1

Turing's Electronic Brains

EVERY STORY NEEDS to start somewhere, and for AI, we have many possible choices, because the dream of AI, in some form or other, is an ancient one.

We could choose to begin in classical Greece, with the story of Hephaestus, blacksmith to the gods, who had the power to bring metal creatures to life.

We could begin in the city of Prague in the 1600s, where, according to legend, the head rabbi created the golem—a magical being fashioned from clay, intended to protect the city's Jewish population from anti-Semitic attacks.

We could begin with James Watt in eighteenth-century Scotland, designing the Governor, an ingenious automatic control system for the steam engines he was building, thereby laying the foundations for modern control theory.

We could begin in the early nineteenth century with the young Mary Shelley, cooped up in a Swiss villa during a spell of bad weather, creating the story of Frankenstein to entertain her husband, the poet Percy Bysshe Shelley, and their family friend, the notorious Lord Byron.

We could begin in London in the 1830s with Ada Lovelace, estranged daughter of the same Lord Byron, striking up a friendship with Charles Babbage, curmudgeonly inventor of mechanical calculating machines, and inspiring the brilliant young Ada to speculate about whether machines might ultimately be creative.

We could equally well begin with the eighteenth-century fascination with automata—cunningly designed machines that gave some illusion of life.

We have many possible choices for the beginning of AI, but for me, the beginning of the AI story coincides with the beginning of the story of computing itself, for which we have a pretty clear starting point: King's College, Cambridge, in 1935, and a brilliant but unconventional young student called Alan Turing.

CAMBRIDGE, 1935

It is hard to imagine now, because he is about as famous as any mathematician could ever hope to be, but until the 1980s, the name of Alan Turing was virtually unknown outside the fields of mathematics and computer science. While students of mathematics and computing might have come across Turing's name in their studies, they would have known little about the full extent of his achievements or his tragic, untimely death. In part, this is because some of Turing's most important work was carried out in secret for the UK government during the Second World War, and the facts of this remarkable work remained classified until the 1970s.[1] But there was surely also prejudice at work here, because Turing was gay at a time when homosexuality was a criminal offense in the United Kingdom. In 1952, he was prosecuted and convicted for what was then called *gross indecency.* His penalty was to take a crude hormone drug that was intended to reduce his sexual desires—a form of "chemical castration." He died, apparently by his own hand, two years later, at the age of just forty-one.[2]

Nowadays, of course, we all know a little of the Turing story, although perhaps not quite as much as we should. The best-known part of the story relates to his code-breaking work at Bletchley Park in the Second World War, made famous in the popular (albeit spectacularly inaccurate) 2014 Hollywood movie *The Imitation Game.* And he certainly deserves enormous credit for that work, which played an important role in the Allied victory. But AI researchers and computer scientists revere him for quite different reasons. He was, for all practical purposes, the inventor of the computer, and shortly after that, he largely invented the field of AI.

Turing was remarkable in many ways, but one of the most remarkable things about him is that he invented computers by accident. As a mathematics student at Cambridge University in the mid-1930s, Turing set himself the precocious challenge of settling one of the leading mathematical problems of the day—a problem that went by the impressive and frankly daunting name of the Entscheidungsproblem. The problem had been posed by the mathematician David Hilbert in 1928. The Entscheidungsproblem asks whether there are mathematical questions that cannot be answered by simply following a recipe. Of course, the questions Hilbert was concerned with were not questions like "Is there a God?" or "What is the meaning of life?" but rather what mathematicians call **decision problems** (*Entscheidungsproblem* is German for "decision problem"). Decision problems are mathematical questions that have a yes/no answer. Here are some examples of decision problems:

- **Is it the case that 2 + 2 = 4?**
- **Is it the case that 4 × 4 = 16?**
- **Is it the case that 7,919 is a prime number?**

As it happens, the answer to all these decision problems is *yes*. I hope you will agree the first two are obvious, but unless you have an unhealthy obsession with prime numbers, you would have had to work a little to answer the last one. So let's think about this final question for a moment.

A prime number, as you will no doubt recall, is a whole number that is only exactly divisible by itself and one. Now, with this in mind, I'm pretty sure you could, at least in principle, find the answer to the final question on your own. There is an obvious method for doing so that is very straightforward, albeit rather tedious for numbers as large as 7,919: check every number that could possibly be a divisor to see whether it divides 7,919 exactly. Follow this procedure carefully, and you will discover that none do: 7,919 is indeed a prime number.[3]

What is important here is that there are precise and unambiguous methods for answering questions like those above. These methods don't require any *intelligence* to apply them—they are nothing more than *recipes,* which can be followed by rote. All we need to do to find the answer is to follow the recipe *precisely.*

Since we have a technique that is guaranteed to answer the question (given sufficient time), we say that questions in the form "Is *n* a prime number?" are **decidable.** I emphasize that all this means is that, whenever we are faced with a question in the form "Is *n* a prime number?" we know that we can definitely find the answer if we are given sufficient time: we follow the relevant recipe, and eventually, we will get the correct answer.

Now, the Entscheidungsproblem asks whether *all* mathematical decision problems like those we saw above are decidable or whether there are some for which there is *no* recipe for finding the answer—no matter how much time you are prepared to put in.

This is a very fundamental question—it asks whether mathematics can be reduced to merely following recipes. And answering this fundamental question was the daunting challenge that Turing set himself in 1935—and which he triumphantly resolved, with dizzying speed.

When we think of deep mathematical problems, we imagine that any solution to them must involve complex equations and long proofs. And sometimes this is indeed the case—when the British mathematician Andrew Wiles famously proved Fermat's last theorem in the early 1990s, it took years for the mathematical community to understand the hundreds of pages of his proof and become confident that it was indeed correct. By these standards, Turing's solution to the Entscheidungsproblem was positively eccentric.

Apart from anything else, Turing's proof is short and comparatively accessible (once the basic framework has been established, the proof is really just a few lines long). But most important, to solve the Entscheidungsproblem, Turing realized that he needed to be able to make the idea of a recipe that can be followed precisely exact. To do this, he invented a mathematical problem-solving machine—nowadays, we call these **Turing machines** in his honor. A Turing machine is a mathematical description of a recipe, like the one for checking prime numbers mentioned above. All a Turing machine does is to follow the recipe it was designed for. I should emphasize that, although Turing called them *machines,* at this point they were nothing more than an abstract mathematical idea. The idea of solving a deep mathematical problem by *inventing a machine* was unconventional, to say the least—I suspect many mathematicians of the day were mystified.

Turing machines are very powerful beasts. Any kind of mathematical recipe that you might care to think of can be encoded as a Turing machine. And if all mathematical decision problems can be solved by following a recipe, then for any decision problem, you should be able to design a Turing machine to solve it. To settle Hilbert's problem, all you had to do was show that there was some decision problem that could not be answered by any Turing machine. And that is what Turing did.

His next trick was to show that his machines could be turned into *general-purpose* problem-solving machines. He designed a Turing machine that will follow *any* recipe that you give it. We now call these general-purpose Turing machines **universal Turing machines:**[4] and a computer, when stripped down to its bare essentials, is simply a universal Turing machine made real. The programs that a computer runs are just recipes, like the one for prime numbers that we discussed above.

Although it isn't central to our story, it is worth at least mentioning how Turing settled the Entscheidungsproblem using his new invention—apart from the fact that it was extraordinarily ingenious, it also has some bearing on the question of whether AI is ultimately possible.

His idea was that Turing machines could be programmed to *answer questions about other Turing machines.* He considered the following decision problem: given a Turing machine and an associated input, will it be guaranteed to eventually halt with an answer, or could it carry on doing its work forever? This is a decision problem of the type that we discussed above—albeit a much more involved one. Now, suppose there was a machine that could solve this problem. Turing saw that the existence of a Turing machine that could answer this question would create a contradiction. It followed that there could be no recipe for checking whether a Turing machine halts, and so the question "Does a Turing machine halt?" is an **undecidable problem.** He had thus established that there are decision problems that cannot be solved by simply following a recipe. He thereby settled Hilbert's Entscheidungsproblem: mathematics could not be reduced to following recipes. In fact, he was almost scooped by a Princeton mathematician by the name of Alonzo Church, but Turing's proof is very different from that of Church: much more direct and much more general. And a by-product of Turing's proof was the invention of the universal Turing machine—a general-purpose problem-solving machine.

Turing didn't invent his machines with the idea of actually building them, but the idea occurred to him soon enough, and many others too. In wartime Munich, Konrad Zuse designed a computer called the Z3 for the Reich Air Ministry—although it was not quite a modern computer, it introduced many of the key ingredients of one. Across the Atlantic in Pennsylvania, a team led by John Mauchly and J. Presper Eckert developed a machine called ENIAC to compute artillery tables. With some tweaks by the brilliant Hungarian mathematician John von Neumann, ENIAC established the fundamental architecture of the modern computer (the architecture of conventional computers is called the Von Neumann architecture, in his honor). Over in postwar England, Fred Williams and Tom Kilburn built the Manchester Baby, which led directly to the world's first commercial computer, the Ferranti Mark 1—Turing himself joined the staff of Manchester University in 1948 and wrote some of the first programs to run on it.

By the 1950s, all the key ingredients of the modern computer had been developed. Machines that realized Turing's mathematical vision were a practical reality—all you needed was enough money to buy one and a building big enough (to house the Ferranti Mark 1 required two storage bays, each sixteen feet long, eight feet high, and four feet wide; the machine consumed twenty-seven kilowatts of electricity—about enough to power three modern homes). Of course, they have been getting smaller and cheaper ever since.

WHAT ELECTRONIC BRAINS ACTUALLY DO

Nothing pleases a newspaper editor more than a catchy headline, and when the first computers were built after the Second World War, newspapers across the world heralded the arrival of a miraculous new invention—*the electronic brain.* These fearsomely complex machines were apparently capable of dazzling feats of mathematics, for example, processing huge volumes of convoluted arithmetic problems much faster and more accurately than any human could ever dream of doing. To those unacquainted with the realities of computers, it must have seemed that machines capable of such tasks must be gifted with some kind of superior intelligence. *Electronic brain* therefore seemed like an obvious label, and it stuck. (One still came across this kind of language as late as the early

1980s, when I first became interested in computers.) In fact, it turns out that what these electronic brains were doing was something incredibly useful—and something that people find tremendously difficult—but not something that requires *intelligence.* And understanding exactly what computers are designed to do—and what they can't do—is central to understanding AI and why it is so difficult.

Remember that Turing machines and their physical manifestation in the form of computers are nothing more than *machines for following instructions.* That is their sole purpose—that is all they are designed to do and all they can do. The instructions that we give a Turing machine are what we nowadays call an *algorithm* or *program.*[5] Most programmers are probably not even aware of the fact that they are interacting with what amounts to a Turing machine, and for good reason—programming Turing machines directly is incredibly fiddly and irksome, as generations of frustrated computer science students would confirm. Instead, we build higher-level languages on top of the Turing machines, to make it simpler to program them—programming languages like Python, Java, and C. All these languages really do is hide some of the gory details of the machine from the programmer in order to make it a bit more accessible. But they are still incredibly fiddly and irksome, which is why programming is hard, why computer programs crash so often, and why good programmers get paid so much.

I have no intention of trying to teach you to program, but it is useful to have some sense of what kinds of instructions computers can follow. Roughly speaking, all a computer can do is follow lists of instructions such as the following:[6]

- **Add *A* to *B***
- **If the result is bigger than *C*, then do *D*; otherwise, do *E***
- **Repeatedly do *F* until *G***

Every computer program boils down to lists of instructions similar to these. Microsoft Word and PowerPoint boil down to instructions like these. *Call of Duty* and *Minecraft* boil down to instructions like these. Facebook, Google, and eBay boil down to instructions like these. The apps on your smartphone, from your web browser to Tinder, all boil down to instructions like these. *And if we are to build intelligent machines, then their intelligence must ultimately reduce to simple, explicit instructions like these.* This, in essence, is the fundamental challenge of AI. The question of whether AI is possible ultimately amounts to whether we can produce intelligent behavior simply by following lists of instructions like these.

In the remainder of this chapter, I want to dig into this observation and try to make explicit some of the implications it has for AI. Before I do this, however, in case it seems like I'm trying to convince you that computers are actually rather useless beasts, I feel obliged to point out several features that make computers rather more useful than they might at this stage appear.

The first point to make is that computers are *fast.* Very, very, very fast. Of course, you know this—but our experiences in the everyday world do not really equip us to understand quite how fast computers actually are. So let us quantify this statement. At the time of writing, a reasonable desktop computer operating at full speed can carry out up to *one hundred billion instructions of the type listed above every second.* One hundred billion is approximately the number of stars in our galaxy, but that probably doesn't help much. So let me put it this way. Suppose you wanted to emulate the computer, by manually carrying out the computer's instructions yourself. Imagine you carry out one instruction every ten seconds and that you don't pause for breaks—you work twenty-four/seven/three-sixty-five to get the job done. Then it would take you about *3,700 years* to do what the computer does in *just one second.*

Of course, as well as being *much* slower as a machine for following recipes, you are different from a computer in one other key respect: there is just no way that you could spend a serious amount of time following instructions in this way without making mistakes. Computers, by contrast, very rarely make mistakes. While programs frequently crash, these crashes are almost invariably the fault of the people who wrote the program rather than a fault in the computer itself. Modern computer processors are phenomenally reliable—they are expected to operate on average for up to fifty thousand hours before failing, faithfully carrying out tens of billions of instructions for every second of those hours.

One final observation. Although computers are just machines for following instructions, this does not mean that they are incapable of making decisions. Computers certainly *can* make decisions—it is just that we have to give them precise instructions about *how* to make the decision. And a computer can subsequently adjust these instructions for itself, as long as we have instructed it how to do so—in this way, as we will see, a computer can change its behavior over time—it can *learn.*

WHY IS AI HARD?

Computers, then, can *reliably* follow *very simple* instructions *very, very quickly,* and they can *make decisions* as long as those decisions are precisely specified. Now, some things that we might want computers to do for us are very easy to encode in this way—but others are not. To understand why AI is hard—why progress has proved so elusive over the years—it helps to look at problems that are easy to encode in this way and problems that are not, and to see why this is the case. Figure 1 displays some tasks that we might want computers to do and shows how difficult it has proved to get computers to achieve them.

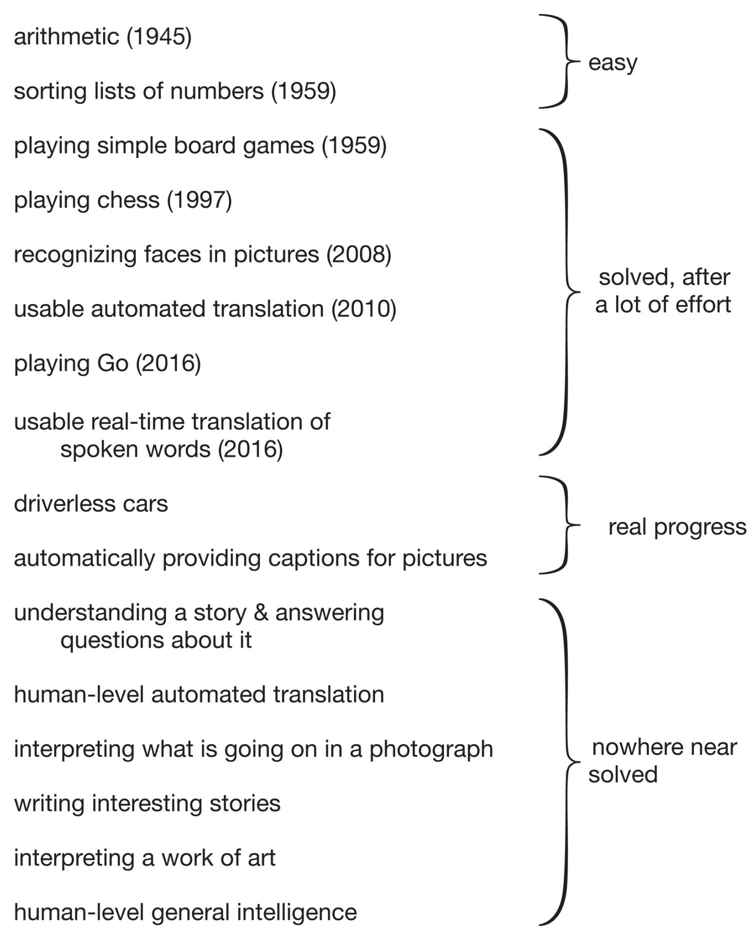

Figure 1: Some tasks that we want computers to be able to do ranked in order of difficulty. Years in parentheses indicate approximately when the problem was solved. At present, we have no idea about how to get computers to do the tasks at the bottom of the list.

At the top is arithmetic. It was very easy to get computers to do arithmetic, because all the basic arithmetical operations (addition, subtraction, multiplication, division) can be carried out by a simple recipe—you were taught such recipes at school, even if you don't recall them now. These recipes can be directly translated into computer programs, and so problems involving the straightforward application of arithmetic were solved in the very earliest days of computing. (The first program Turing wrote for the Manchester Baby computer when he joined the staff of Manchester University was to carry out long division—it must have been an odd experience for Turing, returning to basic arithmetic after solving one of the deepest mathematical problems of the twentieth century.)

Next we have sorting, by which I just mean arranging a list of numbers into, say, ascending order or a list of names into alphabetical order. There are some very simple recipes for sorting—see if you can think of one. However, the most obvious of these are painfully slow and not usable in practice: 1959 saw the invention of a technique called Quicksort, which for the first time provided a really practical way to do this. (In the sixty years since it was invented, nobody has managed to do much better than Quicksort.[7])

Then we move on to problems that require much more effort. Playing board games well turned out to be a major challenge, for a reason that is fundamental to the AI story. It turns out that there is in fact a very simple and elegant recipe for playing board games well; it is based on a technique called **search,** about which we will hear a lot more in the next chapter. The problem is that although the basic recipe for game playing by search is very easy to program, it doesn't work on all but the most trivial of games, because it requires too much time or too much computer memory—even if we could use every atom in the universe to build a computer, that computer would not be remotely powerful enough to be able to play chess or Go using "naive" search. To make search viable, we need something extra—and as we will see in the next chapter, this is where AI comes in.

This situation—we know how to solve a problem in principle, but these techniques don't work in practice because they would require impossibly large amounts of computing resources—is extremely common in AI, and a huge body of research has grown up around dealing with it.

The next problems on the list, though, are very much not of this type. Recognizing faces in a picture, automated translation, and usable real-time translation of spoken words are all very different kinds of problems because, unlike board games, conventional computing techniques give us no clues about how to write a recipe to solve them. We need a completely new approach. In each of these cases, though, the problem has been solved, by a technique called *machine learning,* about which we will hear a lot more later on in this book.

Next, we move on to driverless cars. This problem is fascinating because it is something that seems so straightforward for people to do. We don't associate the ability to drive a car with *intelligence.* But it turned out to be fearsomely hard to get computers to drive cars. The main problem is that a car needs to understand where it is and what is going on around it. Imagine a driverless car at a busy road intersection in New York City. There will probably be many vehicles in continual movement, with pedestrians, cyclists, construction, traffic signs, and road markings in the picture as well. It may be raining or snowing or foggy, just to complicate things further (and in New York, it may well be all three at the same time). In situations like this, the main difficulty is not *deciding what to do* (slow down, speed up, turn left, turn right, etc.) but rather *making sense of what is going on around you*—identifying where you are, what vehicles are present, where they are and what they are doing, where the pedestrians are, and so on. If you have all that information, then deciding what you need to do is usually going to be pretty easy. (We will discuss the specific challenges of driverless cars later in the book.)

Then we move on to problems that we really have little idea how to solve. How can a computer understand a complex story and answer questions about it? How can a computer translate a rich, nuanced text such as a novel? How can a computer interpret what is going on in a picture—not just identifying the people in it, but *actually interpreting what is going on*? How can a computer write an interesting story or interpret a work of art such as a painting? And the final one—the grand dream of computers with general-purpose human-level intelligence—is the greatest of all.

Progress in AI, then, means progressively getting computers to do more and more of these tasks in figure 1. Those that are difficult are so for usually one of two reasons. The first possibility is that while we do in fact have a recipe for the problem that works *in principle,* it doesn't work *in practice* because it would require impossibly large amounts of computing time and memory. Board games like chess and Go fall into this category. The second possibility is that we just have no real idea what a recipe for solving the problem might look like (e.g., recognizing faces), in which latter case we would need something completely new (e.g., machine learning). Pretty much all of contemporary AI is concerned with problems that fall into one of these two categories.

Let's now turn to the hardest problem in figure 1—the grand dream of general-purpose, human-level intelligence. Not surprisingly, this challenge has attracted a lot of attention, and one of the earliest and most influential thinkers on the subject was our old friend Alan Turing.

THE TURING TEST

The development of the first computers in the late 1940s and early 1950s prompted a flurry of public debate about the potential of these wondrous feats of modern science. One of the highest-profile contributions to the debate at the time was a book entitled *Cybernetics,* written by an MIT mathematics professor called Norbert Wiener. The book made explicit parallels between machines and animal brains and nervous systems, and touched on many ideas relating to AI. It attracted huge public interest, despite the fact that it was surely incomprehensible to any but the most dedicated and mathematically adept reader. Questions such as whether machines could "think" began to be seriously debated in the press and on radio shows (in 1951, Turing himself participated in a BBC radio show on this very subject). Although it didn't yet have a name, the idea of AI was in the air.

Prompted by the public debate, Turing began to think seriously about the possibility of artificial intelligence. He was particularly irritated with claims being made in the public debate along the lines of "Machines will never be able to do *x*" (where *x* is, for example, think, reason, or be creative). He wanted to definitively silence those who argued that "machines cannot think." To this end, he proposed a test, which we now call the **Turing test.**

Turing's test has been hugely influential since he first described it in 1950, and it is still the subject of serious research to the present day. Sadly, though, for reasons that will become clear later, it failed to silence the doubters.

Turing's inspiration was a Victorian-era parlor game called the Imitation Game. The basic idea of the Imitation Game was for someone to try to tell if another person was a man or a woman simply on the basis of the answers they gave to questions that were posed to them. Turing proposed that you could use a similar test for AI. The way the test is usually described is something like this:

Human interrogators interact via a computer keyboard and screen with something that is either a person or a computer program—the interrogator does not know in advance whether it is a person or a program. The interaction is purely in the form of textual questions and answers: the interrogator types a question, and a response is displayed. The task of the interrogator is to determine whether the thing being interrogated is a person or a computer program.

Now, suppose that the thing being interrogated is indeed a computer program, but after some reasonable amount of time, the interrogators cannot reliably tell whether they are interacting with a program or a person. Then surely, Turing argued, you should accept that the program has some sort of human-level intelligence (or whatever you want to call it).

Turing's brilliant move here is to sidestep all the questions and debate whether a program was "really" intelligent (or conscious or whatever). Whether or not the program is "really" thinking (or is conscious or self-aware or whatever) is irrelevant because it is doing something that makes it *indistinguishable from the "real thing."* The key word here is *indistinguishable.*

The Turing test is a neat example of a standard technique in science. If you want to find out whether two things are the same or different, ask whether there is some reasonable test you can apply that would distinguish them. If there is a reasonable test that one would pass but the other would not, then you can be confident that they are different. If you can't distinguish them by any such test, then you can't claim they are different. Turing's test is concerned with distinguishing machine intelligence from human intelligence, and the test is whether a person can distinguish the behavior that each produces.

Notice that the only evidence that the interrogators have to go on are the inputs and outputs—the questions sent by the interrogator and the responses that the interrogator later receives. The thing they are interrogating is a *black box* as far as the Turing test is concerned, in the sense that we are not allowed to examine its internal structure: all that we have are the inputs and outputs. This is important, because it means the Turing test cannot take into account *how* the entity being tested actually works—only the behavior it produces.

Turing's article "Computing Machinery and Intelligence," describing his test, was published in the prestigious international journal *Mind* in 1950.[8] Although many articles touching on AI-like ideas had been published before this, Turing approached the subject for the first time from the standpoint of the modern digital computer. As such, his article is generally recognized as the first AI publication.

TURING TEST NONSENSE

The Turing test is simple, elegant, and easy to understand. However, it had the unfortunate side effect of establishing the test as the holy grail of AI—with unfortunate consequences that resonate to the present day.

The problem is that most attempts to tackle the Turing test tend to use cheap tricks to try to befuddle the interrogators into believing that they are dealing with human intelligence rather than by trying to seriously engage with the actual issues of intelligent behavior. The most famous example of this was a program called **ELIZA,** written by the German American MIT computer scientist Joseph Weizenbaum in the mid-1960s. ELIZA was never intended by Weizenbaum to be a contender for the Turing test—it was a serious and influential scientific experiment in its own right—but sadly, it has since become synonymous with superficial approaches to AI generally and the Turing test in particular.

ELIZA takes the role of a psychiatrist talking to a patient, asking open-ended questions to get the interviewee to contemplate themselves. ELIZA worked by picking up on key words like *family, friends, lonely, sad,* and so on and then used canned scripts associated with the relevant word to issue follow-up questions. You don't have to be very perceptive to see beyond its smoke and mirrors. Here is a dialogue with a modern version of the system (ELIZA responses are in italics):

> I am lonely.
> *Are you lonely often?*
> Yes. Almost every day.
> *Why do you say almost every day?*
> I think it is my family.
> *Tell me something about your family.*
> They hate me.
> *What makes you believe they hate you?*
> They are mean to me.
> *Earlier you said you felt lonely?*
> What is the meaning of life?
> *Maybe your life has something to do with this.*
> What is the purpose of existence?
> *Is it because of your plans that you say what is the purpose of existence?*

The opening is promising enough, but even after a couple of sentences, the superficiality of the interaction becomes painfully clear—ELIZA obviously has no understanding of the dialogue (and of course Weizenbaum never claimed that it did—that was not the point of his experiment).

Weizenbaum could hardly have anticipated the impact that ELIZA would have, both within academia and beyond. He was, apparently, startled to discover that users took his simple experiment seriously. It immediately entered the folklore of AI. I remember as a Ph.D. student in the 1980s hearing that lonely and socially ill-adjusted graduate students would converse all night with ELIZA, as a proxy for the human relationships they were incapable of forming; there was even a story—hopefully apocryphal—about ELIZA unwittingly encouraging a depressed student to commit suicide.

ELIZA's legacy lives on to this day, in the form of the annual Loebner Prize competition. It isn't clear whether Turing imagined that anyone would ever actually try out his test for real, but, in 1990, that is precisely what American millionaire inventor Hugh Loebner decided to do. Every year, the Loebner Prize invites the submission of computer programs to engage in the Turing test, attempting to convince a panel of judges that they are in fact people. A successful entry stands to win $100,000.

Loebner's competition was instantly controversial—famous AI scientist Marvin Minsky dismissed it as a "meaningless publicity stunt." The competition seems to generate at least one hopelessly misleading headline every year. The problem is that entries in the Loebner competition are, for the most part, variations on the ELIZA theme. Rather than trying to win by engaging the interrogators in a meaningful conversation that demonstrates human-level comprehension, common sense, and understanding, they use tricks to misdirect and mislead.

ELIZA is the direct ancestor of a phenomenon that makes AI researchers groan whenever it is mentioned: the internet chatbot. These are internet-based programs that attempt to engage users in conversation, often via social media platforms such as Twitter. Most internet chatbots use nothing more than keyword-based canned scripts in the same way that ELIZA did, and as a consequence, the conversations they produce are every bit as superficial and uninteresting. Chatbots of this kind are not AI.

VARIETIES OF ARTIFICIAL INTELLIGENCE

For all the nonsense it has given rise to, the Turing test is an important part of the AI story because, for the first time, it gave researchers interested in this emerging discipline a target to aim at. When someone asked you what your goal was, you could give a straightforward and precise answer: *My goal is to build a machine that can meaningfully pass the Turing test.* Today, I think very few, if any, serious AI researchers would give this answer, but it had a crucial historical role, and I believe it still has something important to tell us today.

Much of the attraction of the Turing test undoubtedly lies in its simplicity, but clear as the test appears to be, it nevertheless raises many problematic questions about AI.

Imagine that you are an interrogator in the Turing test. The answers you receive in the test convince you that you are not interacting with a chatbot: whatever is on the other end is demonstrating the kind of understanding of your questions and producing the kinds of answers that a human might demonstrate. It subsequently transpires that the thing on the other end is in fact a computer program. Now, as far as the Turing test is concerned, the matter is settled. The program is doing something indistinguishable from human behavior: end of debate. But there are still at least two logically distinct possibilities:

1. The program *actually understands* the dialogue, in much the same sense that a person does.
2. The program does not understand, but can *simulate* such understanding.

The first claim—that the program *really does understand*—is much stronger than the second. I suspect most AI researchers—and probably most readers of this book—would accept that programs of type 2 are feasible, at least in principle, but would need a lot more convincing before they accepted that programs of type 1 are. And indeed, it is not obvious how we could substantiate a claim that a program was of type 1—this is not what the Turing test claims to do. (Turing, I suspect, would have been irritated by the distinction: he would have pointed out that one of the main reasons he invented the test was to put an end to arguments about such distinctions.) Aiming to build a program of type 1 is therefore a much more ambitious and controversial goal than building a program of type 2.

The goal of building programs *that really do have understanding (consciousness, etc.) in the way that people do* is called **strong AI;** the weaker goal, of building programs that demonstrate the same capability but without any claim that they *actually possess these attributes,* is called **weak AI.**

BEYOND THE TURING TEST

There are many variations of Turing's indistinguishability test. For example, in a much stronger version of the test, we can imagine robots that attempt to pass themselves off as people in the everyday world. Here, "indistinguishability" is interpreted in a very demanding way—it means that the machines are indistinguishable from people. (I'm assuming that you aren't allowed to dissect them, so the test is still a sort of black box.) For the foreseeable future, this sort of thing is firmly in the realms of fiction. Indeed, a world where robots were hard to distinguish from people was the basis for at least one very good film, Ridley Scott's 1982 classic, *Blade Runner,* in which Rick Deckard, portrayed by a young Harrison Ford, spends his days carrying out cryptic tests with the aim of determining whether what appears to be a beautiful woman is in fact a robot. Similar themes are explored in movies such as *Ex Machina* (2014).

Although *Blade Runner* scenarios are not in prospect, researchers have begun to ask whether there are variations of the Turing test that might meaningfully test for genuine intelligence and are resistant to trickery of the chatbot variety. One very simple idea is to test for *comprehension,* and one manifestation of this idea is the use of what are called **Winograd schemas.** These are short questions, perhaps best illustrated by examples:[9]

> **Statement 1a:** The city councilors refused the demonstrators a permit because they <u>feared</u> violence.
>
> **Statement 1b:** The city councilors refused the demonstrators a permit because they <u>advocated</u> violence.
>
> **Question:** Who [feared/advocated] violence?

Notice that these two statements differ from each other in just one word (underlined in each case), but that small change dramatically alters their meaning. The point of the test is to identify who "they" refers to in each case. In Statement 1a, "they" clearly refers to the councilors (they are the ones that fear violence from the demonstrators), while in Statement 1b, "they" refers to the demonstrators (the councilors are concerned about the fact that the demonstrators are advocating violence).

Here is another example:

> **Statement 2a:** The trophy doesn't fit into the brown suitcase because it is too <u>small</u>.
>
> **Statement 2b:** The trophy doesn't fit into the brown suitcase because it is too <u>large</u>.
>
> **Question:** What is too [small/large]?

Obviously, in Statement 2a, the suitcase is too small; in statement 2b, the trophy is too large.

Now, most literate adults would easily be able to handle these examples and others like them—but they resist the kind of cheap tricks used by chatbots. To be able to give the correct answer, it is hard to avoid the conclusion that you really need to *understand* the text and to have some *knowledge* about the kind of scenario in question. To understand the difference between Statements 1a and 1b, for example, you would need to know something about demonstrations (demonstrations often lead to violence) and councilors (that they have the power to grant or deny permits for demonstrations to take place and that they will try to avoid situations that lead to violence).

Another similar challenge for AI involves understanding of the human world and the unwritten rules that govern our relationships within it. Consider the following short dialogue, due to the psychologist and linguist Steven Pinker:

> Bob: I'm leaving you.
> Alice: Who is she?

Can you explain this dialogue? Of course you can. It's a staple of TV soap operas: Alice and Bob are in a relationship, and Bob's announcement leads Alice to believe that Bob is leaving her for another woman—and she wants to know who the other woman is. We might also speculate that Alice is pretty angry.

But how could a computer be programmed to understand such dialogues? Such a capability would surely be essential for understanding stories and indeed for writing them. This common-sense, everyday ability to understand the workings of human beliefs, desires, and relationships would be an essential requirement for a computer program capable of following *Days of Our Lives.* We all have this capability, and it would also seem to be a key requirement for both strong and weak AI. But the ability for machines to comprehend and answer questions about such scenarios remains a long way off.

GENERAL AI

While the grand dream of AI seems intuitively obvious, as we have demonstrated, it is surprisingly hard to define what it means or when we will know that we have found it. For this reason, although strong AI is an important and fascinating part of the AI story, it is largely irrelevant to contemporary AI research. Go to a contemporary AI conference, and you will hear almost nothing about it—except possibly late at night, in the bar.

A lesser goal is to build machines that have general-purpose human-level intelligence. Nowadays, this is usually referred to as **artificial general intelligence (AGI)** or just **general AI. AGI** roughly equates to having a computer that has the full range of intellectual capabilities that a person has —this would include the ability to converse in natural language (cf. the Turing test), solve problems, reason, perceive its environment, and so on, at or above the same level as a typical person. The literature on AGI usually isn't concerned with issues such as consciousness or self-awareness, so AGI might be thought of as a weaker version of weak AI.[10]

However, even this lesser goal is somewhat on the margins of contemporary AI. Instead, what AI researchers usually focus on is building computer programs that can carry out tasks that currently require brains—progressively moving through the list of problems that we saw in figure 1. This approach to AI—getting computers to do very specific tasks—is sometimes called **narrow AI,** but I have rarely heard this term used within the AI community. We don't refer to what we do as narrow AI—because narrow AI *is* AI. This might be disappointing to those of us that long for robot butlers, though it will perhaps be welcome news to those that have nightmares about the robot uprising.

So now you know what AI is and have some idea about why it's hard to achieve. But how, exactly, do AI researchers go about it?

BRAIN OR MIND?

How might we go about producing human-level intelligent behavior in a computer? Historically, AI has adopted one of two main approaches to this problem. Put crudely, the first possibility involves trying to model *the mind*: the processes of conscious reasoning, problem solving, and so on, which we all make use of as we go about our lives. This approach is called **symbolic AI,** because it makes use of symbols that stand for things that the system is reasoning about. For example, a symbol "room451" within a robot's control system might be the name that the robot uses for your bedroom, and a symbol "cleanRoom" might be used as the name for the activity of cleaning a room. As the robot figures out what to do, it makes explicit use of these symbols, and so, for example, if the robot decides to carry out the action "cleanRoom(room451)," then this means that the robot has decided to clean your bedroom. The symbols the robot is using *mean something* in the robot's environment.

For about thirty years, from the mid-1950s until the late 1980s, symbolic AI was the most widely adopted approach to building AI systems. It has a lot of advantages, but perhaps the most important of these is that it is *transparent:* when the robot concludes that it will "cleanRoom(room451)," then we can immediately understand what it has decided to do. But also, I think it was popular because it seems to reflect our conscious thought processes. We think in terms of symbols—words—and when deciding what to do, we might have a mental conversation with ourselves about the pros and cons of various courses of action. Symbolic AI aspired to capture all this. As we will see in chapter 3, symbolic AI peaked in the early 1980s.

The alternative to modeling the mind is to model the *brain.* One extreme possibility would be to try to simulate a complete human brain (and perhaps a human nervous system) within a computer. After all, human brains are the only things that we can say with certainty are capable of producing human-level intelligent behavior. The problem with this approach is that the brain is an unimaginably complex organ: a human brain contains about one hundred billion interconnected components, and we don't remotely understand the structure and operation of these well enough to duplicate it. It isn't a realistic possibility any time soon, and I suspect it is unlikely ever to be possible (although I'm sorry to say that hasn't stopped some people from trying).[11]

What we can do instead, however, is to take inspiration from some structures that occur in the brain and model these as components in intelligent systems. This research area is called **neural networks,** or **neural nets**—the name comes from the cellular information processing units called *neurons* that we find in the microstructure of the brain. The study of neural nets goes back to before the emergence of AI itself and has evolved alongside the mainstream of AI. It is this area that has shown impressive progress this century, which has led to the current boom in AI.

Symbolic AI and neural nets are very different approaches, with utterly different methodologies. Both have moved in and out of fashion over the past sixty years, and as we will see, there has even been acrimony between the two schools. However, as AI emerged as a new scientific discipline in the 1950s, it was symbolic AI that largely held sway.