door latches urged for Boeing 747 jumbo jets," *Washington Post*, August 24, 1989.

Autonomy

To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts and learning processes, we say that the agent lacks **autonomy**. A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowledge. For example, a vacuum-cleaning agent that learns to predict where and when additional dirt will appear will do better than one that does not.

As a practical matter, one seldom requires complete autonomy from the start: when the agent has had little or no experience, it would have to act randomly unless the designer gave some assistance. Just as evolution provides animals with enough built-in reflexes to survive long enough to learn for themselves, it would be reasonable to provide an artificial intelligent agent with some initial knowledge as well as an ability to learn. After sufficient experience of its environment, the behavior of a rational agent can become effectively *independent* of its prior knowledge. Hence, the incorporation of learning allows one to design a single rational agent that will succeed in a vast variety of environments.

2.3 The Nature of Environments

Task environment

Now that we have a definition of rationality, we are almost ready to think about building rational agents. First, however, we must think about **task environments**, which are essentially the "problems" to which rational agents are the "solutions." We begin by showing how to specify a task environment, illustrating the process with a number of examples. We then show that task environments come in a variety of flavors. The nature of the task environment directly affects the appropriate design for the agent program.

2.3.1 Specifying the task environment

PEAS

In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent's actuators and sensors. We group all these under the heading of the **task environment**. For the acronymically minded, we call this the **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors) description. In designing an agent, the first step must always be to specify the task environment as fully as possible.

The vacuum world was a simple example; let us consider a more complex problem: an automated taxi driver. Figure 2.4 summarizes the PEAS description for the taxi's task environment. We discuss each element in more detail in the following paragraphs.

First, what is the **performance measure** to which we would like our automated driver to aspire? Desirable qualities include getting to the correct destination; minimizing fuel consumption and wear and tear; minimizing the trip time or cost; minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits. Obviously, some of these goals conflict, so tradeoffs will be required.

Next, what is the driving **environment** that the taxi will face? Any taxi driver must deal with a variety of roads, ranging from rural lanes and urban alleys to 12-lane freeways. The roads contain other traffic, pedestrians, stray animals, road works, police cars, puddles, and potholes. The taxi must also interact with potential and actual passengers. There are also some optional choices. The taxi might need to operate in Southern California, where snow is seldom a problem, or in Alaska, where it seldom is not. It could always be driving on the right, or we might want it to be flexible enough to drive on the left when in Britain or Japan. Obviously, the more restricted the environment, the easier the design problem.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

Figure 2.4 PEAS description of the task environment for an automated taxi driver.

The **actuators** for an automated taxi include those available to a human driver: control over the engine through the accelerator and control over steering and braking. In addition, it will need output to a display screen or voice synthesizer to talk back to the passengers, and perhaps some way to communicate with other vehicles, politely or otherwise.

The basic **sensors** for the taxi will include one or more video cameras so that it can see, as well as lidar and ultrasound sensors to detect distances to other cars and obstacles. To avoid speeding tickets, the taxi should have a speedometer, and to control the vehicle properly, especially on curves, it should have an accelerometer. To determine the mechanical state of the vehicle, it will need the usual array of engine, fuel, and electrical system sensors. Like many human drivers, it might want to access GPS signals so that it doesn't get lost. Finally, it will need touchscreen or voice input for the passenger to request a destination.

In Figure 2.5, we have sketched the basic PEAS elements for a number of additional agent types. Further examples appear in Exercise 2.PEAS. The examples include physical as well as virtual environments. Note that virtual task environments can be just as complex as the "real" world: for example, a **software agent** (or software robot or **softbot**) that trades on auction and reselling Web sites deals with millions of other users and billions of objects, many with real images.

Software agent

Softbot

2.3.2 Properties of task environments

The range of task environments that might arise in AI is obviously vast. We can, however, identify a fairly small number of dimensions along which task environments can be categorized. These dimensions determine, to a large extent, the appropriate agent design and the applicability of each of the principal families of techniques for agent implementation. First we list the dimensions, then we analyze several task environments to illustrate the ideas. The definitions here are informal; later chapters provide more precise statements and examples of each kind of environment.

Fully observable vs. **partially observable**: If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are *relevant* to the choice of action; relevance, in turn, depends on the

Fully observable

Partially observable

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments	Touchscreen/voice entry of symptoms and findings
Satellite image analysis system	Correct categorization of objects, terrain	Orbiting satellite, downlink, weather	Display of scene categorization	High-resolution digital camera
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, tactile and joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, raw materials, operators	Valves, pumps, heaters, stirrers, displays	Temperature, pressure, flow, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, feedback, speech	Keyboard entry, voice

Figure 2.5 Examples of agent types and their PEAS descriptions.

performance measure. Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world. An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data—for example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares, and an automated taxi cannot see what other drivers are thinking. If the agent has no sensors at all then the environment is **unobservable**. One might think that in such cases the agent's plight is hopeless, but, as we discuss in Chapter 4, the agent's goals may still be achievable, sometimes with certainty.

Unobservable

Single-agent

Multiagent

Single-agent vs. **multiagent**: The distinction between single-agent and multiagent environments may seem simple enough. For example, an agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two-agent environment. However, there are some subtle issues. First, we have described how an entity *may* be viewed as an agent, but we have not explained which entities *must* be viewed as agents. Does an agent A (the taxi driver for example) have to treat an object B (another vehicle) as an agent, or can it be treated merely as an object behaving according to the laws of physics, analogous to waves at the beach or leaves blowing in the wind? The key distinction is whether B's behavior is best described as maximizing a performance measure whose value depends on agent A's behavior.

For example, in chess, the opponent entity B is trying to maximize its performance measure, which, by the rules of chess, minimizes agent A's performance measure. Thus, chess is a **competitive** multiagent environment. On the other hand, in the taxi-driving environment, avoiding collisions maximizes the performance measure of all agents, so it is a partially **cooperative** multiagent environment. It is also partially competitive because, for example, only one car can occupy a parking space.

Competitive

Cooperative

The agent-design problems in multiagent environments are often quite different from those in single-agent environments; for example, communication often emerges as a rational behavior in multiagent environments; in some competitive environments, randomized behavior is rational because it avoids the pitfalls of predictability.

Deterministic vs. **nondeterministic**. If the next state of the environment is completely determined by the current state and the action executed by the agent(s), then we say the environment is deterministic; otherwise, it is nondeterministic. In principle, an agent need not worry about uncertainty in a fully observable, deterministic environment. If the environment is partially observable, however, then it could *appear* to be nondeterministic.

Deterministic

Nondeterministic

Most real situations are so complex that it is impossible to keep track of all the unobserved aspects; for practical purposes, they must be treated as nondeterministic. Taxi driving is clearly nondeterministic in this sense, because one can never predict the behavior of traffic exactly; moreover, one's tires may blow out unexpectedly and one's engine may seize up without warning. The vacuum world as we described it is deterministic, but variations can include nondeterministic elements such as randomly appearing dirt and an unreliable suction mechanism (Exercise 2.VFIN).

One final note: the word **stochastic** is used by some as a synonym for "nondeterministic," but we make a distinction between the two terms; we say that a model of the environment is stochastic if it explicitly deals with probabilities (e.g., "there's a 25% chance of rain tomorrow") and "nondeterministic" if the possibilities are listed without being quantified (e.g., "there's a chance of rain tomorrow").

Stochastic

Episodic vs. **sequential**: In an episodic task environment, the agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. Many classification tasks are episodic. For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions; moreover, the current decision doesn't affect whether the next part is defective. In sequential environments, on the other hand, the current decision could affect all future decisions.[4] Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

Episodic

Sequential

Static vs. **dynamic**: If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. Dynamic environments, on the other hand, are continuously asking the agent what it wants to do; if it hasn't decided yet,

Static

Dynamic

[4] The word "sequential" is also used in computer science as the antonym of "parallel." The two meanings are largely unrelated.

that counts as deciding to do nothing. If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is **semidynamic**. Taxi driving is clearly dynamic: the other cars and the taxi itself keep moving while the driving algorithm dithers about what to do next. Chess, when played with a clock, is semidynamic. Crossword puzzles are static.

Semidynamic

Discrete

Continuous

Discrete vs. **continuous**: The discrete/continuous distinction applies to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent. For example, the chess environment has a finite number of distinct states (excluding the clock). Chess also has a discrete set of percepts and actions. Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time. Taxi-driving actions are also continuous (steering angles, etc.). Input from digital cameras is discrete, strictly speaking, but is typically treated as representing continuously varying intensities and locations.

Known

Unknown

Known vs. **unknown**: Strictly speaking, this distinction refers not to the environment itself but to the agent's (or designer's) state of knowledge about the "laws of physics" of the environment. In a known environment, the outcomes (or outcome probabilities if the environment is nondeterministic) for all actions are given. Obviously, if the environment is unknown, the agent will have to learn how it works in order to make good decisions.

The distinction between known and unknown environments is not the same as the one between fully and partially observable environments. It is quite possible for a *known* environment to be *partially* observable—for example, in solitaire card games, I know the rules but am still unable to see the cards that have not yet been turned over. Conversely, an *unknown* environment can be *fully* observable—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them.

As noted on page 57, the performance measure itself may be unknown, either because the designer is not sure how to write it down correctly or because the ultimate user—whose preferences matter—is not known. For example, a taxi driver usually won't know whether a new passenger prefers a leisurely or speedy journey, a cautious or aggressive driving style. A virtual personal assistant starts out knowing nothing about the personal preferences of its new owner. In such cases, the agent may learn more about the performance measure based on further interactions with the designer or user. This, in turn, suggests that the task environment is necessarily viewed as a multiagent environment.

The hardest case is *partially observable*, *multiagent*, *nondeterministic*, *sequential*, *dynamic*, *continuous*, and *unknown*. Taxi driving is hard in all these senses, except that the driver's environment is mostly known. Driving a rented car in a new country with unfamiliar geography, different traffic laws, and nervous passengers is a lot more exciting.

Figure 2.6 lists the properties of a number of familiar environments. Note that the properties are not always cut and dried. For example, we have listed the medical-diagnosis task as single-agent because the disease process in a patient is not profitably modeled as an agent; but a medical-diagnosis system might also have to deal with recalcitrant patients and skeptical staff, so the environment could have a multiagent aspect. Furthermore, medical diagnosis is episodic if one conceives of the task as selecting a diagnosis given a list of symptoms; the problem is sequential if the task can include proposing a series of tests, evaluating progress over the course of treatment, handling multiple patients, and so on.

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Figure 2.6 Examples of task environments and their characteristics.

We have not included a "known/unknown" column because, as explained earlier, this is not strictly a property of the environment. For some environments, such as chess and poker, it is quite easy to supply the agent with full knowledge of the rules, but it is nonetheless interesting to consider how an agent might learn to play these games without such knowledge.

The code repository associated with this book (`aima.cs.berkeley.edu`) includes multiple environment implementations, together with a general-purpose environment simulator for evaluating an agent's performance. Experiments are often carried out not for a single environment but for many environments drawn from an **environment class**. For example, to evaluate a taxi driver in simulated traffic, we would want to run many simulations with different traffic, lighting, and weather conditions. We are then interested in the agent's average performance over the environment class.

Environment class

2.4 The Structure of Agents

So far we have talked about agents by describing *behavior*—the action that is performed after any given sequence of percepts. Now we must bite the bullet and talk about how the insides work. The job of AI is to design an **agent program** that implements the agent function—the mapping from percepts to actions. We assume this program will run on some sort of computing device with physical sensors and actuators—we call this the **agent architecture**:

Agent program

Agent architecture

$$agent = architecture + program\,.$$

Obviously, the program we choose has to be one that is appropriate for the architecture. If the program is going to recommend actions like *Walk*, the architecture had better have legs. The architecture might be just an ordinary PC, or it might be a robotic car with several onboard computers, cameras, and other sensors. In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the actuators as they are generated. Most of this book is about designing agent programs, although Chapters 26 and 27 deal directly with the sensors and actuators.

function TABLE-DRIVEN-AGENT(*percept*) **returns** an action
  persistent: *percepts*, a sequence, initially empty
    table, a table of actions, indexed by percept sequences, initially fully specified

  append *percept* to the end of *percepts*
  action ← LOOKUP(*percepts*, *table*)
  return *action*

Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

2.4.1 Agent programs

The agent programs that we design in this book all have the same skeleton: they take the current percept as input from the sensors and return an action to the actuators.[5] Notice the difference between the agent program, which takes the current percept as input, and the agent function, which may depend on the entire percept history. The agent program has no choice but to take just the current percept as input because nothing more is available from the environment; if the agent's actions need to depend on the entire percept sequence, the agent will have to remember the percepts.

We describe the agent programs in the simple pseudocode language that is defined in Appendix B. (The online code repository contains implementations in real programming languages.) For example, Figure 2.7 shows a rather trivial agent program that keeps track of the percept sequence and then uses it to index into a table of actions to decide what to do. The table—an example of which is given for the vacuum world in Figure 2.3—represents explicitly the agent function that the agent program embodies. To build a rational agent in this way, we as designers must construct a table that contains the appropriate action for every possible percept sequence.

It is instructive to consider why the table-driven approach to agent construction is doomed to failure. Let $\mathcal{P}$ be the set of possible percepts and let T be the lifetime of the agent (the total number of percepts it will receive). The lookup table will contain $\sum_{t=1}^{T} |\mathcal{P}|^t$ entries. Consider the automated taxi: the visual input from a single camera (eight cameras is typical) comes in at the rate of roughly 70 megabytes per second (30 frames per second, 1080×720 pixels with 24 bits of color information). This gives a lookup table with over $10^{600,000,000,000}$ entries for an hour's driving. Even the lookup table for chess—a tiny, well-behaved fragment of the real world—has (it turns out) at least 10^{150} entries. In comparison, the number of atoms in the observable universe is less than 10^{80}. The daunting size of these tables means that (a) no physical agent in this universe will have the space to store the table; (b) the designer would not have time to create the table; and (c) no agent could ever learn all the right table entries from its experience.

Despite all this, TABLE-DRIVEN-AGENT *does* do what we want, assuming the table is filled in correctly: it implements the desired agent function.

[5] There are other choices for the agent program skeleton; for example, we could have the agent programs be **coroutines** that run asynchronously with the environment. Each such coroutine has an input and output port and consists of a loop that reads the input port for percepts and writes actions to the output port.

function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

if *status* = *Dirty* **then return** *Suck*
else if *location* = *A* **then return** *Right*
else if *location* = *B* **then return** *Left*

Figure 2.8 The agent program for a simple reflex agent in the two-location vacuum environment. This program implements the agent function tabulated in Figure 2.3.

The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behavior from a smallish program rather than from a vast table.

We have many examples showing that this can be done successfully in other areas: for example, the huge tables of square roots used by engineers and schoolchildren prior to the 1970s have now been replaced by a five-line program for Newton's method running on electronic calculators. The question is, can AI do for general intelligent behavior what Newton did for square roots? We believe the answer is yes.

In the remainder of this section, we outline four basic kinds of agent programs that embody the principles underlying almost all intelligent systems:

- Simple reflex agents;
- Model-based reflex agents;
- Goal-based agents; and
- Utility-based agents.

Each kind of agent program combines particular components in particular ways to generate actions. Section 2.4.6 explains in general terms how to convert all these agents into *learning agents* that can improve the performance of their components so as to generate better actions. Finally, Section 2.4.7 describes the variety of ways in which the components themselves can be represented within the agent. This variety provides a major organizing principle for the field and for the book itself.

2.4.2 Simple reflex agents

The simplest kind of agent is the **simple reflex agent**. These agents select actions on the basis of the *current* percept, ignoring the rest of the percept history. For example, the vacuum agent whose agent function is tabulated in Figure 2.3 is a simple reflex agent, because its decision is based only on the current location and on whether that location contains dirt. An agent program for this agent is shown in Figure 2.8. Simple reflex agent

Notice that the vacuum agent program is very small indeed compared to the corresponding table. The most obvious reduction comes from ignoring the percept history, which cuts down the number of relevant percept sequences from 4^T to just 4. A further, small reduction comes from the fact that when the current square is dirty, the action does not depend on the location. Although we have written the agent program using if-then-else statements, it is simple enough that it can also be implemented as a Boolean circuit.

Simple reflex behaviors occur even in more complex environments. Imagine yourself as the driver of the automated taxi. If the car in front brakes and its brake lights come on, then you should notice this and initiate braking. In other words, some processing is done on the

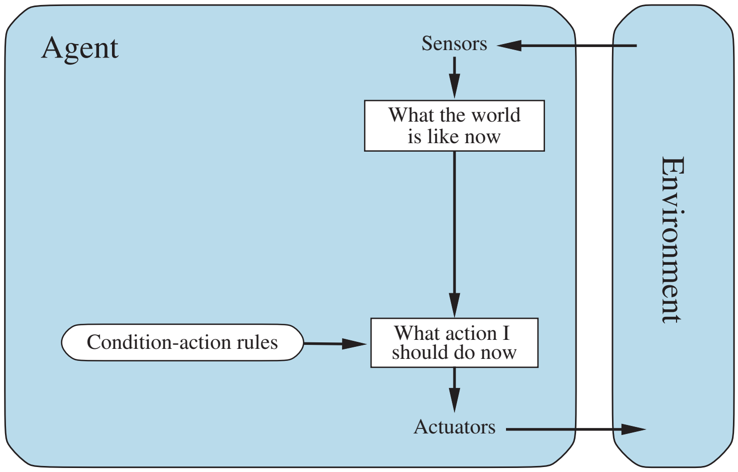

Figure 2.9 Schematic diagram of a simple reflex agent. We use rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process.

visual input to establish the condition we call "The car in front is braking." Then, this triggers some established connection in the agent program to the action "initiate braking." We call such a connection a **condition–action rule**,[6] written as

Condition–action rule

if *car-in-front-is-braking* **then** *initiate-braking*.

Humans also have many such connections, some of which are learned responses (as for driving) and some of which are innate reflexes (such as blinking when something approaches the eye). In the course of the book, we show several different ways in which such connections can be learned and implemented.

The program in Figure 2.8 is specific to one particular vacuum environment. A more general and flexible approach is first to build a general-purpose interpreter for condition–action rules and then to create rule sets for specific task environments. Figure 2.9 gives the structure of this general program in schematic form, showing how the condition–action rules allow the agent to make the connection from percept to action. Do not worry if this seems trivial; it gets more interesting shortly.

An agent program for Figure 2.9 is shown in Figure 2.10. The INTERPRET-INPUT function generates an abstracted description of the current state from the percept, and the RULE-MATCH function returns the first rule in the set of rules that matches the given state description. Note that the description in terms of "rules" and "matching" is purely conceptual; as noted above, actual implementations can be as simple as a collection of logic gates implementing a Boolean circuit. Alternatively, a "neural" circuit can be used, where the logic gates are replaced by the nonlinear units of artificial neural networks (see Chapter 22).

Simple reflex agents have the admirable property of being simple, but they are of limited intelligence. The agent in Figure 2.10 will work *only if the correct decision can be made on the basis of just the current percept—that is, only if the environment is fully observable.*

[6] Also called **situation–action rules**, **productions**, or **if–then rules**.

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

Even a little bit of unobservability can cause serious trouble. For example, the braking rule given earlier assumes that the condition *car-in-front-is-braking* can be determined from the current percept—a single frame of video. This works if the car in front has a centrally mounted (and hence uniquely identifiable) brake light. Unfortunately, older models have different configurations of taillights, brake lights, and turn-signal lights, and it is not always possible to tell from a single image whether the car is braking or simply has its taillights on. A simple reflex agent driving behind such a car would either brake continuously and unnecessarily, or, worse, never brake at all.

We can see a similar problem arising in the vacuum world. Suppose that a simple reflex vacuum agent is deprived of its location sensor and has only a dirt sensor. Such an agent has just two possible percepts: [*Dirty*] and [*Clean*]. It can *Suck* in response to [*Dirty*]; what should it do in response to [*Clean*]? Moving *Left* fails (forever) if it happens to start in square *A*, and moving *Right* fails (forever) if it happens to start in square *B*. Infinite loops are often unavoidable for simple reflex agents operating in partially observable environments.

Escape from infinite loops is possible if the agent can **randomize** its actions. For example, if the vacuum agent perceives [*Clean*], it might flip a coin to choose between *Right* and *Left*. It is easy to show that the agent will reach the other square in an average of two steps. Then, if that square is dirty, the agent will clean it and the task will be complete. Hence, a randomized simple reflex agent might outperform a deterministic simple reflex agent. Randomization

We mentioned in Section 2.3 that randomized behavior of the right kind can be rational in some multiagent environments. In single-agent environments, randomization is usually *not* rational. It is a useful trick that helps a simple reflex agent in some situations, but in most cases we can do much better with more sophisticated deterministic agents.

2.4.3 Model-based reflex agents

The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can't see now*. That is, the agent should maintain some sort of **internal state** that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state. For the braking problem, the internal state is not too extensive—just the previous frame from the camera, allowing the agent to detect when two red lights at the edge of the vehicle go on or off simultaneously. For other driving tasks such as changing lanes, the agent needs to keep track of where the other cars are if it can't see them all at once. And for any driving to be possible at all, the agent needs to keep track of where its keys are. Internal state

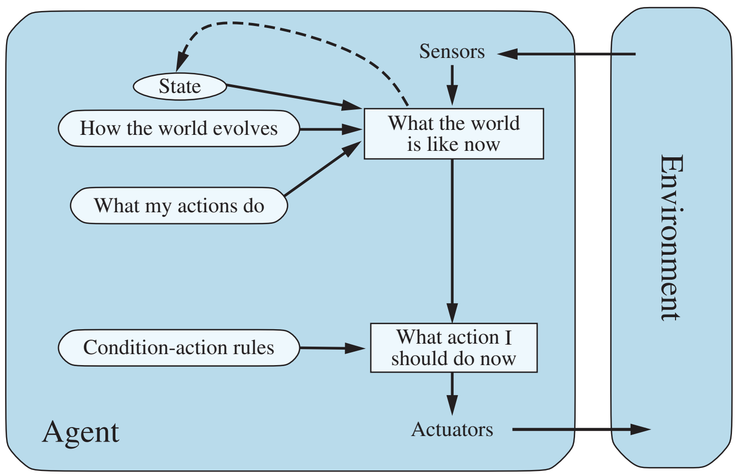

Figure 2.11 A model-based reflex agent.

Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program in some form. First, we need some information about how the world changes over time, which can be divided roughly into two parts: the effects of the agent's actions and how the world evolves independently of the agent. For example, when the agent turns the steering wheel clockwise, the car turns to the right, and when it's raining the car's cameras can get wet. This knowledge about "how the world works"—whether implemented in simple Boolean circuits or in complete scientific theories—is called a **transition model** of the world.

Transition model

Second, we need some information about how the state of the world is reflected in the agent's percepts. For example, when the car in front initiates braking, one or more illuminated red regions appear in the forward-facing camera image, and, when the camera gets wet, droplet-shaped objects appear in the image partially obscuring the road. This kind of knowledge is called a **sensor model**.

Sensor model

Together, the transition model and sensor model allow an agent to keep track of the state of the world—to the extent possible given the limitations of the agent's sensors. An agent that uses such models is called a **model-based agent**.

Model-based agent

Figure 2.11 gives the structure of the model-based reflex agent with internal state, showing how the current percept is combined with the old internal state to generate the updated description of the current state, based on the agent's model of how the world works. The agent program is shown in Figure 2.12. The interesting part is the function UPDATE-STATE, which is responsible for creating the new internal state description. The details of how models and states are represented vary widely depending on the type of environment and the particular technology used in the agent design.

Regardless of the kind of representation used, it is seldom possible for the agent to determine the current state of a partially observable environment *exactly*. Instead, the box labeled "what the world is like now" (Figure 2.11) represents the agent's "best guess" (or sometimes best guesses, if the agent entertains multiple possibilities). For example, an automated taxi

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              transition_model, a description of how the next state depends on
                   the current state and action
              sensor_model, a description of how the current world state is reflected
                   in the agent's percepts
              rules, a set of condition–action rules
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, transition_model, sensor_model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

may not be able to see around the large truck that has stopped in front of it and can only guess about what may be causing the hold-up. Thus, uncertainty about the current state may be unavoidable, but the agent still has to make a decision.

2.4.4 Goal-based agents

Knowing something about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable—for example, being at a particular destination. The agent program can combine this with the model (the same information as was used in the model-based reflex agent) to choose actions that achieve the goal. Figure 2.13 shows the goal-based agent's structure.

Goal

Sometimes goal-based action selection is straightforward—for example, when goal satisfaction results immediately from a single action. Sometimes it will be more tricky—for example, when the agent has to consider long sequences of twists and turns in order to find a way to achieve the goal. **Search** (Chapters 3, 4, and 6) and **planning** (Chapter 11) are the subfields of AI devoted to finding action sequences that achieve the agent's goals.

Notice that decision making of this kind is fundamentally different from the condition–action rules described earlier, in that it involves consideration of the future—both "What will happen if I do such-and-such?" and "Will that make me happy?" In the reflex agent designs, this information is not explicitly represented, because the built-in rules map directly from percepts to actions. The reflex agent brakes when it sees brake lights, period. It has no idea why. A goal-based agent brakes when it sees brake lights because that's the only action that it predicts will achieve its goal of not hitting other cars.

Although the goal-based agent appears less efficient, it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified. For example, a goal-based agent's behavior can easily be changed to go to a different destination,

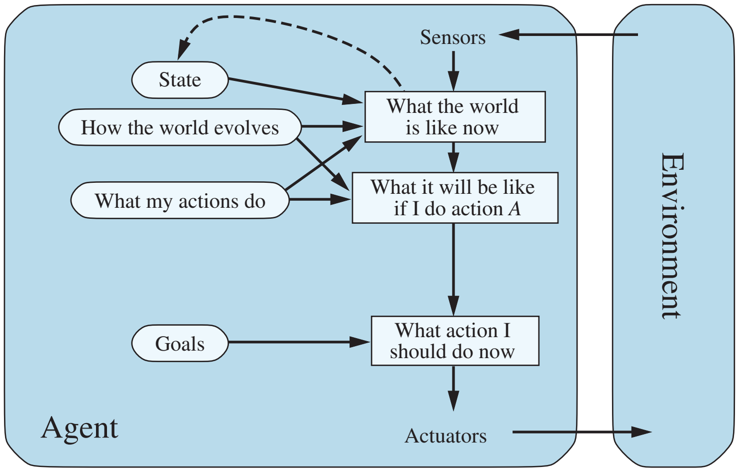

Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

simply by specifying that destination as the goal. The reflex agent's rules for when to turn and when to go straight will work only for a single destination; they must all be replaced to go somewhere new.

2.4.5 Utility-based agents

Goals alone are not enough to generate high-quality behavior in most environments. For example, many action sequences will get the taxi to its destination (thereby achieving the goal), but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude binary distinction between "happy" and "unhappy" states. A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent. Because "happy" does not sound very scientific, economists and computer scientists use the term **utility** instead.[7]

Utility

We have already seen that a performance measure assigns a score to any given sequence of environment states, so it can easily distinguish between more and less desirable ways of getting to the taxi's destination. An agent's **utility function** is essentially an internalization of the performance measure. Provided that the internal utility function and the external performance measure are in agreement, an agent that chooses actions to maximize its utility will be rational according to the external performance measure.

Utility function

Let us emphasize again that this is not the *only* way to be rational—we have already seen a rational agent program for the vacuum world (Figure 2.8) that has no idea what its utility function is—but, like goal-based agents, a utility-based agent has many advantages in terms of flexibility and learning. Furthermore, in two kinds of cases, goals are inadequate but a utility-based agent can still make rational decisions. First, when there are conflicting goals, only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate tradeoff. Second, when there are several goals that the agent can

[7] The word "utility" here refers to "the quality of being useful," not to the electric company or waterworks.

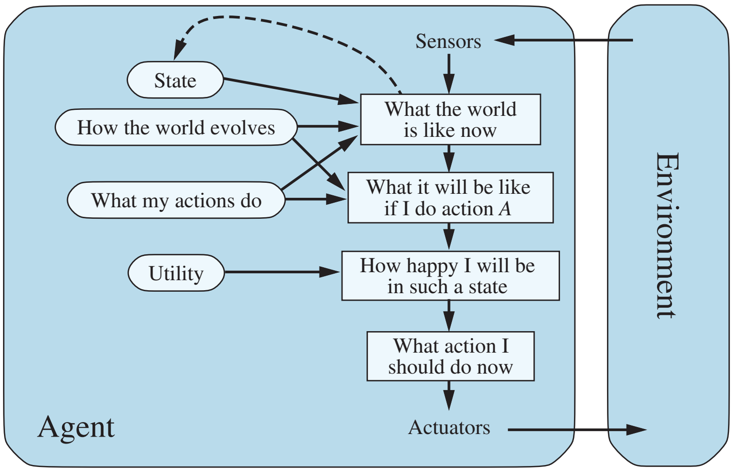

Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed against the importance of the goals.

Partial observability and nondeterminism are ubiquitous in the real world, and so, therefore, is decision making under uncertainty. Technically speaking, a rational utility-based agent chooses the action that maximizes the **expected utility** of the action outcomes—that is, the utility the agent expects to derive, on average, given the probabilities and utilities of each outcome. (Appendix A defines expectation more precisely.) In Chapter 15, we show that any rational agent must behave *as if* it possesses a utility function whose expected value it tries to maximize. An agent that possesses an *explicit* utility function can make rational decisions with a general-purpose algorithm that does not depend on the specific utility function being maximized. In this way, the "global" definition of rationality—designating as rational those agent functions that have the highest performance—is turned into a "local" constraint on rational-agent designs that can be expressed in a simple program.

Expected utility

The utility-based agent structure appears in Figure 2.14. Utility-based agent programs appear in Chapters 15 and 16, where we design decision-making agents that must handle the uncertainty inherent in nondeterministic or partially observable environments. Decision making in multiagent environments is also studied in the framework of utility theory, as explained in Chapter 17.

At this point, the reader may be wondering, "Is it that simple? We just build agents that maximize expected utility, and we're done?" It's true that such agents would be intelligent, but it's not simple. A utility-based agent has to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning. The results of this research fill many of the chapters of this book. Choosing the utility-maximizing course of action is also a difficult task, requiring ingenious algorithms that fill several more chapters. Even with these algorithms, perfect rationality is usually

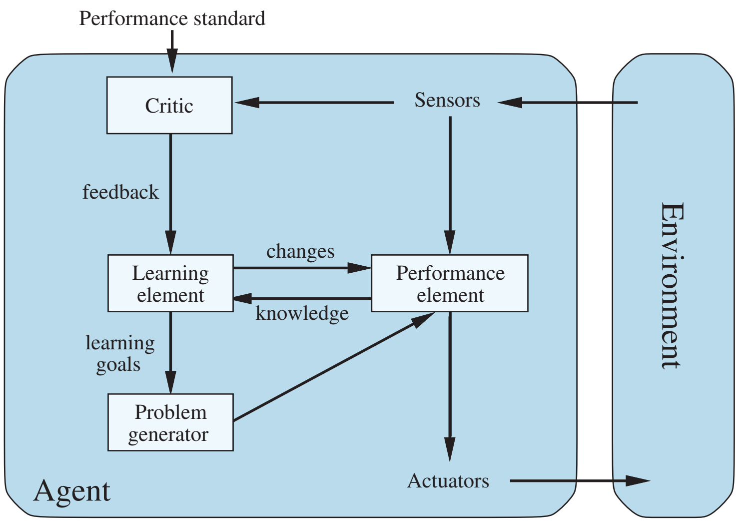

Figure 2.15 A general learning agent. The "performance element" box represents what we have previously considered to be the whole agent program. Now, the "learning element" box gets to modify that program to improve its performance.

unachievable in practice because of computational complexity, as we noted in Chapter 1. We also note that not all utility-based agents are model-based; we will see in Chapters 23 and 26 that a **model-free agent** can learn what action is best in a particular situation without ever learning exactly how that action changes the environment.

Model-free agent

Finally, all of this assumes that the designer can specify the utility function correctly; Chapters 16, 17, and 23 consider the issue of unknown utility functions in more depth.

2.4.6 Learning agents

We have described agent programs with various methods for selecting actions. We have not, so far, explained how the agent programs *come into being*. In his famous early paper, Turing (1950) considers the idea of actually programming his intelligent machines by hand. He estimates how much work this might take and concludes, "Some more expeditious method seems desirable." The method he proposes is to build learning machines and then to teach them. In many areas of AI, this is now the preferred method for creating state-of-the-art systems. Any type of agent (model-based, goal-based, utility-based, etc.) can be built as a learning agent (or not).

Learning has another advantage, as we noted earlier: it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow. In this section, we briefly introduce the main ideas of learning agents. Throughout the book, we comment on opportunities and methods for learning in particular kinds of agents. Chapters 19, 21, 22, and 23 go into much more depth on the learning algorithms themselves.

Learning element

Performance element

A learning agent can be divided into four conceptual components, as shown in Figure 2.15. The most important distinction is between the **learning element**, which is responsible for making improvements, and the **performance element**, which is responsible for selecting external actions. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions. The learning element uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modified to do better in the future. Critic

The design of the learning element depends very much on the design of the performance element. When trying to design an agent that learns a certain capability, the first question is not "How am I going to get it to learn this?" but "What kind of performance element will my agent use to do this once it has learned how?" Given a design for the performance element, learning mechanisms can be constructed to improve every part of the agent.

The critic tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent's success. For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so. It is important that the performance standard be fixed. Conceptually, one should think of it as being outside the agent altogether because the agent must not modify it to fit its own behavior.

The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and informative experiences. If the performance element had its way, it would keep doing the actions that are best, given what it knows, but if the agent is willing to explore a little and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run. The problem generator's job is to suggest these exploratory actions. This is what scientists do when they carry out experiments. Galileo did not think that dropping rocks from the top of a tower in Pisa was valuable in itself. He was not trying to break the rocks or to modify the brains of unfortunate pedestrians. His aim was to modify his own brain by identifying a better theory of the motion of objects. Problem generator

The learning element can make changes to any of the "knowledge" components shown in the agent diagrams (Figures 2.9, 2.11, 2.13, and 2.14). The simplest cases involve learning directly from the percept sequence. Observation of pairs of successive states of the environment can allow the agent to learn "What my actions do" and "How the world evolves" in response to its actions. For example, if the automated taxi exerts a certain braking pressure when driving on a wet road, then it will soon find out how much deceleration is actually achieved, and whether it skids off the road. The problem generator might identify certain parts of the model that are in need of improvement and suggest experiments, such as trying out the brakes on different road surfaces under different conditions.

Improving the model components of a model-based agent so that they conform better with reality is almost always a good idea, regardless of the external performance standard. (In some cases, it is better from a computational point of view to have a simple but slightly inaccurate model rather than a perfect but fiendishly complex model.) Information from the external standard is needed when trying to learn a reflex component or a utility function.

For example, suppose the taxi-driving agent receives no tips from passengers who have been thoroughly shaken up during the trip. The external performance standard must inform the agent that the loss of tips is a negative contribution to its overall performance; then the agent might be able to learn that violent maneuvers do not contribute to its own utility. In a sense, the performance standard distinguishes part of the incoming percept as a **reward** (or **penalty**) that provides direct feedback on the quality of the agent's behavior. Hard-wired performance standards such as pain and hunger in animals can be understood in this way. Reward Penalty

More generally, *human choices* can provide information about human preferences. For example, suppose the taxi does not know that people generally don't like loud noises, and settles on the idea of blowing its horn continuously as a way of ensuring that pedestrians know it's coming. The consequent human behavior—covering ears, using bad language, and possibly cutting the wires to the horn—would provide evidence to the agent with which to update its utility function. This issue is discussed further in Chapter 23.

In summary, agents have a variety of components, and those components can be represented in many ways within the agent program, so there appears to be great variety among learning methods. There is, however, a single unifying theme. Learning in intelligent agents can be summarized as a process of modification of each component of the agent to bring the components into closer agreement with the available feedback information, thereby improving the overall performance of the agent.

2.4.7 How the components of agent programs work

We have described agent programs (in very high-level terms) as consisting of various components, whose function it is to answer questions such as: "What is the world like now?" "What action should I do now?" "What do my actions do?" The next question for a student of AI is, "How on Earth do these components work?" It takes about a thousand pages to begin to answer that question properly, but here we want to draw the reader's attention to some basic distinctions among the various ways that the components can represent the environment that the agent inhabits.

Roughly speaking, we can place the representations along an axis of increasing complexity and expressive power—atomic, factored, and structured. To illustrate these ideas, it helps to consider a particular agent component, such as the one that deals with "What my actions do." This component describes the changes that might occur in the environment as the result of taking an action, and Figure 2.16 provides schematic depictions of how those transitions might be represented.

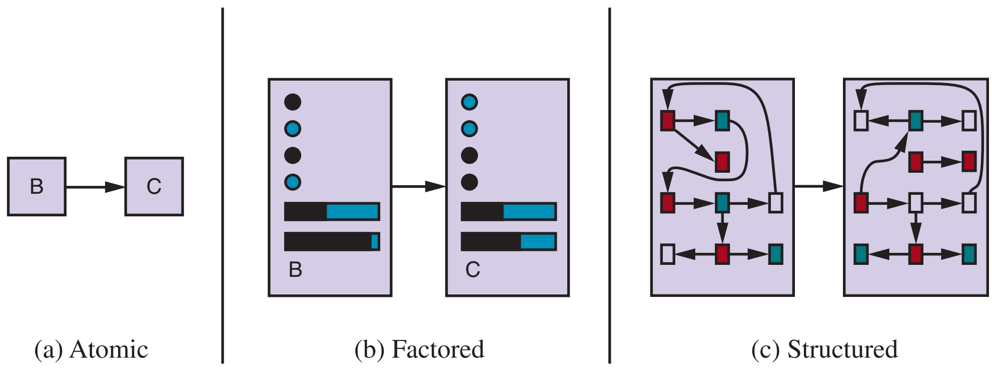

Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

In an **atomic representation** each state of the world is indivisible—it has no internal structure. Consider the task of finding a driving route from one end of a country to the other via some sequence of cities (we address this problem in Figure 3.1 on page 82). For the purposes of solving this problem, it may suffice to reduce the state of the world to just the name of the city we are in—a single atom of knowledge, a "black box" whose only discernible property is that of being identical to or different from another black box. The standard algorithms underlying search and game-playing (Chapters 3, 4, and 6), hidden Markov models (Chapter 14), and Markov decision processes (Chapter 16) all work with atomic representations. Atomic representation

A **factored representation** splits up each state into a fixed set of **variables** or **attributes**, each of which can have a **value**. Consider a higher-fidelity description for the same driving problem, where we need to be concerned with more than just atomic location in one city or another; we might need to pay attention to how much gas is in the tank, our current GPS coordinates, whether or not the oil warning light is working, how much money we have for tolls, what station is on the radio, and so on. While two different atomic states have nothing in common—they are just different black boxes—two different factored states can share some attributes (such as being at some particular GPS location) and not others (such as having lots of gas or having no gas); this makes it much easier to work out how to turn one state into another. Many important areas of AI are based on factored representations, including constraint satisfaction algorithms (Chapter 5), propositional logic (Chapter 7), planning (Chapter 11), Bayesian networks (Chapters 12, 13, 14, 15, and 18), and various machine learning algorithms. Factored representation Variable Attribute Value

For many purposes, we need to understand the world as having *things* in it that are *related* to each other, not just variables with values. For example, we might notice that a large truck ahead of us is reversing into the driveway of a dairy farm, but a loose cow is blocking the truck's path. A factored representation is unlikely to be pre-equipped with the attribute *TruckAheadBackingIntoDairyFarmDrivewayBlockedByLooseCow* with value *true* or *false*. Instead, we would need a **structured representation**, in which objects such as cows and trucks and their various and varying relationships can be described explicitly (see Figure 2.16(c)). Structured representations underlie relational databases and first-order logic (Chapters 8, 9, and 10), first-order probability models (Chapter 18), and much of natural language understanding (Chapters 24 and 25). In fact, much of what humans express in natural language concerns objects and their relationships. Structured representation

As we mentioned earlier, the axis along which atomic, factored, and structured representations lie is the axis of increasing **expressiveness**. Roughly speaking, a more expressive representation can capture, at least as concisely, everything a less expressive one can capture, plus some more. Often, the more expressive language is *much* more concise; for example, the rules of chess can be written in a page or two of a structured-representation language such as first-order logic but require thousands of pages when written in a factored-representation language such as propositional logic and around 10^{38} pages when written in an atomic language such as that of finite-state automata. On the other hand, reasoning and learning become more complex as the expressive power of the representation increases. To gain the benefits of expressive representations while avoiding their drawbacks, intelligent systems for the real world may need to operate at all points along the axis simultaneously. Expressiveness

Another axis for representation involves the mapping of concepts to locations in physical memory, whether in a computer or in a brain. If there is a one-to-one mapping between concepts and memory locations, we call that a **localist representation**. On the other hand, Localist representation if the representation of a concept is spread over many memory locations, and each memory location is employed as part of the representation of multiple different concepts, we call that a **distributed representation**. Distributed representations are more robust against noise and information loss. With a localist representation, the mapping from concept to memory location is arbitrary, and if a transmission error garbles a few bits, we might confuse *Truck* with the unrelated concept *Truce*. But with a distributed representation, you can think of each concept representing a point in multidimensional space, and if you garble a few bits you move to a nearby point in that space, which will have similar meaning.

Distributed representation

Summary

This chapter has been something of a whirlwind tour of AI, which we have conceived of as the science of agent design. The major points to recall are as follows:

- An **agent** is something that perceives and acts in an environment. The **agent function** for an agent specifies the action taken by the agent in response to any percept sequence.
- The **performance measure** evaluates the behavior of the agent in an environment. A **rational agent** acts so as to maximize the expected value of the performance measure, given the percept sequence it has seen so far.
- A **task environment** specification includes the performance measure, the external environment, the actuators, and the sensors. In designing an agent, the first step must always be to specify the task environment as fully as possible.
- Task environments vary along several significant dimensions. They can be fully or partially observable, single-agent or multiagent, deterministic or nondeterministic, episodic or sequential, static or dynamic, discrete or continuous, and known or unknown.
- In cases where the performance measure is unknown or hard to specify correctly, there is a significant risk of the agent optimizing the wrong objective. In such cases the agent design should reflect uncertainty about the true objective.
- The **agent program** implements the agent function. There exists a variety of basic agent program designs reflecting the kind of information made explicit and used in the decision process. The designs vary in efficiency, compactness, and flexibility. The appropriate design of the agent program depends on the nature of the environment.
- **Simple reflex agents** respond directly to percepts, whereas **model-based reflex agents** maintain internal state to track aspects of the world that are not evident in the current percept. **Goal-based agents** act to achieve their goals, and **utility-based agents** try to maximize their own expected "happiness."
- All agents can improve their performance through **learning**.

Bibliographical and Historical Notes

The central role of action in intelligence—the notion of practical reasoning—goes back at least as far as Aristotle's *Nicomachean Ethics*. Practical reasoning was also the subject of McCarthy's influential paper "Programs with Common Sense" (1958). The fields of robotics and control theory are, by their very nature, concerned principally with physical agents. The