CHAPTER 1

Computer Functioning

1. What Computers Do

The following pages describe the general mode of functioning of digital computers. The material focuses on how machines can represent information and how computations are achieved.

1.1. The Turing Machine

The idea of a "thinking machine" has a long history. Charles Babbage (1792–1871) designed a machine, called the Analytic Engine, which was in fact a computer. It was a hugely complicated affair of ratchets, levers, and bolts. Finished, it would have been train engine size. It was never built.

In 1946, the mathematician Alan Turing devised the first simple computer, known reasonably as the "Turing machine." It was an "abstract, universal calculating machine" that could process anything amenable to a computational approach. It consisted of a tape divided into separate cells, such that a single character (1 or 0) could be written in each of the cells. The machine would read a character in a cell, write a character, or move one cell in either direction along the tape. At any given moment, the machine would be in a specific state. If it was in State 1, and it read a 1, it would perform certain activities. If it was in State 1 and read a 0, it would perform other activities. Its behavior was thus determined by (1) its state and (2) what it read when in that state. The number of states that the machine could be in were finite. They depended on its construction. It was thus possible to list all the states that a given machine could enter.

A list of the machine states of a given computer, along with the activities relevant to those states, is known as its *machine table*. This includes each possible machine state, each character that the device might scan, what character it will then write on the tape, and what direction it will then move—and finally, the new state into which it will shift. A machine table is shown in Figure W1.1. As represented in the figure, if the machine is in State 1 and it scans a 0, it will print 1, move to the left, and go into State 4. If it is machine State 1 and it scans a 1, on the other hand, it will print 0, move to the right, and go into State 2. The process will then continue.

The Turing machine could have performed any computation, but it would have been too slow to be practical. But electronic computers, performing these same functions at great speed, were in operation by the late 1940s.

MACHINE STATE	IF SCAN:	PRINT:	MOVE	GO INTO
1	1	0	R	2
1	0	1	L	4
2	1	1	R	10
2	0	0	L	1
3	1	0	R	2
3	0	1	R	15
etc.				

Figure W1.1 A Machine Table Showing the States of the Machine and the Activities That It Will Perform on Reading Either a 0 or a 1 in Each State

1.2. Multiple Architectures

A machine that can compute is a machine that can embody a formal process. Certain set ingredients are required. In traditional computing, the machine embodies symbols. It must be able to read the symbols, and there must be a place to hold them. This is the computer's memory. There must also be a locus within the memory at which the program reads each symbol (its working memory).

The program will be able to perform manipulations, or actions. It needs some kind of agent for each possible action. And there must be an executive component that directs what will happen at each stage of the process. The executive determines what symbol will be acted on, by what agent. When an action occurs, the symbols change. The new symbols must of course be held in memory. And at the end, the program will have some way of stopping when the solution is reached.

There is not one way in which these functions can be achieved (as, for instance, in the Turing machine described above), but many, and therefore many different possible machine architectures.

Work across the early 1950s led to the development of a more flexible approach to computer functioning. John von Neumann (1903–1957) played the central role, and programs of the newer kind are named after him. Most modern digital approaches reflect von Neumann architecture.

It must be possible for a program to access (enter) its memory. Access can be achieved on either a relative or an absolute basis. In relative access, a new symbol is placed in memory on the basis of its location relative to the currently active symbol. For instance, in a Turing machine, once a given symbol has been processed, the tape will be moved one step to the right or one step to the left. In von Neumann machines, access can be either relative or absolute. Absolute access involves specifying the location of a symbol by name or number. Each memory location thus has a unique "tag," which the program can read.

1.3. How Computers Embody Information

When I first learned that computers can express or embody information, such as the information contained in the word "dog," I was curious as to how this could possibly be achieved. I could not imagine any way that meaning might be represented (in human or machine), and the thought that knowing how a computer performs this function might be quite illuminating. I had taken it for granted that the medium would not be symbols. This was of course a serious—and odd—mistake, since the medium consists, precisely, of symbols.

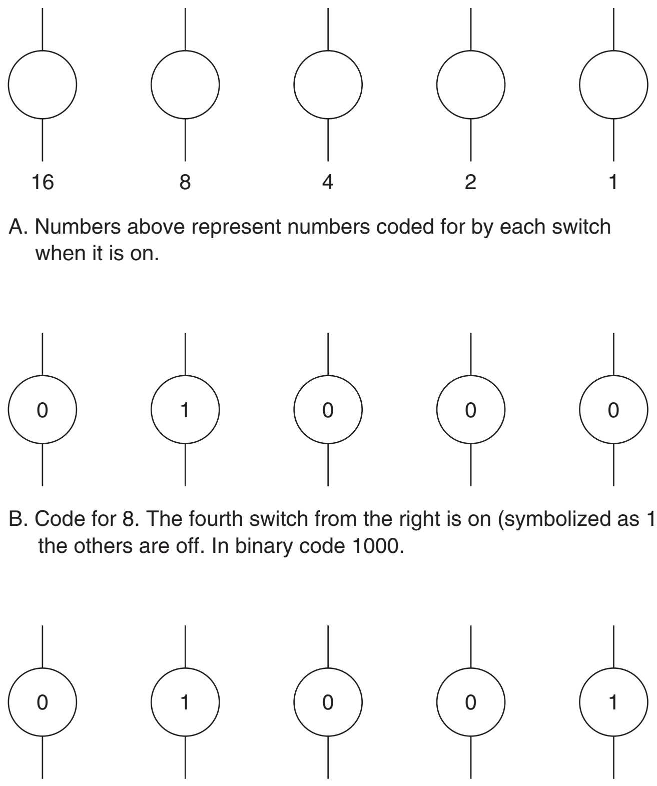
As described in Chapter 1, a symbol is a token that stands for or represents something else, although it bears no internal relationship to the thing represented. In other words, it does not sound, look, or act like the thing represented. Most cognitive simulations today are performed on digital computers. A digital computer possesses switches, arranged in rows. A given switch can be either on or off at any specific time. The switches are accepted as representing numbers. If a switch is on, the number that it represents is being expressed within the computer. The code used is binary, and the numbering is achieved on a right-to-left basis. The switch to the far right symbolizes two to the power of zero (2^0), the next switch to the left symbolizes two to the power of one (2^1), the next to the right again symbolizes two to the power of two (2^2), and so on. To represent numbers that are not direct powers of two, multiple switches are activated together, and the numbers that they symbolize are added. For instance, to represent 6, the switch representing two, which is base two to the power of one (2^1) (located second from the right) and the switch representing four, which is base two to the power of two (2^2) (third from the right) are both turned on, and their numbers are added together.

The numbers can then be used to symbolize other types of information. For instance, in a given program, the number 8 could be used to symbolize the concept HOUSE. In some languages, numbers also designate letters, so that entire words and sentences can be expressed in symbolic letter form. Symbols can also be placed together in longer strings, to represent entire propositions, with both the conceptual representations and the relational grammar being expressed.

Figure W1.2 shows a line of switches used to represent numbers in a digital computer.

2. A Simple Program

The following material describes a program that has been designed to identify whether any statement in a body of text contradicts any other statement—and,



A. Numbers above represent numbers coded for by each switch when it is on.

B. Code for 8. The fourth switch from the right is on (symbolized as 1); the others are off. In binary code 1000.

C. Code for 9. Switches representing 8 and 1 are on. 8 + 1 = 9.

Figure W1.2 Switches Representing Numbers in Binary Code

NOTE: A shows the number corresponding to each switch. The switch to the right represents 2 to the power of 0 (2^0) (which by convention is 1). The next switch to the left represents 2 to the power of 1 (2^1), the next again to the left represents 2 to the power of 2 (2^2 or 2×2), and so on. Line B shows the code for the number 8, which is 2 to the power of 3 (2^3 or 2×2×2). Line C represents the number 9 (8+1).

if so, to eliminate one of the contradictories. The computer might receive the following three input propositions:

a. Calgonite cleans dishes

b. Brooms sweep floors

c. Calgonite smears dishes

The program would accept (a) and (b) as involving no contradiction, but it would reject statement (c) on the basis of its not being compatible with statement (a).

The relevant processing would occur as follows. Each element in the three propositions is represented in the machine by a number symbol. In the present example, three tables are sufficient to provide all the data needed by the program to reject contradictory statements.

Table 1 (in Figure W1.3) provides a specification of the symbol numbers corresponding to the input words, some additional items needed by the program, and a specification of the grammatical form (part of speech) of each element.

The symbols for the relevant words are coded as follows: Calgonite = 1, cleans = 2, dishes = 3, brooms = 4, sweep = 5, floors = 6, buffs = 7, scrubs = 8, polishes = 9, smears = 10.

Table 2 (in Figure W1.3) specifies the three propositions that are coded into the machine. Statement 1 is specified in Row 1, Statement 2 in Row 2, and so on. This table provides information concerning the part of speech of each of the elements contained in the statements.

Table 3 (in Figure W1.3) specifies possible antonyms and synonyms for the elements in Table 1. In this example, only a limited number of items have been included. A program would typically include a broader base of this class of information. Figure W1.3 shows the operation of the program.

A critical aspect of computer functioning involves the following relation: if x, then y. If the machine identifies state x, and only if it identifies that state, it will perform a certain operation. For instance, if it discovers a given type of information in Table 1, it may move to another table. These events occur repeatedly in the simple program described below.

In this example, the machine would be directed to identify whether any two of the input statements had the same subject and object. In computer terms, it would be directed to locate Table 2 and compare the numbers in both column 3 and column 6 for all three rows (rows 1 and 2, 2 and 3, and 1 and 3) In cases where the numbers in columns 3 and 6 were identical, further processing events would be specified. In fact, rows 1 and 3 both involve a 1 in column 3, and a 3 in column 6.

Translating these events into verbal form, the machine has identified the fact that the subjects and objects of Statements 1 and 3 are the same. Under these conditions, it is directed to identify the number corresponding to the verb in Statement 1 and the number corresponding to the verb in Statement 3. These are Numbers 2 and 10, respectively. The number 10 is stored for future reference. In computer terms, the machine has been directed to read the number present in row 1, column 5, and in row 3, column 5, and to store the second number. It is then directed to locate the first of these numbers (#2) in Table 1. The shift to Table 1 is used as a means of finding additional data in Table 3.

The machine locates the first critical verb (Element 2) in Table 1 and reads the corresponding entry in column 2. This entry specifies the location of the 2-element in List 3. It then moves to that location in Table 3. The relevant location is row 1. The program next directs the machine to read row 1, column 3, in Table 3. The final specification is that this number (10) be compared with the stored number. If the two are identical, one of the original input statements must be rejected. It is incompatible with the other. (The stored number in this example was in fact 10.)

In verbal terms, the following has occurred. Once the machine established that the subjects and objects of Statements 1 and 3 were the same, it identified the verb

NUMBER SYMBOL	VERBAL DESCRIPTION
1	CALGONITE
2	CLEAN
3	DISHES
4	BROOM
5	SWEEP
6	FLOOR
7	BUFF
8	SCRUB
9	POLISH
10	SMEAR

TABLE 1

ROW	ELEMENT	ROW IN TABLE 3	PART OF SPEECH
1	1	4	1
2	2	1	2
3	3	5	1
4	4	6	1
5	5	2	2
6	6	7	1
7	7	8	2
8	8	9	2
9	9	3	2
10	10	10	2

* 1 = noun
* 2 = verb

TABLE 2 (Propositions)

ROW	PROP #	SUBJECT	TENSE	VERB	OBJECT
1	1	1	0	2	3
2	2	4	0	5	6
3	3	1	0	10	3

TABLE 3 (Antonyms/Synonyms)

ROW	ROW IN TABLE 1	ANTONYMS	SYNONYMS
1	2	10	8
2	5	–	2
3	9	10	7

Figure W1.3 Operation of the Program

NOTE: Table 1 shows the grammatical properties of the words. Table 2 shows the propositions. Table 3 shows antonyms and synonyms for symbols 2, 5, and 9 (clean, sweep, and polish). The antonym for 2 ("clean") is 10 ("smear"). This indicates that statements 1 and 3 in Table 2 are contradictory.

of Statement 1 in Table 3. (It used Table 2 to find this location.) Having located the required verb, it moved to the column specifying antonyms for that verb. Since an entry here corresponded to the verb in Statement 3, this established the presence of a contradiction.

Most programs require the manipulation of considerably larger quantities of information. However, the underlying events are the same.

3. Production Systems

Production systems are now widely used in programs that simulate human memory. They were first introduced in the 1940s and later adapted to model human cognition (Newell & Simon, 1972).

A processing system can be described as being, at any given time, in a certain state. That is, certain symbols are present, and they relate to one another in a specific fashion. As the system begins to function, this state (s) will change. This means that a process will begin at state s and move through various transitions to si, sii, siii, and so on. All the possible states of the system are called the *state space*. The series of states that the system moves through is called its *trajectory*, or the trajectory of s.

A discrete dynamical system is a system that moves in this fashion through specific, discrete states. A production system is in fact a discrete dynamical system. It possesses the following three components:

1. A global database	This is the set of data in the system.
2. A set of production rules	These are rules that move the system from state to state.
3. A control structure	This is the executive component that determines which production rules will be used.

A production system also operates on an IF/THEN basis. The program will specify certain target symbols. It will then determine whether these symbols (which indicate a state of affairs) are present in working memory. IF a target set of symbols is present, THEN the corresponding production rule will fire.

The control structure determines which of the production rules will fire first. In a production system, it is usual for many active processors, described as demons, to be watching for different target symbols. That is, each demon is specialized to a particular symbol or group of symbols. If a demon's target appears, it will fire a (specific) production. The system is thus not restricted to a rigid sequence of events (as described in the Turing machine function above). Instead, it can fire productions somewhat flexibly, depending on the state of the system as a whole at any given moment (which in turn depends on the symbols in the database).

When more than one IF condition is met at the same time, the control structure determines which rule will take precedence. The identification of target symbols is achieved on a matching basis. A match will occur when the symbols in working memory meet the IF specifications for a given production.

Production systems are often used in programs whose information takes propositional form. Thus, an IF condition could be "IF the object is large, brown, and of irregular shape," THEN represent, "The growth is pathological."

As the system moves through its various states, it performs the required computational function.

Production systems are now widely used to simulate human memory. They also provide the basis of expert systems, programs that can perform "expert" tasks, such as medical diagnosis or chess playing.

4. Connectionism

When researchers began the first attempts to simulate human cognition on computers, the approach involved the goal of duplicating the activities of neurons. They wished to identify the specific actions performed by the brain and embody them in a computer. But the gap between the physical nature of the brain and psychological representation proved too large. No one succeeded in creating a program that could simulate human cognition by means of processing the duplicated actions performed by neurons or neural sets.

The approach that met with success was instead a strictly informational approach. It reflected the type of model described above. The researchers established symbols and identified the computational events that would be needed to move a program through multiple steps that would lead to the correct output. Computational output was achieved, but without reference to how the brain might achieve the same output. As suggested above, it was also widely believed within the artificial intelligence (AI) community that the specific way in which computation is achieved is unimportant (Pylyshyn, 1984).

The view that the medium providing a cognitive function is not critical, however, has generally been opposed by researchers in the fields of neurophysiology and neuroscience, and by some researchers in AI itself. First, it is agreed by most cognitive scientists that the brain is the physical medium of cognitive functioning. At the least, the two separate disciplines of psychology and neuroscience are engaged in exploring a related topic and should be able to work more efficiently on the basis of shared ideas rather than as two mutually isolated programs. In addition, the possibility still exists that cognitive events may one day be explained in full on the basis of an understanding of neural functions (Churchland, 1986).

But perhaps more compelling arguments than these have emerged within the AI community itself. Some researchers believe that the traditional, symbolic AI methods are limited, and they hope to find a more powerful vehicle for the modeling of our cognition. It must be allowed that the structure of a system necessarily defines the kind of functions that system can perform. Given this straightforward fact, the argument can be made that an organization closer to that of the brain may provide capacities that are not available in vehicles that operate in a clearly unbrainlike fashion.

As described in Chapter 1, considerations of the kind outlined above led in the end to a new way of modeling cognition, variously described as *connectionist* *models*, *artificial neural networks* (ANNs), and *parallel distributed processing* (PDP). Connectionism is based, at a very crude level (as its proponents admit), on the mode of functioning of neurons in the brain.

Connectionist models are written as computer programs. They implement an architecture that consists of rows of units. The units are taken to be the rough equivalent of neurons. Most models involve three rows of units. The first is the input layer. It receives input from outside the system. The second row is identified as the middle or "hidden" layer. Then there is the final, or output, layer, which expresses the result of the system's processing.

Each row of units is connected to the next by means of links or associations. Any given unit may have many, or no, links with the next row. Each link carries a weight. The weight determines how much activation will be passed to the receiving unit (the one "deeper" in the system). And each unit has a threshold. If incoming activation reaches or exceeds that threshold, the unit will fire, or be activated, passing activation along any links leading from it to other units.

The threshold level of a unit determines only whether it will be activated. It does not determine the amount of activation passed on to the next layer. Only the weights of the links determine this factor.

Thus, if a unit with a threshold of 10 is activated and has a given link leading to the next layer with a weight of +1 on the link, the amount of activation carried by that link to the unit in the next layer will be only +1.

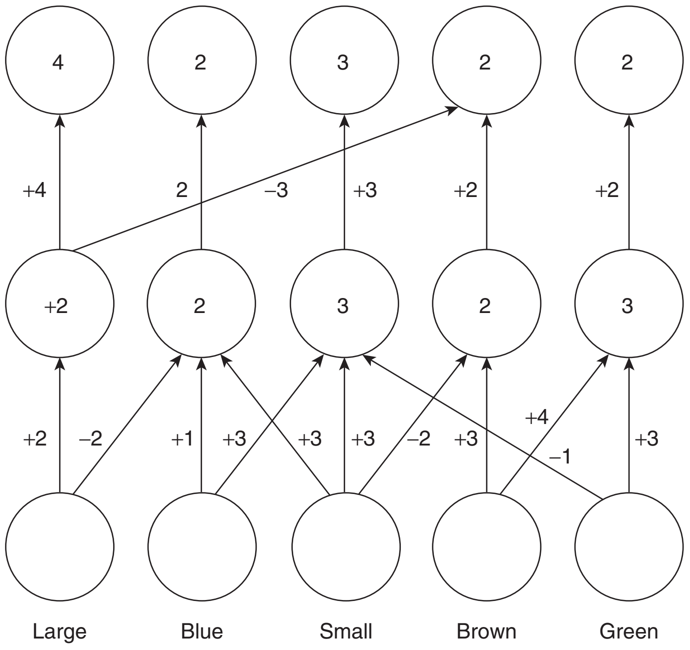
There are both excitatory and inhibitory connections in the system. Suppose a unit in the middle layer is activated and sends the activation along a link weighted +5 to the output layer. If the unit in the output layer has a threshold of 4, it will fire. Equally, if this output unit receives an input with a weight of +1 from a unit in the middle layer, another input of +1 (from another unit), and yet another input of +2, it will fire.

But if it receives an input of +4 from a unit in the middle layer and also an input of –1 from another unit in the middle layer, it will not fire.

Input to the system is expressed in terms of vectors. Thus, a vector of 1,0,0,1,1,0,1,1 would indicate that there was input to the first left-hand unit of the input layer, no input to the second unit from the left, none to the third, but input to the fourth, and so on. Input may involve the reception of a particular type of feature or property. Output is also represented as a vector or 1s and 0s, this time symbolizing the information finally generated by the program.

Connectionist nets can perform any type of computation, but they have been particularly successful in the domain of perception and pattern recognition. For instance, the input to a net might involve the features of certain birds (one might be large and blue, with a long beak and white breast, another might be small and short billed, with other distinguishing features). A net could be programmed to identify the animals, such that the description of the first bird would lead to the coded output for "blue jay," the second for "sparrow," and so on.

A small part of a connectionist net is shown in Figure W1.4. The net has been simplified; all the connections that would normally be present even in this part of the net have not been included.



FEATURES	**OUTPUT**
Large & Brown	10001 = Owl
Small & Brown	01101 = Sparrow
Large & Blue	10100 = Blue jay
Small & Green	01001 = Hummingbird

Figure W1.4. A Simplified Connectionist Net

NOTE: This net identifies birds when a description of their features is input. For any unit, the likelihood of its being activated depends on the sum of the numbers on the arrows leading to that unit. If that sum is larger than the unit's threshold, as shown inside the unit circles, the unit will fire. The amount of activation then passed on to the next unit depends on the weight on the arrow between them. When "large and brown" (as vector 1,0,0,1,0 is entered, the output will be vector 1,0,0,0,1, which symbolizes "owl."

The critical factor in determining the output of a network involves the weights on the connections. In some cases it is possible to identify what weights are needed to produce a desired output, simply by looking at the situation (as in the example shown above). But most networks are too complex for the right weights to be obvious from the beginning. The correct weighting may be achieved through step-by-step adjustments, moving generally in the desired direction. This approach is known as "training" a network. The most common training procedure is called *backward error propagation* or *backprop*. Suppose you want a given output for a four-unit output layer. You want the first two units to fire and the second two units not to fire (symbolized as vector 1100). This pattern represents some information content. You begin with small random weights on the connections. Input is given to the system, and the actual output, that is, which output units are in fact activated, is compared to the desired output. Perhaps in this run all four output units fire. The first and second have responded as desired, but the third and fourth have not.

You will then slightly lower the excitatory weights on the links leading to output units 3 and 4, and slightly increase the inhibitory weights. Input is then given again to the system. If the output is still not exactly as desired, the weights are adjusted a second (or third or *n*th time), until the correct outcome is achieved.

Training a system must be handled in this small step by small step fashion because each connection is normally involved across many different input-output associations. It is not a question of finding a set of correct weights that will achieve one specific output, as in the example given here. The system must provide many correct outputs, for many diverse inputs. That is, the same connections and weights are used to provide different information as output, under different input conditions.

There are other approaches to the design of a connectionist network, some of which involve less direct control by the designer. For instance, a network may be allowed to function across various trials, with weights being changed a little, randomly, on each trial. The weights used in the most successful run will then be adopted. And some networks have feedback devices built into them, so that they shape their own architecture (Franklin, 1995, chap. 6.) It is thus an interesting feature of connectionist nets that they can learn.

5. A Comparison of AI and Connectionist Models

A new understanding of cognition has developed over the past 50 years, inspired by the nature of computers—or more exactly, computational functions. It is now widely believed that the brain involves the processing of information. What this means is some form of manipulation of information content, associated with changing states of the brain. Perhaps the most important new understanding to emerge from the AI tradition involves the fact that the relevant functions are extremely intricate. The informational work that goes into the most trivial psychological act—say, looking at and perceiving a lamp—is almost unimaginably complex.

Information processing as understood within the symbolic AI tradition involves the claim that such processing is based on (1) symbols and (2) operations performed on those symbols, under the control of a program. Connectionism is said to challenge this view.

The first difference is that a neural net has no program. The units have associations among them. The pattern of processing activity depends solely on these associations or links. If A is activated and is linked to B on the basis of a weight that exceeds B's threshold, B will "fire." Thus, all processing is built directly into the architecture of the net itself.

The second issue that can distinguish symbolic AI and neural net programs involves the question of representation. A unit in a neural net can stand for a concept, just as a series of electronic switches can embody a concept in a traditional AI program—in which case that unit could be viewed as a symbol representing the concept. However, neural nets are also capable of *distributed representation*. In the case of distributed representation, a concept can be coded as a pattern of activity across many units within the net. A given unit can also participate in the expression of other concepts. Thus, a unique pattern of activity involving units here symbolized as units 4, 10, 16, 11, 35, and 36 might, when playing through in a certain order, embody the concept TREE. The units 2, 4, 7, 10, 20, 11, and 24, playing through in a certain order might embody the concept MOUNTAIN. Note that units 2, 4, and 11 participate in both these representations.

Sometimes the units code for the features of a concept. Thus, different concepts might share in some features, while being nonidentical across other feature sets. In a program of this kind, similar concepts would show overlaps among the activities of the units in the net. But in other cases there are units involved in representation that do not correspond even to known or identifiable *features*. They are not smaller symbols, participating in a larger representation; they do not stand for anything that we could name. These units are said to express *microfeature*s (Rumelhart & McClelland, 1986) or *subsymbols* (Smolensky, 1988). It is in this sense that neural nets are sometimes described as "subsymbolic."

Most connectionists believe that representations are nonetheless present in their nets. There is some state or ongoing states of the net that provide a true representational function. Some individuals, however, take the more radical stand that representation, as we normally think of it (the representation of meanings) is not present in either neural nets or in the human brain—that there is no extended state that *is* the representation *dog* or *rose* or *table*, taken as a whole (Smolensky, 1988).

A neural net performs computations. It could in theory do anything that a symbolic AI program could do. (Connectionist programs cannot achieve this at present in the real world, but in theory, they could.) Each approach has particular strengths. Symbolic AI programs excel at formal, high-level cognitive functions such as mathematics and reasoning, and in representing higher-order structures. Neural nets are particularly good at simulating pattern-matching functions. They are therefore good with models relating to perception.

Neural nets show several properties that parallel those of a human brain. (Symbolic AI programs never show such properties unless they are deliberately programmed to do so; their architecture provides no natural correspondence.) For instance, nets show *graceful degradation*. If part of the net is slightly damaged, or malfunctioning, the net may continue to process information and get the output partially right. With symbolic AI, in contrast, if even a small part of the program is wrong, the entire enterprise tends to crash. Such programs are described as "brittle." For the same reason that they function when partially malfunctioning, neural nets can also typically handle incomplete or distorted input.

Perhaps even more striking, the nets spontaneously generalize. That is, they generalize even when no attempt has been made to build this tendency into the program. Suppose a net has learned of certain properties present in both the gorilla and the chimpanzee and further learned that these creatures are apes. If the system then encounters a description of an orangutan, implying features that are similar to the chimp's and the gorilla's (although not identical to either), the net will probably conclude that the orangutan is an ape.

The argument has been made that some properties of connectionist nets can provide a natural explanation of certain salient properties of human memory (McClelland & Rumelhart, 1986, 1988; McCulloch & Pitts, 1943). The nets show a pattern, for instance, similar to the interference in recall that occurs when we attempt to remember sets of information that are similar to one another. Roughly, if A and B are similar, we may misrecall B in place of A. This outcome would occur in a neural net due to feature overlap. A and B would share many features in common, and a heavier pattern of activation developed by the B pattern might exceed the pattern of activation developed by the A pattern—a little as if the A activation was "pulled into" the B.

In addition to these positive qualities, neural nets also show some limitations. In general it has not been possible to include more than one middle layer (to enable more complex processing) because the patterns of activation get lost in the net.