Main content

Chapter

15

Artificial Intelligence

Main content

Chapter Introduction

r studying this chapter, you will be able to:

Describe the two types of artificial intelligence
Explain the pros and cons of various knowledge representation methods
Explain the parts of a simple neural network, how it works, and how it can incorporate machine learning
Describe how intelligent state-space search algorithms work

Give examples of possible usage for each of the following: swarm intelligence, intelligent agents, and expert systems
Explain what a robot is, and list some tasks for which robots are currently suited

Explain what a drone is, and list some tasks drones can perform
Change font sizeMain content

15.1 Introduction

Artificial intelligence (AI) is the branch of computer science that explores techniques for incorporating aspects of intelligence into computer systems. This definition, however, raises more questions than it answers. What really is "intelligence"? Is it a uniquely human attribute? If a computer system exhibits behavior that we might characterize as intelligent, does that make it truly intelligent? What sorts of behaviors demonstrate intelligence?

Alan Turing, whose investigations into the fundamental nature of computation led to the Turing machine (Chapter 12), was also interested in artificial intelligence. In 1950, before the term *artificial intelligence* was coined,* he proposed a test to measure the intelligent behavior of machines. The **Turing test** allows a human to interrogate two entities, both hidden from the interrogator (Figure 15.1). One entity is a human and the other a machine (a computer). The interrogator can ask the entities questions and receive their responses. The communication is carried on in some form that does not alone reveal which entity is the computer; for example, the interrogator's questions could be typed on a keyboard and the responses printed out. If, as a result of this questioning, the interrogator is unable to determine which entity is the human and which is the computer, then the computer has exhibited sufficiently human intelligence to pass the Turing test. This test does not explore the nature of human intelligence in a deep philosophical way; it merely says that if a machine exhibits behavior indistinguishable from that of a human, then how can we tell the difference—a question that has been explored more than once in science fiction.

Figure 15.1 The Turning test

We need to distinguish between *general artificial intelligence* and *narrow artificial intelligence*. The aim of general artificial intelligence is to model human intelligence, that is, all aspects of human intelligence, or at least as many as possible. This includes the capability for understanding natural language and nuances of meaning; the ability to accumulate knowledge and adapt that knowledge to apply it to new situations or problems; being able to interpret sensory information and then draw reasonable conclusions from it; a lot of "common sense"; and finally some amount of emotional empathy (remember how Commander Data of *Star Trek* fame always wanted an "emotion chip"?).

Victory in the Turing Test?

Numerous Turing test competitions have been held over the years, but no computer had sufficiently convinced its interrogators of its "humanity" until June 2014. On the 60th anniversary of Alan Turing's death, a Turing test competition was held at the Royal Society in London. A chatbot (chatter robot or chatterbot) pretended to be a 13-year-old boy from Ukraine named Eugene Goostman. Goostman (let's call it by the name the chatbot claimed) held 30 five-minute sessions involving a total of 10 interrogators. And in 33% of these conversations, Goostman won, that is, "he" was proclaimed to be the human. The passing score for the test was 30%. Goostman had scored 29% in a 2012 competition, so evidently "he" had gotten a little smarter since then. More impressive, this was the first Turing test with open-ended questions allowed, rather than one in which questions were set ahead of time or could only pertain to a specific topic. The software powering Goostman was indeed remarkable. It was clever that the storyline had Goostman as an adolescent, presumably speaking English as a second language. This justified Goostman's lack of knowledge in some areas and perhaps some bad grammar. The software developers also incorporated a rather cocky attitude, a tendency to change the subject, and a little wackiness in Goostman's responses. All of this typical teenage behavior made a more convincing case for Goostman's "humanness."

But some AI experts claim that the Turing test really isn't a test of human intelligence, and that a win such as Goostman achieved is more misdirection and fakery than a demonstration of true human intelligence. An alternative challenge has been proposed based on something called the Winograd schema in honor of Stanford professor Terry Winograd, an early AI researcher. A **Winograd schema** is a sentence that involves two entities and a pronoun that refers back to one of the two entities; the challenge is to decide to which entity the pronoun refers. Such challenges are simple for humans based on life experience or common sense, but may be difficult for a computer. For example,

John couldn't see the stage with Billy in front of him because he is so short.* Who is so short, John or Billy?

Obviously (responds the human), the "he" refers to John. Note that if "short" were changed to "tall," the answer would be Billy; this one-word change that switches the answer is also a required feature of a Winograd schema.

General artificial intelligence was the original thrust of artificial intelligence research. And much progress has been made in natural language understanding, machine learning, facial recognition, and other areas. Conversely, attempts to model human intelligence within a computer have in turn made contributions to *cognitive science*, the study of how humans think and learn. But just as we learned in Chapter 13 that a model cannot capture all aspects of the physical phenomenon it represents, so artificial intelligence cannot (yet) capture all aspects of human intelligence. Science fiction conjures scenarios of thinking and all-knowing computers controlling the world, but the advances in general artificial intelligence have been far more modest.

The most visible successes have occurred in the area of narrow artificial intelligence, which focuses on solving very particular kinds of problems. The skills involved here are more specific, often based on large amounts of data. Game playing, expert systems, robotics, and recommendation systems are areas of great success for narrow artificial intelligence systems. When you ask questions of Siri, Alexa, Cortana, or Google Assistant, those systems are accessing a giant database of possibly pertinent facts. No real humanlike conversation is going to develop out of that. For example:*

Main content

Chapter Contents

15.2 A Division of Labor

To understand better what artificial intelligence is all about, let's consider a division of task types. Humans can perform a great variety of tasks, but we'll divide them into three categories, representative but by no means exhaustive:

- *Computational tasks*
  - Adding a column of numbers
  - Sorting a list of numbers into numerical order
  - Searching for a given name in a list of names
  - Managing a payroll
  - Calculating trajectory adjustments for a space shuttle
- *Recognition tasks*
  - Recognizing your best friend
  - Understanding the spoken word
  - Finding the tennis ball in the grass in your back yard
- *Reasoning tasks*
  - Planning what to wear today
  - Deciding on the strategic direction a company should follow for the next five years
  - Running the triage center in a hospital emergency room after an earthquake

Algorithmic solutions exist for computational tasks (we devised algorithms for sorting and searching in the early chapters of this book). As humans, we can, in principle at least, follow these step-by-step instructions. Computational tasks are also tasks for which accurate answers must be found—sometimes very quickly—and that's where we as humans fall down. We make mistakes, we get bored, and we aren't very speedy. Computers are better (faster and more accurate) at performing computational tasks, provided they are given programs that correctly embody the algorithms. Throughout this book, with its emphasis on algorithms, we've been talking a great deal about designing procedures to solve computational tasks, learning how to translate these procedures into a programming language, and designing machines to execute the resulting programs.

Humans are often better at recognition tasks. We should perhaps expand the name of this task type to sensory/recognition/motor-skills tasks because we receive information through our senses (primarily seeing and hearing), we recognize or "make sense of" the information we receive, and we often respond to the information with some sort of physical response that involves controlled movement. Although we wait until elementary school to learn how to add, an infant just a few weeks old, on seeing its mother's face, recognizes that face and smiles; soon that infant understands the spoken word. You spot the tennis ball in the yard even though it is green and nestled in among other green things (grass, dandelions). You register whether the tennis ball is close or far away, and you manipulate your legs and feet to propel you in the right direction.

How do we do these things? Traditional step-by-step procedural algorithms don't seem to apply, or if they do, we have not yet learned what those algorithms are. Rather, it seems that we as humans succeed at these tasks by processing a huge amount of data and then matching the results against an even larger storehouse of data based on our past experiences. Consider the task of recognizing your best friend. You have, in effect, been shown a number of "pictures" of your friend's face that seem to be "burned into" your memory, along with pictures of the faces of everyone else you know well. When you see your friend, you sort through your mental picture file until you come to a match. It is a bit more complicated than that, however, because if you encounter your friend's sister, you might know who it is even though you have never met her before. If your friend has a different haircut or has started wearing glasses, you will most likely still recognize her or him, in spite of a changed appearance. It would seem that you do not need to find an exact match to one of the images in your mental picture file, but only a reasonably close approximation. Approximation, unlike the exactitude required in computational tasks, is good enough. These complex recognition tasks that we find so easy can be difficult for computers, although facial detection (to aid in focusing) is now a feature of some digital cameras and facial recognition is a part of some photo management software as well as programs used by law enforcement to identify criminals entering a building or boarding an airplane.

When humans perform reasoning tasks, they are also using a large storehouse of experience. This experience involves not just images but also cause-and-effect situations. You know that you should wear a coat when it's cold because you've experienced discomfort in cold weather when you didn't wear a coat. This could be considered "mere" commonsense reasoning, but getting a computer to mimic common sense, to say nothing of higher-order conceptual, planning, or reasoning tasks, is extremely challenging. There may be no "right" answer to such tasks, and the way humans arrive at their respective answers sometimes seems ambiguous or based at least in part on intuition, which may be just another name for knowledge or reasoning that we don't yet understand.

Figure 15.2 summarizes what we've outlined as the relative capabilities of humans and computers in these three types of tasks. Computers fall below humans when procedural algorithms either don't work or aren't known, and when there seems to be a high level of complexity and perhaps approximation or ambiguity. Artificial intelligence seeks ways to improve the computer's ability to perform recognition and reasoning tasks, and we'll look at artificial intelligence approaches in these two areas in the rest of this chapter. As mentioned earlier, however, both types of tasks seem to require a storehouse of information—images, memories, past experiences—for which we'll use the general term *knowledge*. Therefore, we'll first look at various approaches to representing this knowledge.

Figure 15.2Human and computer capabilities

Predicted AI Milestones

In 2015, a group of academic and industry experts in artificial intelligence from around the world were asked to predict when they thought artificial intelligence would be able to outperform humans at various tasks. The average opinions on a few of these tasks were as follows:

Translate languages	2024
Assemble any LEGO kit	2024
Win at the game of Go*	2028

Drive a truck	2027
Write a New York Times best-seller	2049
Perform surgery	2053
All human jobs	2136

Of course, these are only predictions, so they must be taken with much skepticism. Still, do you suppose the citizens of 1900 would have predicted that the first commercial flight school would be opened 10 years later?

15.3 Knowledge Representation

We can consider knowledge about some topic as a body of facts or truths. For the computer to make use of that knowledge, there must be some digital format in which the knowledge is represented and stored within the computer. (At the lowest level, of course, only 0s and 1s are stored within the computer, but strings of 0s and 1s are organized and interpreted at a higher level of abstraction—as integers or characters, for example.) For computational tasks, the relevant knowledge is often isolated numeric or textual items. This is the data that we've manipulated with procedural programs. What about more complex knowledge?

There are many workable representation schemes; let's consider four possibilities.

1. *Natural language*—A paragraph or a page of text that contains all the knowledge we are trying to capture is written in English, Chinese, Spanish, or some other natural language. Here is an example:

   Note that although this representational form is text, it is text in a different sense from the character strings that are used in computational tasks. Here it is not simply the strings of characters that are important but also the meaning that those strings of characters convey. When reading a natural language paragraph, we use our understanding of the richness of the language's vocabulary to extract the meaning. Some researchers believe that the words we read or hear do not actually communicate meaning, but merely act as "triggers" to meanings stored in our brains.

2. *Formal language*—A formal language sacrifices richness of expression for precision of expression. Attributes and cause-and-effect relationships are more explicitly stated. A formal language version of the foregoing natural language paragraph might look like this:

   The term *language* was used in Chapter 11 to mean the set of statements derivable by using the rules of a grammar. But here, the term **formal language** means the language of formal logic, usually expressed more symbolically than we have done in this example. In the notation of formal logic, we might use *dog*(x) to symbolize that the symbolic entity *x* has the attribute of being a dog and brown(*x*) to mean that *x* has the attribute of being brown. Similarly *four-legged*(*x*), tail(x), *mammal*(*x*), and *warm-blooded*(*x*) could symbolize that *x* has these various attributes. The specific entity Spot could be represented by *S*. Then *dog*(*S*) would mean that Spot has the attribute of being a dog. Cause-and-effect relationships are translated into "if-then" statements. Thus, "Every dog has four legs" is equivalent to "For every *x*, if *x* is a dog, then *x* has four legs." An arrow symbolizes cause and effect (if-then); "If *x* is a dog, then *x* has four legs" would be written symbolically as

To show that every *x* that has the dog property also has the four-legged property, we would use a universal quantifier, , which means "for every *x*." Therefore, means "For every *x*, if *x* is a dog, then *x* has four legs" or "Every dog has four legs." Symbolically, the preceding six formal language statements become

Natural Language Statement	***Symbolic Representation***
Spot is a dog.	*dog*(S)
Spot is brown.	*brown*(S)
Every dog has four legs.	
Every dog has a tail.	
Every dog is a mammal.	
Every mammal is warm-blooded.	

The use of formal languages represents one of the major approaches to building artificial intelligence systems. Intelligent behavior is achieved by using symbols to represent knowledge and by manipulating these symbols according to well-defined rules. We'll see an example of this when we discuss expert systems later in this chapter.

3. *Pictorial*—Information can be stored in pictorial form as an image—a grid of pixels that have attributes of shading and color. Using this representation, we might have a picture of Spot, showing that he is brown and has four legs and a tail. We might have some additional labeling that says something like, "This is Spot, the dog." This visual representation might contain additional knowledge about Spot's appearance that is not embodied in the natural language paragraph or the formal language statements, but it would also fail to capture the knowledge that Spot is a mammal and that mammals are warmblooded. It also wouldn't tell us that all dogs have four legs and a tail. (After all, a photo of a three-legged dog does not tell us that all dogs have three legs.)
4. *Graphical*—Here, we are using the term *graphical* not in the sense of "visual" (we have already talked about pictorial representation) but in the mathematical sense of a graph with nodes and connecting arcs. Figure 15.3 is such a graph, also called a **semantic net**, for our dog example. In the terminology of object orientation that was a feature of the programming language(s) of Chapter 9, the rectangular nodes represent classes or objects, the oval nodes represent properties, and the arcs represent relationships. The "is a" relationship represents a subclass of a class that inherits properties from the parent class; "dog" is a subclass of "mammal," and any dog object inherits all the properties of mammals in general, such as being warm-blooded. Objects from the dog class may also have properties of their own. The "instance" relationship shows that something is an object of a class; Spot is a particular object from the dog class and may

have a unique property not necessarily shared by all dogs. The “is”, “has”, “is color” relationships specify properties of a class (the “mammal” class or the “dog”’ class) or of a specific object (Spot).

Figure 15.3A semantic net representation

Any knowledge representation scheme that we select must have the following four characteristics:

1. *Adequacy*—The representation method must be adequate to capture all of the relevant knowledge. Because of its rich, expressive powers, a natural language representation will surely capture a lot of knowledge. However, it might be difficult to extract exactly what that knowledge is. You might have to wade through a lot of unnecessary verbiage (as we discussed in Chapter 2, this is the reason for using pseudocode instead of natural language to describe an algorithm), and you must also understand the nuances of meaning within the natural language. A formal language representation has the advantage of extracting the essentials.
2. *Efficiency*—We want the representational form to be minimalist, avoiding redundant information wherever possible. This means allowing some knowledge that is not explicitly represented to be inferred from the knowledge that is explicitly represented. In the preceding example, it is easy to infer from the natural language, the formal language, or the semantic net that because Spot is a dog, he has four legs and a tail and also is a mammal and, therefore, warm-blooded. This knowledge, as we have said, is not captured in the pictorial format. On the other hand, it would take a much longer natural language paragraph to describe all the additional knowledge about Spot that is captured in the picture.
3. *Extendability*—It should be relatively easy to extend the representation to include new knowledge as it is acquired. For example, the semantic net can easily be extended to tack on another “dog” instance. It would also be easy to capture the fact that dogs have two eyes or that mammals do not lay eggs; these properties can simply be plugged in as new ovals connected into the network.
4. *Appropriateness*—The representation scheme used should be appropriate for the knowledge domain being represented. For example, a pictorial representation scheme would appear to be the most appropriate way to represent the knowledge base for a problem dealing with recognition of visual images, such as identifying your best friend. We saw before that a pictorial representation is probably not appropriate for the kind of knowledge about Spot that is difficult to display visually. The level of granularity needed for the intended application might also influence the appropriateness of a particular scheme. Is a given pictorial representation sufficient, or do we need to “zoom in” and expose more detail? The appropriate representational form for knowledge therefore depends on the knowledge to be captured and on the type of task for which the knowledge is to be used.

Practice Problems

Write a natural language paragraph that describes the concept of a hamburger. Now draw a semantic net that incorporates the same knowledge as your natural language description. Which one is easier for you to produce?

Answer

For example, a hamburger is a kind of sandwich. As such, it comes between two pieces of bread, but it is hot. It must contain ground meat, but it may also have various condiments, such as mustard, ketchup, and a pickle

Convert the following natural language statements to formal symbolic representation, using the properties sandwich(x), hamburger(x), grilledCheese(x), onBread(x), vegetarian(x):

Every hamburger is a sandwich

Every grilled cheese is a sandwich

All sandwiches are on bread

Every grilled cheese is vegetarian

Answer

a.

b.

c.

d.

ChangMain content

Chapter Contents

15.4 Recognition Tasks

If artificial intelligence aims to make computers "think" like humans, then it is natural to investigate and perhaps attempt to mimic the way the human brain functions. It is estimated that the human brain contains about 86 billion neurons. Each neuron is a cell capable of receiving stimuli, in the form of electrochemical signals, from other neurons through its many dendrites (Figure 15.4). In turn, it can send stimuli to other neurons through its single axon. The axon of a neuron does not directly connect with the dendrites of other neurons; rather, it sends signals over small gaps called synapses. Some of the synapses appear to send the neuron activating stimuli, whereas others seem to send inhibiting stimuli. A single neuron collects all the stimuli passing through all the synapses around its dendrites. The neuron sums the activating (positive) and inhibiting (negative) stimuli it receives and compares the result with an internal "threshold" value. If the sum equals or exceeds the threshold value, then the neuron "fires," sending its own signal down its axon to affect other neurons.

Figure 15.4A neuron

Each neuron can be thought of as an extremely simple computational device with a single on/off output. The power of the human brain lies in the vast number of neurons, the many interconnections between them, and the activating/inhibiting nature of those connections. To borrow a term from computer science, the human brain uses a **connectionist architecture**, characterized by a large number of simple "processors" with multiple interconnections. This contrasts quite noticeably with the Von Neumann architecture discussed in Chapter 5 that is still the basis for most computers today. In that model, there are a small number (maybe only one) of very powerful processors with a limited number of interconnections between them. Even the fastest parallel computer in the world as of 2016 had a little over 10.5 million processors rather than the 86 billion found in the human brain—about 0.01%.

In some areas of the brain, an individual neuron may collect signals from as many as 100,000 other neurons and send signals to an equally large number of neurons. This extensive parallelism is evidently required because of the relatively slow time frame within which a neuron fires. In the human brain, neurons operate on a time scale of milliseconds (thousandths of a second), as opposed to the nanoseconds (billionths of a second) in which computer operations are measured, a difference of 6 orders of magnitude. In a human processing task that takes about 1/10 second (recognition of your friend's face), the number of steps that can be executed by a single neuron would be on the order of 100. In the same time period, a typical computer processor could perform about 1 billion machine language operations. To carry out the complexity of a recognition task, then, requires the parallel activities of a large number of neurons executing cooperatively within this short time frame. In addition, massive parallelism supplies redundancy so that information is not stored only in one place but is shared within the network of neurons. Thus, the deterioration of a limited number of individual neurons (a process that happens constantly as biological cells wear out) does not cause a failure of the information processing capabilities of the network.

Artificial intelligence systems for recognition tasks have tried to mimic this connectionist approach. *Artificial neural networks*, usually just called **neural networks**, can be created by simulating individual neurons in hardware and connecting them in a massively parallel network of simple devices that act somewhat like biological neurons. Alternatively, the effect of a neural network may be simulated in software on an ordinary sequential-processing computer. In either case, each neuron has a threshold value, and its incoming lines carry weights that represent stimuli. The neuron fires when the sum of the incoming weights equals or exceeds its threshold value; the input lines are activated via the firing of other neurons.

Figure 15.5 represents a neuron with a threshold value of 3 and three input lines with weights of 2, – 1, and 2, respectively. If all three input lines are activated, the sum of the incoming signals is and the neuron fires. It also fires if only lines 1 and 3 are activated because the sum of the incoming signals is then . Any other combination of activated input lines cannot carry sufficient stimulation to fire the neuron. (Real, biological neurons fire with intensities that vary through a continuous range but, as usual, our simplified computer representation of such analog values uses a set of discrete values.)

Figure 15.5 One neuron with three inputs

Figure 15.6 depicts a neural net with an input layer and an output layer of neurons. An input value is presented to neuron in the input layer via a line with signal strength The values of are usually binary (0 or 1), so that this line carries a signal of either 0 when *x* is 0, or the weight . when is 1. The weights to the input neurons, as well as the weights from the input layer to the output layer, can be positive, negative, or zero.

Figure 15.6 Neural network model

In the neural network shown in Figure 15.7, we have eliminated connections of weight 0 and all remaining connections have weight 1. Here and have binary values of 0 or 1. If or or both have the value 1, then a signal of 1 is passed to one or both of the neurons in the input layer, causing one or both of them to fire, which causes the single neuron in the output layer to fire and produce an output of 1. If both and have the value 0, then neither neuron in the input layer fires, the single neuron in the output layer does not fire, and the network output is 0. This neural network is acting like an OR gate (refer to Figure 4.17).

Figure 15.7 **A simple neural network—OR gate**

It turns out to be impossible to build such a network to represent the Boolean operation called exclusive OR, or XOR, whose truth table is shown in Figure 15.8. Here, the output is true (1) when one or the other input is true, but not when both are true. In Figure 15.9, no matter what values we give for the weights and thresholds, it is not possible to generate this behavior. If exactly one input signal of 1 is enough to fire the output neuron, which is the desired behavior, then two input signals of 1 can only increase the tendency for the output neuron to fire.

Figure 15.8 **The truth table for XOR**

Figure 15.9 **An attempt at an XOR network**

To represent the XOR operation requires a “hidden layer” of neurons between the input and output layers. Neural networks with a hidden layer of neurons are useful for recognition tasks, where we want a certain pattern of output signals for a certain pattern of input signals. The XOR network, for example, recognizes when its two binary inputs do not agree. (Exercise 7 at the end of this chapter asks you to complete a neural network for the XOR operation.)

Conventional computer processing works on a knowledge base where the information is stored as data in specific memory cells that can be accessed by the program as needed. In a neural network, both the knowledge representation and the “programming” are stored in the network itself as the weights of the connections and the thresholds of the neurons. If you want to build a neural network that performs in a certain way, how do you determine these values? In a simple network, trial and error can produce a solution, but such is not the case for a network with thousands of neurons. Fortunately, the right answer doesn’t have to be found the first time. Remember that neural networks are modeled on the human brain; you learned to recognize your best friend through repeated “learning experiences” that modified your knowledge base until you came to associate certain features or characteristics with that individual.

Similarly, a neural network can learn from experience by modifying the weights on its connections (even making some connections “disappear” by assigning them 0 weights). A network can be given a set of weights and thresholds that is simply an initial educated guess. The network is then presented with **training data**, for which the correct outputs are known. The actual output from the network is compared with the correct output for one set of input values from the training data. For those output neurons that produce correct values, their threshold values and the weights on their inputs do not change. Output neurons that produce erroneous values can err in one of two ways. If an output neuron fires when it is not supposed to, then the positive (excitatory) input values coming into it are adjusted downward, and the negative (inhibitory) weights coming into it are adjusted upward. If it fails to fire when it is supposed to, the opposite adjustment is made. But before these adjustments take place, information on the errors is passed back from each erroneous output neuron to the neurons in the hidden layer that are connected to it. Each hidden-layer neuron adds these error counts to derive an estimate of its own error. This estimate is used to calculate the adjustments to be made on the weights of the connections coming to it from the input-layer neurons. Finally, the weights are all adjusted, and then the process is repeated for the next set of input values from the training data.

This **back propagation algorithm**, so named for the error estimates that are passed back from the output layer, eventually causes the network to settle into a stable state where it can correctly respond, to any desired degree of accuracy, to all inputs in the training set. In effect, the successive changes in weights have reinforced good behavior and discouraged bad behavior (much as we train our pets) until the paths for good behavior are imprinted on the connections (as in Fido's brain). The network has "learned" what the proper connection weights should be, and its ability to recognize the training data is embedded somehow in the collective values of these weights. At the end of its training, the neural network is ready to go to work on new recognition problems that are similar to, but not the same as, the training data and for which the correct answers are unknown.

This process is an example of **machine learning**, where, without specific step-by-step programming, computing agents learn and improve from past errors made on known training data. Then, when deemed sufficiently trained, they can be set loose on data with no known answers. Machine learning is also one of the tools of data science, where the objective is to discover previously unknown patterns in large amounts of data.

Neural networks have found their way into dozens of real-world applications. A few of these are handwriting recognition, speech recognition, identifying billing patterns indicative of credit card fraud, predicting the odds of susceptibility to cancer, analyzing magnetic resonance images in medicine, adapting mirror shapes for astronomical observations, and discovering the best routing algorithm in a large communications network (a problem we mentioned in Chapter 7). With the ever-decreasing cost of massively parallel networks, it appears that neural networks will continue to find new applications.

Brain on a Chip

In August 2014, IBM announced a new computer chip that mimics the way the brain works. The chip, called TrueNorth, consists of 4,096 " neurosynaptic" cores arranged in a grid. Each core contains 256 programmable neurons (think of programming here as setting the neuron threshold to control whether the neuron fires) and 256 axons (think of these as input signals). Information travels from axons to neurons controlled by programmable synapses (think of programming here as setting the weights on connection lines). Altogether the TrueNorth chip contains 5.4 billion transistors, over 1 million programmable neurons, and over 260 million programmable synapses. All this processing power

consumes energy at about the rate of a hearing aid battery. And for scalability the chips can also be connected together. Talk about parallel processing!
This milestone chip is the result of work begun in 2008 funded by DARPA (Defense Advanced Research Projects Agency) under a program called Systems of Neuromorphic Adaptive Plastic Scalable Electronics, or SyNAPSE. An estimate, based on the number of neurons, is that TrueNorth has roughly the complexity of a honeybee's brain, which has approximately 960,000 neurons. So TrueNorth, while mimicking the way real neurons work in biological brains, has a long way to go to match human brainpower.

The trend in machine learning since 2012 has primarily been in so-called deep *learning*. Deep learning uses fairly standard computer architecture to simulate neural networks with multiple layers (that's where the "deep" comes in) and back propagation from the output layer all the way back to the input layer to improve accuracy, much as we have described. The focus is on developing neural networks for specific AI applications such as image recognition. However, in September 2016, IBM announced a breakthrough in the usefulness of TrueNorth. A new algorithm had been developed that allowed deep learning networks to be built on TrueNorth's brain-simulating hardware. The resulting new system has highly accurate image recognition capability while being scalable, much more energy efficient, and faster than traditional deep learning architectures.

Practice Problems

If input line 1 is stimulated in the following neural network (and line 2 is not stimulated), will the output line fire? Explain.

Answer

No. N1 and N4 fire, but N2 and N3 do not, so N5 does not.

If the firing threshold on node N5 is changed from its current value of 3 to 5, under what conditions will it fire?

Answer

It will never fire. The maximum incoming signal that node N5 can receive is , which is not enough to make it fire if its threshold value is 5.

Main content

Chapter Contents

15.4 Recognition Tasks

If artificial intelligence aims to make computers "think" like humans, then it is natural to investigate and perhaps attempt to mimic the way the human brain functions. It is estimated that the human brain contains about 86 billion neurons. Each neuron is a cell capable of receiving stimuli, in the form of electrochemical signals, from other neurons through its many dendrites (Figure 15.4). In turn, it can send stimuli to other neurons through its single axon. The axon of a neuron does not directly connect with the dendrites of other neurons; rather, it sends signals over small gaps called synapses. Some of the synapses appear to send the neuron activating stimuli, whereas others seem to send inhibiting stimuli. A single neuron collects all the stimuli passing through all the synapses around its dendrites. The neuron sums the activating (positive) and inhibiting (negative) stimuli it receives and compares the result with an internal "threshold" value. If the sum equals or exceeds the threshold value, then the neuron "fires," sending its own signal down its axon to affect other neurons.

Figure 15.4A neuron

Each neuron can be thought of as an extremely simple computational device with a single on/off output. The power of the human brain lies in the vast number of neurons, the many interconnections between them, and the activating/inhibiting nature of those connections. To borrow a term from computer science, the human brain uses a **connectionist architecture**, characterized by a large number of simple "processors" with multiple interconnections. This contrasts quite noticeably with the Von Neumann architecture discussed in Chapter 5 that is still the basis for most computers today. In that model, there are a small number (maybe only one) of very powerful processors with a limited number of interconnections between them. Even the fastest parallel computer in the world as of 2016 had a little over 10.5 million processors rather than the 86 billion found in the human brain—about 0.01%.

In some areas of the brain, an individual neuron may collect signals from as many as 100,000 other neurons and send signals to an equally large number of neurons. This extensive parallelism is evidently required because of the relatively slow time frame within which a neuron fires. In the human brain, neurons operate on a time scale of milliseconds (thousandths of a second), as opposed to the nanoseconds (billionths of a second) in which computer operations are measured, a difference of 6 orders of magnitude. In a human processing task that takes about 1/10 second (recognition of your friend's face), the number of steps that can be executed by a single neuron would be on the order of 100. In the same time period, a typical computer processor could perform about 1 billion machine language operations. To carry out the complexity of a recognition task, then, requires the parallel activities of a large number of neurons executing cooperatively within this short time frame. In addition, massive parallelism supplies redundancy so that information is not stored only in one place but is shared within the network of neurons. Thus, the deterioration of a limited number of individual neurons (a process that happens constantly as biological cells wear out) does not cause a failure of the information processing capabilities of the network.

Artificial intelligence systems for recognition tasks have tried to mimic this connectionist approach. *Artificial neural networks*, usually just called **neural networks**, can be created by simulating individual neurons in hardware and connecting them in a massively parallel network of simple devices that act somewhat like biological neurons. Alternatively, the effect of a neural network may be simulated in software on an ordinary sequential-processing computer. In either case, each neuron has a threshold value, and its incoming lines carry weights that represent stimuli. The neuron fires when the sum of the incoming weights equals or exceeds its threshold value; the input lines are activated via the firing of other neurons.

Figure 15.5 represents a neuron with a threshold value of 3 and three input lines with weights of 2, – 1, and 2, respectively. If all three input lines are activated, the sum of the incoming signals is and the neuron fires. It also fires if only lines 1 and 3 are activated because the sum of the incoming signals is then . Any other combination of activated input lines cannot carry sufficient stimulation to fire the neuron. (Real, biological neurons fire with intensities that vary through a continuous range but, as usual, our simplified computer representation of such analog values uses a set of discrete values.)

Figure 15.5One neuron with three inputs

Figure 15.6 depicts a neural net with an input layer and an output layer of neurons. An input value is presented to neuron in the input layer via a line with signal strength The values of are usually binary (0 or 1), so that this line carries a signal of either 0 when *x* is 0, or the weight . when is 1. The weights to the input neurons, as well as the weights from the input layer to the output layer, can be positive, negative, or zero.

Figure 15.6Neural network model

In the neural network shown in Figure 15.7, we have eliminated connections of weight 0 and all remaining connections have weight 1. Here and have binary values of 0 or 1. If or or both have the value 1, then a signal of 1 is passed to one or both of the neurons in the input layer, causing one or both of them to fire, which causes the single neuron in the output layer to fire and produce an output of 1. If both and have the value 0, then neither neuron in the input layer fires, the single neuron in the output layer does not fire, and the network output is 0. This neural network is acting like an OR gate (refer to Figure 4.17).

Figure 15.7A simple neural network—OR gate

It turns out to be impossible to build such a network to represent the Boolean operation called exclusive OR, or XOR, whose truth table is shown in Figure 15.8. Here, the output is true (1) when one or the other input is true, but not when both are true. In Figure 15.9, no matter what values we give for the weights and thresholds, it is not possible to generate this behavior. If exactly one input signal of 1 is enough to fire the output neuron, which is the desired behavior, then two input signals of 1 can only increase the tendency for the output neuron to fire.

Figure 15.8The truth table for XOR

Figure 15.9An attempt at an XOR network

To represent the XOR operation requires a “hidden layer” of neurons between the input and output layers. Neural networks with a hidden layer of neurons are useful for recognition tasks, where we want a certain pattern of output signals for a certain pattern of input signals. The XOR network, for example, recognizes when its two binary inputs do not agree. (Exercise 7 at the end of this chapter asks you to complete a neural network for the XOR operation.)

Conventional computer processing works on a knowledge base where the information is stored as data in specific memory cells that can be accessed by the program as needed. In a neural network, both the knowledge representation and the "programming" are stored in the network itself as the weights of the connections and the thresholds of the neurons. If you want to build a neural network that performs in a certain way, how do you determine these values? In a simple network, trial and error can produce a solution, but such is not the case for a network with thousands of neurons. Fortunately, the right answer doesn't have to be found the first time. Remember that neural networks are modeled on the human brain; you learned to recognize your best friend through repeated "learning experiences" that modified your knowledge base until you came to associate certain features or characteristics with that individual.

Similarly, a neural network can learn from experience by modifying the weights on its connections (even making some connections "disappear" by assigning them 0 weights). A network can be given a set of weights and thresholds that is simply an initial educated guess. The network is then presented with **training data**, for which the correct outputs are known. The actual output from the network is compared with the correct output for one set of input values from the training data. For those output neurons that produce correct values, their threshold values and the weights on their inputs do not change. Output neurons that produce erroneous values can err in one of two ways. If an output neuron fires when it is not supposed to, then the positive (excitatory) input values coming into it are adjusted downward, and the negative (inhibitory) weights coming into it are adjusted upward. If it fails to fire when it is supposed to, the opposite adjustment is made. But before these adjustments take place, information on the errors is passed back from each erroneous output neuron to the neurons in the hidden layer that are connected to it. Each hidden-layer neuron adds these error counts to derive an estimate of its own error. This estimate is used to calculate the adjustments to be made on the weights of the connections coming to it from the input-layer neurons. Finally, the weights are all adjusted, and then the process is repeated for the next set of input values from the training data.

This **back propagation algorithm**, so named for the error estimates that are passed back from the output layer, eventually causes the network to settle into a stable state where it can correctly respond, to any desired degree of accuracy, to all inputs in the training set. In effect, the successive changes in weights have reinforced good behavior and discouraged bad behavior (much as we train our pets) until the paths for good behavior are imprinted on the connections (as in Fido's brain). The network has "learned" what the proper connection weights should be, and its ability to recognize the training data is embedded somehow in the collective values of these weights. At the end of its training, the neural network is ready to go to work on new recognition problems that are similar to, but not the same as, the training data and for which the correct answers are unknown.

This process is an example of **machine learning**, where, without specific step-by-step programming, computing agents learn and improve from past errors made on known training data. Then, when deemed sufficiently trained, they can be set loose on data with no known answers. Machine learning is also one of the tools of data science, where the objective is to discover previously unknown patterns in large amounts of data.

Neural networks have found their way into dozens of real-world applications. A few of these are handwriting recognition, speech recognition, identifying billing patterns indicative of credit card fraud, predicting the odds of susceptibility to cancer, analyzing magnetic resonance images in medicine, adapting mirror shapes for astronomical observations, and discovering the best routing algorithm in a large communications network (a problem we mentioned in Chapter 7). With the ever-decreasing cost of massively parallel networks, it appears that neural networks will continue to find new applications.

Brain on a Chip

In August 2014, IBM announced a new computer chip that mimics the way the brain works. The chip, called TrueNorth, consists of 4,096 " neurosynaptic" cores arranged in a grid. Each core contains 256 programmable neurons (think of programming here as setting the neuron threshold to control whether the neuron fires) and 256 axons (think of these as input signals). Information travels from axons to neurons controlled by programmable synapses (think of programming here as setting the weights on connection lines). Altogether the TrueNorth chip contains 5.4 billion transistors, over 1 million programmable neurons, and over 260 million programmable synapses. All this processing power consumes energy at about the rate of a hearing aid battery. And for scalability the chips can also be connected together. Talk about parallel processing!
This milestone chip is the result of work begun in 2008 funded by DARPA (Defense Advanced Research Projects Agency) under a program called Systems of Neuromorphic Adaptive Plastic Scalable Electronics, or SyNAPSE. An estimate, based on the number of neurons, is that TrueNorth has roughly the complexity of a honeybee's brain, which has approximately 960,000 neurons. So TrueNorth, while mimicking the way real neurons work in biological brains, has a long way to go to match human brainpower.

The trend in machine learning since 2012 has primarily been in so-called deep *learning*. Deep learning uses fairly standard computer architecture to simulate neural networks with multiple layers (that's where the "deep" comes in) and back propagation from the output layer all the way back to the input layer to improve accuracy, much as we have described. The focus is on developing neural networks for specific AI applications such as image recognition. However, in September 2016, IBM announced a breakthrough in the usefulness of TrueNorth. A new algorithm had been developed that allowed deep learning networks to be built on TrueNorth's brain-simulating hardware. The resulting new system has highly accurate image recognition capability while being scalable, much more energy efficient, and faster than traditional deep learning architectures.

Practice Problems

If input line 1 is stimulated in the following neural network (and line 2 is not stimulated), will the output line fire? Explain.

Answer

No. N1 and N4 fire, but N2 and N3 do not, so N5 does not.

If the firing threshold on node N5 is changed from its current value of 3 to 5, under what conditions will it fire?

Answer

It will never fire. The maximum incoming signal that node N5 can receive is , which is not enough to make it fire if its threshold value is 5.

Main content

Chapter Contents

15.5.1 Intelligent Searching

Earlier in this book, we presented two algorithms for searching—sequential search and binary search. These search algorithms look for a perfect match between a specific target value and an item in a list. The amount of work involved is for sequential search and for binary search.

A *decision tree* for a search algorithm illustrates the possible next choices of items to search if the current item is not the target. In a sequential search, there is only one item to try next: the next item in the list. The decision tree for sequential search is therefore linear, as shown in Figure 15.10. A decision tree for a binary search (in which the search items are in sorted order), such as the one shown in Figure 15.11, reflects the fact that if the current item is not the target, there are only two next choices: the midpoint of the sublist before this node or the midpoint of the sublist after this node. Furthermore, the binary search algorithm specifies which of the two nodes to try next.

Figure 15.10 Decision tree for sequential search

Figure 15.11 Decision tree for binary search

The classical search problem benefits from two simplifications:

1. The search domain (the set of items being searched) is highly constrained. At each point in the search, if the target is not found, the choice of where to look next is determined.
2. We seek a perfect match, so the comparison of the target against the list item results in a binary decision—either they match or they do not.

Suppose, however, that condition 1 does not hold; the search domain is such that after any one node has been searched (unsuccessfully), there are an enormous number of potential next choices, and there is no algorithm to dictate which of these next choices is best. Figure 15.12 attempts to portray this scenario. In the terminology of artificial intelligence, such a figure is called a **state-space graph**, and we seek to perform a **state-space search** to find a *solution path* through the graph. The idea is that each node of the graph represents a "state" of our problem, and we have some "goal state" or states in mind. For example, in a game of tic-tac-toe, our initial state is the empty game grid, and our goal state is a winning configuration. A solution path takes us from the initial state to a winning configuration, and the graph nodes along the way represent the intermediate configurations. In addition to finding a winning sequence of moves for a board game (tic-tac-toe, checkers, chess, and so forth), many other types of problems, such as finding the shortest path through a network or finding the most successful investment strategy in the stock market, fall into the state-space search category. In some of these problems, condition 2 of the classical search problem—that of seeking an exact match with a specified target value—is not present either. We simply want to acquire as many characteristics of the desired goal as possible, and we need some measure of when we are “close enough.”

Figure 15.12 **A state-space graph with exponential growth**

A *brute force* approach for finding a solution path traces all branches of the state-space graph so that all possible choices are tested and no test cases are repeated. This becomes a massive bookkeeping task because the number of branches grows exponentially. Given that time and computing resources are limited, an intelligent search needs to be employed. An intelligent search narrows the number of branches that must be tried and thereby puts a cap on the otherwise exponential growth of the problem. Intelligent searching involves applying some **heuristic** (which means, roughly, an “educated guess”) to evaluate the differences between the present state and the goal state and to move us to a new state that minimizes those differences—namely, the state that maximizes our progress toward the goal state.

An intelligent chess-playing strategy, for example, is one that makes an appropriate first move and that, at each step, makes a move more likely than others to lead to a winning board configuration. Even a grand master of chess cannot pursue the brute force approach of mentally trying out all possible next moves, all the possible moves that follow from each of those moves, and so on, for very many steps. (In Section 1.2, we showed that using a brute force approach, a computer would require a billion billion billion years to make its first move!) Intelligent searching is required to prune the number of potential candidates to a reasonable size. There must be a deep storehouse of experience that can be “consulted” on the basis of the present configuration of the board. A grandmaster-level player may need a mental database of around 50,000 of these board configurations, each with its associated information about the best next move.

Building a machine that can beat a human at chess was long thought to be a supreme test of artificial intelligence—machines that “think.” Successfully playing chess, it was believed, surely epitomized logical reasoning, true “intelligence.” Chess is difficult for humans. Yet, the rules for chess are straightforward; it is simply the size of the state-space that is overwhelming. As artificial intelligence researchers delved deeper into supposedly “simpler” problems such as visual recognition or natural language understanding—things we humans do easily—it became clear that these were the harder challenges for machines. Playing chess came to be viewed as the last of the “easy” hard problems.

Change font sizeMain content

Chapter Contents

15.5.2 Swarm Intelligence

Recall that the connectionist architecture—neural networks—draws its inspiration from nature, namely, the human brain. Another approach to achieving a desired end, *swarm intelligence*, also draws its inspiration from nature, modeling the behavior of, for example, a colony of ants. Each ant is an unsophisticated creature with limited capabilities, yet acting as a collective, an ant colony can accomplish remarkable tasks. Ants can find the shortest route from a nest to a food source, carry large items, emigrate as a colony from one location to another, and form bridges. An ant “communicates” with other ants by laying down a scent trail, called a *pheromone trail*; other ants follow this trail and reinforce its strength by laying down their own pheromones. Given a choice, ants have a higher probability of following the strongest pheromone trail. Hence, the ant that took the shortest path to food and returned to tell about it lays down a trail that other ants follow and reinforce faster than the trail laid down by an ant that took a longer path. Because pheromone trails evaporate quickly, the collective intelligence of the colony is constantly updated to respond to current conditions of its environment.

The **swarm intelligence model** captures this collective behavior. Computer scientists create algorithms that simulate the process of having simple agents (analogous to the ants) operate independently and follow each other’s “trails” to find the most efficient routes. This algorithmic approach is called Ant Colony Optimization (ACO) and has been used commercially in vehicle routing, job scheduling, and the sensing of biological or chemical contaminants. Studies have demonstrated the use of such simple agents in telecommunications networks to avoid the complexity of a centralized control system to compute and distribute routing tables within a network.

Robot Swarms

Several research groups are experimenting with swarms of tiny robots. These entities, like ants, have very limited capabilities as individuals, but they can communicate with one another and, as a collective, perform simple tasks. At Harvard’s School of Engineering and Applied Sciences, over 1,000 robots, each only a few centimeters across, can collectively form themselves into any given 2-D shape once an image of that shape has been presented to them. Individual robots, using their limited processing power and the artificial intelligence algorithms they have been given, can sense the edge of the group, track their distance from the origin, avoid collisions, and even clear up little traffic jams until, finally, they have formed the desired shape. The image here shows the Kilobots with Harvard research scientist Michael Rubenstein. To see these Kilobots in action, go to http://news.harvard.edu/gazette/story/2014/08/the-1000-robot-swarm.

Wendy Maeda/The Boston Globe/Getty Images

Studies (with smaller swarms) at the Sheffield Centre for Robotics in the United Kingdom have produced robotic collectives that can organize themselves by order of priority, and can fetch an object by surrounding it and working together to push it across the floor. Scientists foresee uses for robot swarms in environmental cleanup, search and rescue operations, or even in noninvasive micromedicine.

Main content

Chapter Contents

15.5.3Intelligent Agents

Swarm intelligence rests in the colony as a whole, which seems to acquire “knowledge” that is greater than the sum of its parts. At the opposite end of the spectrum are intelligent agents. An **intelligent agent** is a form of software technology that is designed to interact collaboratively with a user somewhat in the mode of a personal assistant.

Imagine that you have hired your own (human) personal assistant. In the beginning, you must tell your assistant what to do and how you want it done. Over time, however, your assistant comes to know more about you and soon can anticipate which tasks need to be done and how to perform them, which items to bring to your attention, and so forth. Your assistant becomes more valuable as he or she becomes more self-directed, always acting with your best interests in mind. You, in turn, put more and more trust in your assistant.

Like the human personal assistant, an intelligent agent does not merely wait for user commands but begins to initiate communication, anticipate what is required, take action, and perform tasks on its own on the basis of its growing knowledge of your needs and preferences. Here are some examples that exist today:

- A personalized web search engine that allows you to profile items of interest to you and then automatically delivers appropriate information from the web. For example, you may request updated weather conditions for your geographic area, along with news items related to sports and European trade. At periodic time intervals, this **push technology** downloads your updated, personalized information to your screen (or smartphone) to be displayed whenever no other task is active.
- A more intelligent version of this personalized web searcher that enables you to rate each article it sends you and then dynamically adjusts the information that it sends in the future as it learns about your preferences.
- An even more intelligent search agent that not only narrows down choices from topics you have chosen but can suggest new, related topics for you to explore. This is accomplished by having your agent communicate with similar agents on the web, even when you are not online. If your agent knows of your interest in French cuisine, for example, it communicates with other agents to find those that represent users with the same interest. It may learn from these agents that many of their users are also interested in red wines. Your agent then judges whether these suggestions are coming from agents whose recommendations on the whole have been well received by you in the past. If so, it asks whether you also want information about red wines. If you do not agree to this proposal, your agent notes which agents made that suggestion and, on the next pass, gives less consideration to their ideas. The more agents that participate, the more accurate each one becomes at "understanding" the interests of its user.
- The intelligent agent that "lives" in your smartphone (Siri for iPhones, Google Assistant for Android phones and iPhones), able to answer questions, tell you where the nearest hospital is, make dinner reservations for you at your favorite restaurant, remind you of items on your calendar, and much more. Or the intelligent agent that "lives" in your living room (Alexa on Amazon's Echo device, Google Assistant on the Google Home device), able to check your flight status, track a package, play music, tell you a joke, and more.
- An ecommerce company that uses **recommendation software** (an intelligent agent) to welcome a returning customer to its webpage and make suggestions on future purchases based on what this customer or other customers have done in the past. "Customers who bought this item also bought. . . ."
- A manufacturing plant that uses an intelligent agent to negotiate with suppliers on the price and scheduling of parts delivery to maximize efficiency of production.

Intelligent agent technology has been an area of interest in artificial intelligence for many years. However, intelligent agents need to display significantly greater learning capabilities and "common sense" before most users will trust them to make autonomous decisions regarding the allocation of time and money. Until then, they will be relegated to presenting suggestions to their human users. However, when a sufficient level of trust in intelligent agent technology has been achieved, and when human users are willing to allow their software to make independent decisions, we will take applications like the following for granted:

- *Financial agents* that negotiate with one another over the web for the sale and purchase of goods and services, using price/cost parameters set by the sellers and buyers (sort of an automated eBay)
- *Travel and tourism agents* (electronic, not human) that book airline flights, rent automobiles, and make hotel reservations for you on the basis of your destination, schedule, price range, and preferences
- *Office manager agents* that screen incoming email, put meetings on their users' schedules, and draft replies

Main content

Chapter Contents

15.5.4 Expert Systems

Although intelligent agents incorporate a body of knowledge to "filter" their choices and thereby appear to capture certain aspects of human reasoning, they still perform relatively limited tasks. Consider the more unstructured scenario of managing the triage center in a busy hospital emergency room. The person in charge draws on

- (1)

  past experience and training to recognize various medical conditions (which may involve many recognition subtasks),
- (2)

  understanding of those conditions and their probable consequences, and
- (3)

  knowledge about the hospital's capabilities and resources in general and at the moment.

From this knowledge base, a chain of reasoning is followed that leads, for example, to a decision to treat patient A immediately in a particular fashion and to let patient B wait. We consider this to be evidence of quite general "logical reasoning" in humans.

Artificial intelligence simulates this kind of reasoning through the use of **expert systems**, also called rule-based systems or knowledge-based systems. (The latter term is a bit confusing because all "intelligent activity" rests on some base of knowledge.) An expert system attempts to mimic the human ability to engage pertinent facts and string them together in a logical fashion to reach some conclusion. An expert system must therefore contain these two components:

- A *knowledge base*—A set of facts about the subject matter
- An *inference engine*—A mechanism for selecting the relevant facts and for reasoning from them in a logical way

Note that the knowledge base contains facts about a *specific* subject domain to narrow the scope to a manageable size.

The facts in the knowledge base consist of certain simple assertions. For example, let's say that the domain of inquiry is U.S. presidents. Three simple assertions are

1. Lincoln was president during the Civil War.
2. Kennedy was president before Nixon.
3. FDR was president before Kennedy.

Another type of fact is a *rule*, a statement of the form *if* . . . *then* . . ., which says that whenever the clause following "if" is true, so is the clause following "then." For example, here are two rules that, taken together, define what it means for one president to precede another in office. In these rules, *X*, *Y*, and *Z* are variables.

I. If *X* was president before Y, then X precedes Y.
II. If *X* was president before *Z* and *Z* precedes *Y*, then *X* precedes *Y*.

What conclusions can be reached from this collection of three assertions and two rules? Assertion 2 says that Kennedy was president before Nixon. This matches the "if" clause of rule I, where *X* is Kennedy and *Y* is Nixon. From this, the "then" clause of rule I yields a new assertion, that Kennedy precedes Nixon, which we'll call assertion 4. Now assertion 3 says that FDR was president before Kennedy, and assertion 4 says that Kennedy precedes Nixon. This matches the "if" clause of rule II, where *X* is FDR, *Z* is Kennedy, and *Y* is Nixon. From this, the "then" clause of rule II yields a new assertion, that FDR precedes Nixon, which we'll call assertion 5. Hence,

- 4.

  Kennedy precedes Nixon.
- 5.

  FDR precedes Nixon.

are two new conclusions or assertions. These assertions were not part of the original knowledge base and were previously unknown. Instead, they were obtained from what was known through a process of logical reasoning. The knowledge base has been extended. We could also say that the system has *learned* two new pieces of knowledge.

If this example sounds familiar, it is because it is part of the example we used in Chapter 10 to illustrate the logic programming language Prolog. Prolog provides one means of implementing an inference engine for an expert system.

The inference engine is basically using the following pattern of reasoning:

This reasoning process, as we noted in Chapter 10, goes by the Latin name of *modus ponens*, which means "method of assertion." It gives us a method for making new assertions. We humans use this deductive reasoning process all the time, for example,

I know that if my grade on the math final was below 50, then I will fail the course. My grade on the final was below 50. I can infer exactly what is going to happen!

However, it is also suitable for computerization because it is basically a matching algorithm that can be implemented by brute force trial and error. Systems like Prolog, however, apply some additional guidelines in their search for matches to speed up the process; that is, they employ a form of intelligent searching.

Inference engines for expert systems can proceed in several ways. *Forward chaining* begins with assertions and tries to match those assertions to the “if” clauses of rules, thereby generating new assertions. These may in turn be matched with “if” clauses, generating still more assertions. This is the process we used in our example. *Backward chaining* begins with a proposed conclusion and tries to match it with the “then” clauses of rules. If successful, it next looks at the corresponding “if” clauses of those rules and tries to match those with assertions, or with the “then” clauses of other rules. This process continues until all “if” clauses that arise have been successfully matched with assertions, in which case the proposed conclusion is justified, or until no match is possible, in which case the proposed conclusion is rejected. Backward chaining in our example would start with the proposed conclusion that FDR precedes Nixon, and the system would then work backward to justify this conclusion.

In addition to the knowledge base and the inference engine, most rule-based systems also have an **explanation facility**. This allows the user to see the assertions and rules used in arriving at a conclusion, as a sort of check on the path of reasoning or for the user’s own enlightenment.

Of course, a rule-based system about some particular domain is only as good as the assertions and rules that make up the knowledge base. The builder of such a system acquires the information for the knowledge base by consulting “experts” in the domain and mining their expertise. This process, called **knowledge engineering**, requires a great deal of interaction with the human expert, much of it in the domain environment. If the domain expert is the manager of a chemical processing plant, for example, a decision to “turn down valve A whenever the temperature in pipe P exceeds and valves B and C are both closed” may be such an ingrained behavior that the expert won’t remember it as part of a question-and-answer session on “what you do on your job.” It only emerges by on-site observation. For the hospital example, one might need to follow people around in the emergency room, observe their decisions, and later question them on why those decisions were made. It is also possible to incorporate probabilities to model the thinking process, for example, “If the patient has fever and stomach pains, the probability of appendicitis is 73% and the probability of gall bladder problems is 27%, therefore I first check for A and then for B.”

Practice Problems

Given the assertion “Frank is bald” and the rule “If *X* is bald, then *X* is tall,” what conclusion can be inferred? If Frank were known to be tall, would that necessarily imply that he was bald?

Answer

Frank is tall. Knowing that Frank is tall does not necessarily mean that he is bald.

Given the assertion “Frank is not bald” and the rule “If *X* is bald, then *X* is tall,” what conclusion can be inferred?

Answer

No conclusion can be inferred. Frank might or might not be tall.

Given the assertion “Frank is bald” and the rule “If *X* is tall, then *X* is bald,” what conclusion can be inferred?

Answer

No conclusion can be inferred. Frank might or might not be tall.

Given the assertion “Frank is not bald” and the rule “If *X* is tall, then *X* is bald,” what conclusion can be inferred?

Answer

Frank is not tall (because if he were, he would be bald).

Expert systems have been implemented in many domains, including specific forms of medical diagnosis, computer chip design, monitoring of manufacturing processes, financial planning, purchasing decisions for retail stores, automotive troubleshooting, and diagnosis of failures in electronic systems. They will no doubt be even more commonplace in the future.

15.5.5 The Games We Play

These days, the world of video game playing draws many enthusiasts. We will explore some of the computer technologies that make video games possible in the next chapter. But here, we want to trace the progression of the “artificial intelligence” needed to have the computer play (rather than just display) a game.

Board Games. Almost everyone is familiar with the simple pencil-and-paper game of tic-tac-toe. Two players draw Xs and Os, respectively, in the squares of a grid. The first player to draw three of his or her symbol in a row (vertically, horizontally, or diagonally) wins. While the “X” player is trying to build such a row, the “O” player is trying to block it, and vice versa. For example, if the current configuration looks like this:

then the “O” player should write an O in the middle square to block X’s diagonal. The rules are pretty simple, making it a good children’s game. For experienced players, most games will end in a draw.

Writing a computer program to play tic-tac-toe is fairly easy. (The first-ever computer game was tic-tac-toe, written for the EDSAC computer in 1952; it played perfect games against human opponents. The first computer program Bill Gates wrote, at the age of 13, played tic-tac-toe.) The state-space graph (see Section 15.5.1) for tic-tac-toe is relatively small. Because there are nine positions, each of which can contain X, O, or blank, at first it seems that there are board configurations. However, many of these are essentiality the same. For example, a single X in a corner square with all other cells blank occurs four ways because there are four corners, but these are all rotations of the same thing, so this is really only one configuration. Eliminating these similarities, there are only 765 distinct configurations. We want a solution path from the initially empty board to a winning configuration, and because of the small state-space, a brute force approach is feasible. If we assume that each player can write in any vacant cell, that the game is terminated with any three-in-a-row symbol or a full grid, and that configurations that are essentially the same are ignored, there are 26,830 possible games (paths through the state space). By following the simple strategies of trying to make three in a row

while trying to block your opponent, and concluding the game when the outcome is determined, the number of paths through the state space is much smaller—1,145 games. These can be analyzed in their entirety to find a winning path.
Checkers is a board game with more complex rules than tic-tac-toe and a far larger state space; there are (500 billion billion) possible board configurations. The Chinook project, involving a group of computer scientists at the University of Alberta in Canada, headed by Professor Jonathan Schaeffer, began in 1989. This project was to develop a checkers-playing computer program. In April 2007, it was announced that Chinook was finally perfected. From the standard starting positions used in tournament play, Chinook can never lose; the best a skilled opponent can achieve is a draw. Multiple computers—as many as 200 at a time—worked simultaneously to carry out the (100 trillion) computations needed to determine how Chinook should make its moves so that it never loses.
The solutions to both tic-tac-toe and checkers are complete solutions—the "winning move" to make from any board position has been determined and has been built into the software. The game of chess is still more complex. As mentioned earlier, a computer that could play winning chess games was thought to be the pinnacle of artificial intelligence. In May 1997, international attention was focused on a historic chess match between world champion Garry Kasparov and the IBM chess-playing supercomputer known as Deep Blue. Kasparov and Deep Blue played neck and neck. Both relied on the respective strengths of their "species," Kasparov utilizing recognition and reasoning, and Deep Blue churning out its high-speed computations. In the final game, Kasparov lost the match by falling for a well-known trap. Kasparov's error, which was considered a major blunder for a player of his ability, probably reflected his weariness and the emotional strain of competing against an unexpectedly strong, utterly impassive foe. These human frailties, of course, were not shared by Deep Blue or its successor, Deep Junior, shown in Figure 15.13 playing against Kasparov in 2003.

Figure 15.13Man vs. computer

Source: Bebeto Matthews/Associated Press

Kasparov could evaluate up to three chess positions per second, or 540 in the 3 minutes allowed between moves; he selected which few positions to evaluate on the basis of his experience, study of successful strategies or tactical approaches, and intuition. Deep Blue could evaluate up to 200,000,000 chess positions per second, or 50 billion in 3 minutes, using its 512 communicating processors. But even Deep Blue could not pursue a true brute force approach of playing out to the bitter end every possible consequence of every potential move from a given configuration. Instead, its programmers provided a number of heuristics that helped "prune" the search tree to a manageable number of paths, in effect selecting what seemed to be the most promising moves.

The game of Go is an ancient Chinese board game played by two players using black-and-white stones placed on a (usually) grid. Stones represent territory controlled by a player. Once put in place, they are never moved, but can be removed if "captured" by the opposing player. The object of the game is to control a maximum amount of territory. A stone surrounded by stones of the opposite color is subject to capture, which makes it advisable to keep your stones clustered closely together. But that defeats the object of

increasing the territory you control. Although the rules of Go are simple, there is a great deal of strategy involved in successful play. Obstacles to success include the large size of the grid and the number of pieces on the grid, the fact that pieces generally remain on the board, the fact that a player has a huge number of moves at his or her disposal at any time, and the fact that a given move could be good or bad depending on the intent of the player. It is estimated there are possible Go games that can be played.
Recall that the AI experts (see the Special Interest Box "Predicted AI Milestones" earlier in this chapter) estimated that it would be 2028 before artificial intelligence would be able to win at the game of Go. However, in May 2017, only two years after these predictions were made, Google's AlphaGo software was pitted in a series of three Go matches against the world's top Go player at The Future of Go Summit held in China, the birthplace of Go. AlphaGo won the series in a clean sweep. (Before we celebrate the impressive artificial intelligence win, we should note the pretty remarkable fact that the human player, at the time of this competition, was a 19-year-old Chinese man named Ke Jie.)

AlphaGo did not play by memorizing gazillions of possible paths to victory at each step. Rather, it learned from experience as a result of playing a large number of games against itself (machine learning). After the final match, Ke Jie said "This Summit is one of the greatest matches that I've had. I believe, it's actually one of the greatest matches in history."

Google is retiring AlphaGo from the world of Go, but plans to put the team that developed AlphaGo to work on other grand challenges, such as reducing energy consumption or finding cures for certain diseases. Meanwhile, the high-level human Go players plan to study AlphaGo's innovative moves during the tournament to improve their own play.

Quiz Games. *Jeopardy!* is a popular American television quiz show. The show draws an estimated 25 million viewers per week who watch Alex Trebek, the host since 1984, throw "answers" at three contestants, who have to supply the corresponding "question." *Jeopardy!* took a startling new turn when, in 2011, an IBM supercomputer was one of the three contestants. Named Watson, after IBM's founder Thomas Watson, the computer got off to a slow start, but over three days of competition, bested two previous *Jeopardy!* champions, Ken Jennings and Brad Rutter.

In Figure 15.14, the Watson avatar is glowing proudly. But behind the scenes is the real Watson, consisting of 90 IBM Power 750 servers, each with 32 core processors and up to 256 GB of RAM (Figure 15.15). When Watson got a clue, processing algorithms analyzed the clue and then ran several different searches that could produce as many as 300–500 candidate answers. This phase used heavy-duty computing resources (note the large amount of main memory storage) and although several servers may have been working at once on the same clue, this was basically not a parallelized process. However, the next phase of "scoring" the candidate answers to decide on the best response was distributed across the cluster and done in parallel.

Figure 15.14IBM's Watson beats its human opponents on *Jeopardy*!

Source: Seth Wenig/AP

Figure 15.15 **IBM's Watson supercomputer**

Source: Courtesy of IBM

Certainly Watson had a huge database of facts at its disposal, but what the IBM researchers and their collaborators accomplished is much more than a massive search process. Web search engines conduct massive searches, but they basically do word matching and then present you with a list of potentially relevant documents or other artifacts to sift through yourself, although they do vast statistical analyses to attempt to improve relevance. Watson analyzed clues fraught with vagaries of English language—puns, humor, rhymes, riddles, plays on words—to understand their meaning.

Watson was not infallible. With a category of "U.S. cities" and the clue "Its largest airport is named for a World War II hero; its second largest for a World War II battle," Watson responded "What is Toronto?", although with a low level of confidence in its answer. * But before you feel smug, consider Watson's correct response in the category of "edible rhyme time" and the clue "A long, tiresome speech delivered by a frothy pie topping." It had to recognize that the clue consists of two parts, a long tiresome speech and a frothy pie topping. It had to find potential answers for each part, and then apply the constraint that the two answers must rhyme. Can you answer this—in under three seconds?*

Watson has evolved since its game-playing *Jeopardy!* days, now employing its natural language processing capabilities in many fields of interest (see the Special Interest Box, "'Now I Understand,' Said the Machine," in Chapter 11).

Main content

Chapter Contents

15.6 Robots and Drones

15.6.1 Robots

The term **robot** implies a device, which may or may not be humanlike in form, that has the ability to gather sensory information from its surroundings and to autonomously (without direct human instructions) perform mechanical actions of some sort in response. The term *robot* was used in a play written in 1921 by Czech author Karel Capek. The play was titled *R.U.R.*, short for *Rossum's Universal Robots*. In the play, a scientist invents robots that perform simple repetitive tasks to help people but who take over the world when their human owners try to use them to fight wars. The word *robot* comes from the Czech word *robota*, meaning slavelike labor. Robots have been part of science fiction ever since—think C-3PO and R2-D2 in the *Star Wars* movies.

Fact has not yet caught up with science fiction, but today there are a surprising number of applications of robots. Here's a partial list of the uses to which robots are put in manufacturing, science, the military, medicine, and the consumer marketplace:

- Assembling automobile parts
- Packaging food and drugs
- Placing and soldering wires in circuits
- Bomb disposal
- Exploration (the Mars Rovers)
- Underwater salvage
- Microsurgery
- Emergency search and rescue
- Fetching and packing items from warehouse storage

Robots currently perform many tasks that are too repetitive or dangerous for humans. Robots, of course, do not get bored, they can perform the same tasks with the same precision every time, and they can work in hostile environments.

We have all seen photos of robots performing manufacturing tasks; in Figure 15.16, robots are busy welding car frames. But what if the car itself is the robot? Google began developing an autonomous (self-driving) car in 2009. It used modified commercial automobiles fitted with lots of sensors and software. On top of these cars is a revolving turret called a *lidar* device, short for light detection and ranging. The lidar constructs a 3-D map of the car's environs, which the software then compares with a high-definition (at the one-inch level) map of the area in which the car is traveling.

Figure 15.16 Robots welding car frames on an assembly line

© RainerPlendl/ Shutterstock.com

In May 2014 Google revealed a new prototype autonomous car—one without a steering wheel, brake pedal, or accelerator pedal (see Figure 15.17; it was intentionally made to be cute so as not to frighten people). As of March 2016, the Google test cars had logged 1,500,000 autonomous driving miles with a remarkable safety record, but always with a trained human ready to take control if needed. In December 2016, Google spun off its autonomous car program as a new business called Waymo. Waymo then launched a program in Phoenix, Arizona, for public trials of their autonomous vehicles, that is, selected ordinary citizens are invited to use an autonomous vehicle for everyday uses such as driving to work, school, or the grocery store, and report on their experiences.

Figure 15.17 Google's prototype autonomous car

© AP Photo/Google/REX

Now there are a number of companies, including traditional automakers, in various stages of designing and testing autonomous vehicles. By May 2016, sixteen states and Washington D.C. had passed legislation relating to autonomous vehicles, some creating study groups, some encouraging development and testing, and some permitting autonomous vehicles to be driven on public roads. In 2016, Florida became the first state to authorize the use of autonomous vehicles on public roads with no requirement for a human driver in the car. In May 2017, the governor of New York announced a oneyear trial of autonomous cars on public roads in the state, which requires a human with a valid driver's license in the driver's seat as a precaution; someday we may see fully autonomous vehicles navigating the challenging traffic of Manhattan.

Wait—Where Am I?

Those who call on Uber for a ride in Pittsburgh might now find themselves passengers in an autonomous vehicle. Uber began testing its own autonomous vehicles in Pittsburgh in September 2016. These vehicles do have a steering wheel with a driver behind the wheel to take control of the car in an emergency, plus an engineer in the front seat to monitor the car's actions.

Pittsburgh was chosen as a test site because it is an old city with a complicated pattern of roads, lots of bridges, lots of potholes, lots of traffic, and its share of bad winter weather, so it provides quite a challenge for an autonomous vehicle.

Did we mention bridges? Uber cars tend to have difficulty with bridges, as in "disengaging"—dropping out of autonomous mode—in the middle of a long bridge. Why? Because at that point the landmarks by which the autonomous car judges its position relative to its internal map are too far away to see. Another unexpected difficulty occurred because some of the internal maps were compiled by driving around Pittsburg in the winter when there were no leaves on the trees, and when the autonomous car drove the same roads in spring when the trees had grown lots of leaves, the landmarks (trees) the car sensed no longer matched the internal maps.

More and more, however, robots are being developed to interact with humans in a less "robotic" and more "humanlike" way to perform tasks for disabled individuals, to watch over small children, and to entertain and provide companionship. Japan has an interest in developing "humanoid" robots to help care for its aging population.

One of the more well-known humanoid robots is ASIMO (Advanced Step in Innovative Mobility), built by Honda Motor Company, a Japanese corporation. As the name suggests, much of the focus of the design of this robot over earlier models was refinement of the robot's motion capabilities, extending the range of its arm movement and improving the smoothness and stability of its two-legged walking, including walking on uneven ground and navigating stairs. This jointed robot is designed to measure the forces acting on it at each step. If these forces get out of balance, threatening a fall, adjustments are made in the placement of the foot and the position of the torso to regain balance. One only has to watch a toddler learning to walk to see the challenges this represents. ASIMO was created in 2000, but its capabilities have developed over time, with better balance, smoother motions, and increased hand dexterity, including the ability to use both American and Japanese sign language. ASIMO can climb up and down stairs, hop on one foot forwards, backwards, and sideways, and run remarkably smoothly, if somewhat noisily, at a speed of 9 km/hour (almost 5.6 mph) (see Figure 15.18). ASIMO can open and close doors while passing through a doorway, walk down a hallway without bumping into people, guide office guests to a meeting room and serve refreshments on a tray, and unscrew a lid from a jar and pour its liquid contents into a cup without spilling. Moreover, ASIMO, based on input from its multiple sensors, can evaluate its environment and autonomously adapt its actions accordingly; it can also recognize faces and voices. ASIMO is an international celebrity, with public appearances worldwide. It has conducted the Detroit Symphony Orchestra, received a royal welcome in Dubai, and served (in 2017) as Grand Marshall of the Honda Indy Grand Prix auto race in Alabama where it taught one of the drivers some dance moves. ASIMO now is part of the Honda-sponsored Autopia attraction at Disneyland in California.

Figure 15.18 Honda's ASIMO running

AFP/Getty Images

Research is ongoing in the field of robotics. Robotics involves the aspects of artificial intelligence we discussed earlier, namely, recognition tasks and reasoning tasks. Through some sort of elementary vision, auditory, or tactile system, the robot must not only gather sensory information but also filter out the possibly vast amount of data its surroundings might present to it to "recognize." That is, it must be able to make sense of the important features and discard the unimportant. Then the robot must make decisions—reason about—the information it has recognized to be able to take some action. There is also the additional challenge of the mechanics and electronics needed to make the robot respond physically.

Two strategies characterize robotics research. The **deliberative strategy** says that the robot must have an internal representation of its environment and that its actions in response to some stimuli are programmed into the robot based on this model of the environment. This strategy seems to reflect what we as humans think of as high-level cognitive reasoning—we have a mental model of our environment, we reflect on a stimulus from that environment, and make a reasoned decision about the next course of action. (This is a generalization of the expert system idea discussed earlier.)

The **reactive strategy** uses heuristic algorithms to allow the robot to respond directly to stimuli from its environment without filtering through some line of reasoning based on its internal understanding of that environment. This stimulus-response approach seems to reflect human subconscious behavior–holding out our hands to protect ourselves during a fall, for example, or drawing back from a hot flame. Proponents of the deliberative strategy argue that a robot cannot react meaningfully without processing the stimulus and planning a reaction based on its internal representation of the environment. Proponents of the reactive strategy say that such a requirement is too restrictive and does not allow the robot to respond freely to any or all new stimuli it might encounter. Note that we as humans use both our conscious thought processes and our subconscious reactions in our everyday life, so a combination of these strategies may be the most successful approach.

Change font size Main content

Chapter Contents

15.6.2 Drones

A **drone** (more properly called a *UAV* for *U*nmanned *A*erial *V*ehicle) is an aircraft that is either under autonomous control via a computer system on board or is controlled by a human controller at a remote site. In other words, no pilot is on board.

The use of a UAV in the most general sense actually dates back much farther than we would expect, to the American Civil War, in which both Union and Confederate troops launched bomb-filled balloons. But the idea of a drone as we think of it today began in the 1960s during the Vietnam War, in which drones were used for reconnaissance missions.

Drones are still used primarily for military (or law enforcement) purposes today, but their use is expanding into other areas as well. A partial list of what drones are starting to do includes:

- Agricultural-related aerial photography. Drones can take photographs of a farm field row-by-row; each photo is tagged with GPS information. Using specialized software, the resulting data can be used to create fertilizer distribution plans, predict crop yield, and other useful information.
- Delivery of medical supplies or blood to remote areas, such as some Native American reservations or communities where natural disasters have knocked out access roads. Such a program has been underway in Rwanda since July 2016, and plans are underway to test this process in Maryland, Nevada, and Washington, once FAA approval is obtained.
- Surveys of livestock or wildlife. In February 2017 (summer in Antarctica), wildlife biologists were spying on penguins and humpback whales. Using drones is safer and cheaper than using helicopters or small planes. Drones could even be used to spot wildlife poachers (in, say, Africa, not Antarctica).
- Real-time reports on fires, floods, chemical spills, receding glaciers or volcanic activity. The Fire and Rescue Service in the town of Chorley, Lancashire, U.K., is using a drone for monitoring fires, floods, collapsed structures, and even searching for missing persons. The drone can be gotten up in the air quickly and can fly in high winds and poor weather.
- Pipeline inspections. Drones use high-resolution digital, infrared, and thermal imaging to detect and document possible hazard concerns in pipelines, such as methane or ethane gas leaks, pipe erosion, exposed pipes, vegetation overgrowth, and missing or damaged signs or markers. Again, the drone is a faster, safer, and less-expensive solution than helicopters or small aircraft.
- Urban traffic reports. Drones can be used to monitor urban traffic, report rush-hour traffic jams, and so forth. However, Dubai has another solution in mind for avoiding rush-hour traffic; beginning in July 2017, Dubai is scheduled to use one-passenger drones that can fly short, pre-programmed routes. It's sort of Uber-of-the-air, but with no drivers, although flights will be monitored by a ground crew.

Drones are usually not as big as a regular airplane or helicopter. In fact, small drones working together (swarm intelligence again) can get into places where a large vehicle cannot and obtain a more complete picture of the environment they are exploring. For example, human engineers currently inspect bridges for damage, a slow, dangerous, and far-from-foolproof method. Researchers at Tufts University are developing a system that will allow drones to improve the infrastructure problem of decaying bridges. The plan is to attach wireless sensors to bridge joints; these sensors can monitor bridge vibrations and detect changes that may signal structural damage. A group of drones can not only visually survey the bridge but also collect data from the sensors and send it to a central site for analysis, allowing decisions to be made on a priority list for bridge repair. These drones would communicate with each other to check one another's position and flight paths, and to autonomously regroup to collect more data on a suspicious location.

There is also interest in tiny drones. For example, swarms of insect-sized drones could be released in a disaster area to assess damage and search for survivors. Because of their size, they could penetrate hard-to-access areas of debris and transmit signals to (human or robotic) rescue workers. Insect–like drones are not just “small airplanes”–the flight dynamics of insects are much more complex because of an insect’s ability to hover, fly in any direction, land on walls, and so forth.

A Japanese lab is researching the use of small drones to act as artificial “bees” (after all, “drone” is the term for a worker bee). Bees are crucial to the pollination of many important crops and flowering plants, but the bee population is in serious decline. The artificial bee is about the size of a hummingbird, but has a fuzzy strip made from paintbrush hairs that are coated with a gel sticky enough to trap pollen if the bee brushes up against a flower, yet not so sticky that the pollen isn’t released when visiting another flower. Currently the artificial bee is “flown” by a human controller, but future plans include arming it with a visual recognition system so it can find flowers on its own.

The possible use for drones that has most captured public attention is package delivery. Amazon.com’s plan is to use drones to deliver a package weighing up to 5 pounds to the customer within a 30-minute window. You would place your order online, step outside, and wait for a little buzzing drone to gently drop your order on your front porch. In fact, Amazon made its first package drop to a customer in England in December 2016.

Companies, including Amazon, that want to build or use drones for commercial purposes are lobbying for faster approvals by the Federal Aviation Administration. The FAA is proceeding with caution, and the rules adopted in 2016 require that UAVs can weigh no more than 55 pounds, must fly below 400 feet above ground or the top of a building, must fly below 100 miles per hour, cannot fly at night or over crowds of people, must have a human operator, and cannot fly beyond the operator’s visual line of sight. But aside from regulations, there are other issues to consider:

- How will the increasing number of drones work safely in crowded skies, especially if they are all flying below 400 feet? (X is a futuristic lab that is a subsidiary of Alphabet, the parent company of Google. X is working on software to manage “air traffic control” for drones. The system views current flight paths, anticipates when collisions might occur, and updates the drones’ flight paths to avoid them.)
- Are they safe or will they crash into someone’s home or car?
- What about privacy–what prevents a little drone from hovering outside your window and collecting data?

15.7 Conclusion

In this chapter, we have touched on three basic elements of artificial intelligence: knowledge representation, recognition problems, and reasoning problems. We’ve discussed common approaches to building artificial intelligence systems: connectionist architectures (neural networks), genetic or evolutionary approaches (swarm intelligence), and symbolic manipulation (expert systems). We have followed the progress of artificial intelligence in game playing and have also outlined some of the strategies and challenges in robotics design and the use of drones. Yet we have mentioned only a few of the many application areas of AI, a field that is finally beginning to realize its potential. Today we can use speech-recognition systems to control our phones, appliances, and computers by talking to them, exploit face-recognition software to assist the police with their crime-solving responsibilities, use a tablet PC with handwriting-recognition software, and ask a webpage to translate text from one language to another. In 10–20 years, artificial intelligence will be used in ways we have not even imagined today.