4.0 Chapter Overview

The previous chapter introduced the elements of classical cognitive science, the school of thought that dominated cognitive science when it arose in the 1950s and which still dominates the discipline today. However, as cognitive science has matured, some researchers have questioned the classical approach. The reason for this is that in the 1950s, the only plausible definition of information processing was that provided by a relatively new invention, the electronic digital computer. Since the 1950s, alternative notions of information processing have arisen, and these new notions have formed the basis for alternative approaches to cognition.

The purpose of the current chapter is to present the core elements of one of these alternatives, connectionist cognitive science. The chapter begins with several sections (4.1 through 4.4) in which are described the core properties of connectionism and of the artificial neural networks that connectionists use to model cognitive phenomena. These elements are presented as a reaction against the foundational assumptions of classical cognitive science. Many of these elements are inspired by issues related to the implementational level of investigation. That is, connectionists aim to develop biologically plausible or neuronally inspired models of information processing.

The chapter then proceeds with an examination of connectionism at the remaining three levels of investigation. The computational level of analysis is the focus of
Sections 4.5 through 4.7. These sections investigate the kinds of tasks that artificial neural networks can accomplish and relate them to those that can be accomplished by the devices that have inspired the classical approach. The general theme of these sections is that artificial neural networks belong to the class of universal machines.

Sections 4.8 through 4.13 focus on the algorithmic level of investigation of connectionist theories. Modern artificial neural networks employ several layers of processing units that create interesting representations which are used to mediate input-output relationships. At the algorithmic level, one must explore the internal structure of these representations in an attempt to inform cognitive theory. These sections illustrate a number of different techniques for this investigation.

Architectural issues are the topics of Sections 4.14 through 4.17. In particular, these sections show that researchers must seek the simplest possible networks for solving tasks of interest, and they point out that some interesting cognitive phenomena can be captured by extremely simple networks.

The chapter ends with an examination of the properties of connectionist cognitive science, contrasting the various topics introduced in the current chapter with those that were explored in Chapter 3 on classical cognitive science.

4.1 Nurture versus Nature

The second chapter of John Locke's (1977) *An Essay Concerning Human Understanding*, originally published in 1706, begins as follows:

> It is an established opinion among some men that there are in the understanding certain innate principles; some primary notions, characters, as it were, stamped upon the mind of man, which the soul receives in its very first being, and brings into the world with it. (Locke, 1977, p. 17)

Locke's most famous work was a reaction against this view; of the “some men” being referred to, the most prominent was Descartes himself (Thilly, 1900).

Locke's *Essay* criticized Cartesian philosophy, questioning its fundamental teachings, its core principles and their necessary implications, and its arguments for innate ideas, not to mention all scholars who maintained the existence of innate ideas (Thilly, 1900). Locke's goal was to replace Cartesian rationalism with empiricism, the view that the source of ideas was experience. Locke (1977) aimed to show “how men, barely by the use of their natural faculties, may attain to all of the knowledge they have without the help of any innate impressions” (p. 17). Locke argued for experience over innateness, for nurture over nature.

The empiricism of Locke and his descendants provided a viable and popular alternative to Cartesian philosophy (Aune, 1970). It was also a primary influence on some of the psychological theories that appeared in the late nineteenth and early
twentieth centuries (Warren, 1921). Thus it should be no surprise that empiricism is
reflected in a different form of cognitive science, connectionism. Furthermore, just as
empiricism challenged most of the key ideas of rationalism, connectionist cognitive
science can be seen as challenging many of the elements of classical cognitive science.

Surprisingly, the primary concern of connectionist cognitive science is not
classical cognitive science’s nativism. It is instead the classical approach’s exces-
sive functionalism, due largely to its acceptance of the multiple realization argu-
ment. Logic gates, the core element of digital computers, are hardware independent
because different physical mechanisms could be used to bring the two-valued logic
into being (Hillis, 1998). The notion of a universal machine is an abstract, logical
one (Newell, 1980), which is why physical symbol systems, computers, or universal
machines can be physically realized using LEGO (Agulló et al., 2003), electric
train sets (Stewart, 1994), gears (Swade, 1993), hydraulic valves (Hillis, 1998) or sili-
con chips (Reid, 2001). Physical constraints on computation do not seem to play an
important role in classical cognitive science.

To connectionist cognitive science, the multiple realization argument is
flawed because connectionists believe that the information processing responsi-
ble for human cognition depends critically on the properties of particular hard-
ware, the brain. The characteristics of the brain place constraints on the kinds of
computations that it can perform and on the manner in which they are performed
(Bechtel & Abrahamsen, 2002; Churchland, Koch, & Sejnowski, 1990; Churchland
& Sejnowski, 1992; Clark, 1989, 1993; Feldman & Ballard, 1982).

Brains have long been viewed as being different kinds of information pro-
cessors than electronic computers because of differences in componentry (von
Neumann, 1958). While electronic computers use a small number of fast compo-
nents, the brain consists of a large number of very slow components, that is, neu-
rons. As a result, the brain must be a parallel processing device that “will tend to
pick up as many logical (or informational) items as possible simultaneously, and
process them simultaneously” (von Neumann, 1958, p. 51).

Von Neumann (1958) argued that neural information processing would be far
less precise, in terms of decimal point precision, than electronic information pro-
cessing. However, this low level of neural precision would be complemented by a
comparatively high level of reliability, where noise or missing information would
have far less effect than it would for electronic computers. Given that the basic
architecture of the brain involves many connections amongst many elementary
components, and that these connections serve as a memory, the brain’s memory
capacity should also far exceed that of digital computers.

The differences between electronic and brain-like information processing are
at the root of connectionist cognitive science’s reaction against classic cognitive
science. The classical approach has a long history of grand futuristic predictions
that fail to materialize (Dreyfus, 1992, p. 85): “Despite predictions, press releases, films, and warnings, artificial intelligence is a promise and not an accomplished fact.” Connectionist cognitive science argues that this pattern of failure is due to the fundamental assumptions of the classical approach that fail to capture the basic principles of human cognition.

Connectionists propose a very different theory of information processing—a potential paradigm shift (Schneider, 1987)—to remedy this situation. Even staunch critics of artificial intelligence research have indicated a certain sympathy with the connectionist view of information processing (Dreyfus & Dreyfus, 1988; Searle, 1992). “The fan club includes the most unlikely collection of people. . . . Almost everyone who is discontent with contemporary cognitive psychology and current ‘information processing’ models of the mind has rushed to embrace the ‘connectionist alternative’” (Fodor & Pylyshyn, 1988, p. 4).

What are the key problems that connectionists see in classical models? Classical models invoke serial processes, which make them far too slow to run on sluggish componentry (Feldman & Ballard, 1982). They involve explicit, local, and digital representations of both rules and symbols, making these models too brittle. “If in a digital system of notations a single pulse is missing, absolute perversion of meaning, i.e., nonsense, may result” (von Neumann, 1958, p. 78). Because of this brittleness, the behaviour of classical models does not degrade gracefully when presented with noisy inputs, and such models are not damage resistant. All of these issues arise from one underlying theme: classical algorithms reflect the kind of information processing carried out by electronic computers, not the kind that characterizes the brain. In short, classical theories are not biologically plausible.

Connectionist cognitive science “offers a radically different conception of the basic processing system of the mind-brain, one inspired by our knowledge of the nervous system” (Bechtel & Abrahamsen, 2002, p. 2). The basic medium of connectionism is a type of model called an artificial neural network, or a parallel distributed processing (PDP) network (McClelland & Rumelhart, 1986; Rumelhart & McClelland, 1986c). Artificial neural networks consist of a number of simple processors that perform basic calculations and communicate the results to other processors by sending signals through weighted connections. The processors operate in parallel, permitting fast computing even when slow componentry is involved. They exploit implicit, distributed, and redundant representations, making these networks not brittle. Because networks are not brittle, their behaviour degrades gracefully when presented with noisy inputs, and such models are damage resistant. These advantages accrue because artificial neural networks are intentionally biologically plausible or neuronally inspired.

Classical cognitive science develops models that are purely symbolic and which can be described as asserting propositions or performing logic. In contrast,
connectionist cognitive science develops models that are subsymbolic (Smolensky, 1988) and which can be described as statistical pattern recognizers. Networks use representations (Dawson, 2004; Horgan & Tienson, 1996), but these representations do not have the syntactic structure of those found in classical models (Waskan & Bechtel, 1997). Let us take a moment to describe in a bit more detail the basic properties of artificial neural networks.

An artificial neural network is a computer simulation of a “brain-like” system of interconnected processing units (see Figures 4-1 and 4-5 later in this chapter). In general, such a network can be viewed as a multiple-layer system that generates a desired response to an input stimulus. That is, like the devices described by cybernetics (Ashby, 1956, 1960), an artificial neural network is a machine that computes a mapping between inputs and outputs.

A network’s stimulus or input pattern is provided by the environment and is encoded as a pattern of activity (i.e., a vector of numbers) in a set of input units. The response of the system, its output pattern, is represented as a pattern of activity in the network’s output units. In modern connectionism—sometimes called New Connectionism—there will be one or more intervening layers of processors in the network, called hidden units. Hidden units detect higher-order features in the input pattern, allowing the network to make a correct or appropriate response.

The behaviour of a processor in an artificial neural network, which is analogous to a neuron, can be characterized as follows. First, the processor computes the total signal (its net input) being sent to it by other processors in the network. Second, the unit uses an activation function to convert its net input into internal activity (usually a continuous number between 0 and 1) on the basis of this computed signal. Third, the unit converts its internal activity into an output signal, and sends this signal on to other processors. A network uses parallel processing because many, if not all, of its units will perform their operations simultaneously.

The signal sent by one processor to another is a number that is transmitted through a weighted connection, which is analogous to a synapse. The connection serves as a communication channel that amplifies or attenuates signals being sent through it, because these signals are multiplied by the weight associated with the connection. The weight is a number that defines the nature and strength of the connection. For example, inhibitory connections have negative weights, and excitatory connections have positive weights. Strong connections have strong weights (i.e., the absolute value of the weight is large), while weak connections have near-zero weights.

The pattern of connectivity in a PDP network (i.e., the network’s entire set of connection weights) defines how signals flow between the processors. As a result, a network’s connection weights are analogous to a program in a conventional computer (Smolensky, 1988). However, a network’s “program” is not of the same type that defines a classical model. A network’s program does not reflect the classical
structure/process distinction, because networks do not employ either explicit symbols or rules. Instead, a network’s program is a set of causal or associative links from signaling processors to receiving processors. The activity that is produced in the receiving units is literally caused by having an input pattern of activity modulated by an array of connection weights between units. In this sense, connectionist models seem markedly associationist in nature (Bechtel, 1985); they can be comfortably related to the old associationist psychology (Warren, 1921).

Artificial neural networks are not necessarily embodiments of empiricist philosophy. Indeed, the earliest artificial neural networks did not learn from experience; they were nativist in the sense that they had to have their connection weights “hand wired” by a designer (McCulloch & Pitts, 1943). However, their associationist characteristics resulted in a natural tendency for artificial neural networks to become the face of modern empiricism. This is because associationism has always been strongly linked to empiricism; empiricist philosophers invoked various laws of association to explain how complex ideas could be constructed from the knowledge provided by experience (Warren, 1921). By the late 1950s, when computers were being used to bring networks to life, networks were explicitly linked to empiricism (Rosenblatt, 1958). Rosenblatt’s artificial neural networks were not hand wired. Instead, they learned from experience to set the values of their connection weights.

What does it mean to say that artificial neural networks are empiricist? A famous passage from Locke (1977, p. 54) highlights two key elements: “Let us then suppose the mind to be, as we say, white paper, void of all characters, without any idea, how comes it to be furnished? . . . To this I answer, in one word, from experience.”

The first element in the above quote is the “white paper,” often described as the tabula rasa, or the blank slate: the notion of a mind being blank in the absence of experience. Modern connectionist networks can be described as endorsing the notion of the blank slate (Pinker, 2002). This is because prior to learning, the pattern of connections in modern networks has no pre-existing structure. The networks either start literally as blank slates, with all connection weights being equal to zero (Anderson et al., 1977; Eich, 1982; Hinton & Anderson, 1981), or they start with all connection weights being assigned small, randomly selected values (Rumelhart, Hinton, & Williams, 1986a, 1986b).

The second element in Locke’s quote is that the source of ideas or knowledge or structure is experience. Connectionist learning rules provide a modern embodiment of this notion. Artificial neural networks are exposed to environmental stimulation—activation of their input units—which results in changes to connection weights. These changes furnish a network’s blank slate, resulting in a pattern of connectivity that represents knowledge and implements a particular input-output mapping.

In some systems, called self-organizing networks, experience shapes connectivity
via unsupervised learning (Carpenter & Grossberg, 1992; Grossberg, 1987, 1988; Kohonen, 1977, 1984). When learning is unsupervised, networks are only provided with input patterns. They are not presented with desired outputs that are paired with each input pattern. In unsupervised learning, each presented pattern causes activity in output units; this activity is often further refined by a winner-take-all competition in which one output unit wins the competition to be paired with the current input pattern. Once the output unit is selected via internal network dynamics, its connection weights, and possibly the weights of neighbouring output units, are updated via a learning rule.

Networks whose connection weights are modified via unsupervised learning develop sensitivity to statistical regularities in the inputs and organize their output units to reflect these regularities. For instance, in a famous kind of self-organizing network called a Kohonen network (Kohonen, 1984), output units are arranged in a two-dimensional grid. Unsupervised learning causes the grid to organize itself into a map that reveals the discovered structure of the inputs, where related patterns produce neighbouring activity in the output map. For example, when such networks are presented with musical inputs, they often produce output maps that are organized according to the musical circle of fifths (Griffith & Todd, 1999; Todd & Loy, 1991).

In cognitive science, most networks reported in the literature are not self-organizing and are not structured via unsupervised learning. Instead, they are networks that are instructed to mediate a desired input-output mapping. This is accomplished via supervised learning. In supervised learning, it is assumed that the network has an external teacher. The network is presented with an input pattern and produces a response to it. The teacher compares the response generated by the network to the desired response, usually by calculating the amount of error associated with each output unit. The teacher then provides the error as feedback to the network. A learning rule uses feedback about error to modify weights in such a way that the next time this pattern is presented to the network, the amount of error that it produces will be smaller.

A variety of learning rules, including the delta rule (Rosenblatt, 1958, 1962; Stone, 1986; Widrow, 1962; Widrow & Hoff, 1960) and the generalized delta rule (Rumelhart, Hinton, & Williams, 1986b), are supervised learning rules that work by correcting network errors. (The generalized delta rule is perhaps the most popular learning rule in modern connectionism, and is discussed in more detail in Section 4.9.) This kind of learning involves the repeated presentation of a number of input-output pattern pairs, called a training set. Ideally, with enough presentations of a training set, the amount of error produced to each member of the training set will be negligible, and it can be said that the network has learned the desired input-output mapping. Because these techniques require many presentations of a set of
patterns for learning to be completed, they have sometimes been criticized as being examples of “slow learning” (Carpenter, 1989).

Connectionism’s empiricist and associationist nature cast it close to the very position that classical cognitivism reacted against: psychological behaviourism (Miller, 2003). Modern classical arguments against connectionist cognitive science (Fodor & Pylyshyn, 1988) cover much of the same ground as arguments against behaviourist and associationist accounts of language (Bever, Fodor, & Garrett, 1968; Chomsky, 1957, 1959a, 1959b, 1965). That is, classical cognitive scientists argue that artificial neural networks, like their associationist cousins, do not have the computational power to capture the kind of regularities modelled with recursive rule systems.

However, these arguments against connectionism are flawed. We see in later sections that computational analyses of artificial neural networks have proven that they too belong to the class “universal machine.” As a result, the kinds of input-output mappings that have been realized in artificial neural networks are both vast and diverse. One can find connectionist models in every research domain that has also been explored by classical cognitive scientists. Even critics of connectionism admit that “the study of connectionist machines has led to a number of striking and unanticipated findings; it’s surprising how much computing can be done with a uniform network of simple interconnected elements” (Fodor & Pylyshyn, 1988, p. 6).

That connectionist models can produce unanticipated results is a direct result of their empiricist nature. Unlike their classical counterparts, connectionist researchers do not require a fully specified theory of how a task is accomplished before modelling begins (Hillis, 1988). Instead, they can let a learning rule discover how to mediate a desired input-output mapping. Connectionist learning rules serve as powerful methods for developing new algorithms of interest to cognitive science. Hillis (1988, p. 176) has noted that artificial neural networks allow “for the possibility of constructing intelligence without first understanding it.”

One problem with connectionist cognitive science is that the algorithms that learning rules discover are extremely difficult to retrieve from a trained network (Dawson, 1998, 2004, 2009; Dawson & Shamanski, 1994; McCloskey, 1991; Mozer & Smolensky, 1989; Seidenberg, 1993). This is because these algorithms involve distributed, parallel interactions amongst highly nonlinear elements. “One thing that connectionist networks have in common with brains is that if you open them up and peer inside, all you can see is a big pile of goo” (Mozer & Smolensky, 1989, p. 3).

In the early days of modern connectionist cognitive science, this was not a concern. This was a period of what has been called “gee whiz” connectionism (Dawson, 2009), in which connectionists modelled phenomena that were typically described in terms of rule-governed symbol manipulation. In the mid-1980s it was sufficiently interesting to show that such phenomena might be accounted for by parallel distributed processing systems that did not propose explicit rules or
symbols. However, as connectionism matured, it was necessary for its researchers to spell out the details of the alternative algorithms embodied in their networks (Dawson, 2004). If these algorithms could not be extracted from networks, then “connectionist networks should not be viewed as theories of human cognitive functions, or as simulations of theories, or even as demonstrations of specific theoretical points” (McCloskey, 1991, p. 387). In response to such criticisms, connectionist cognitive scientists have developed a number of techniques for recovering algorithms from their networks (Berkeley et al., 1995; Dawson, 2004, 2005; Gallant, 1993; Hanson & Burr, 1990; Hinton, 1986; Moorhead, Haig, & Clement, 1989; Omlin & Giles, 1996).

What are the elements of connectionism, and how do they relate to cognitive science in general and to classical cognitive science in particular? The purpose of the remainder of this chapter is to explore the ideas of connectionist cognitive science in more detail.

4.2 Associations

Classical cognitive science has been profoundly influenced by seventeenth-century Cartesian philosophy (Descartes, 1996, 2006). The Cartesian view that thinking is equivalent to performing mental logic—that it is a mental discourse of computation or calculation (Hobbes, 1967)—has inspired the logicism that serves as the foundation of the classical approach. Fundamental classical notions, such as the assumption that cognition is the result of rule-governed symbol manipulation (Craik, 1943) or that innate knowledge is required to solve problems of underdetermination (Chomsky, 1965, 1966), have resulted in the classical being viewed as a newer variant of Cartesian rationalism (Paivio, 1986). One key classical departure from Descartes is its rejection of dualism. Classical cognitive science has appealed to recursive rules to permit finite devices to generate an infinite variety of potential behaviour.

Classical cognitive science is the modern rationalism, and one of the key ideas that it employs is recursion. Connectionist cognitive science has very different philosophical roots. Connectionism is the modern form of empiricist philosophy (Berkeley, 1710; Hume, 1952; Locke, 1977), where knowledge is not innate, but is instead provided by sensing the world. “No man's knowledge here can go beyond his experience” (Locke, 1977, p. 83). If recursion is fundamental to the classical approach's rationalism, then what notion is fundamental to connectionism’s empiricism? The key idea is association: different ideas can be linked together, so that if one arises, then the association between them causes the other to arise as well.

For centuries, philosophers and psychologists have studied associations empirically, through introspection (Warren, 1921). These introspections have revealed the existence of sequences of thought that occur during thinking. Associationism attempted to determine the laws that would account for these sequences of thought.
The earliest detailed introspective account of such sequences of thought can be found in the 350 BC writings of Aristotle (Sorabji, 2006, p. 54): “Acts of recollection happen because one change is of a nature to occur after another.” For Aristotle, ideas were images (Cummins, 1989). He argued that a particular sequence of images occurs either because this sequence is a natural consequence of the images, or because the sequence has been learned by habit. Recall of a particular memory, then, is achieved by cuing that memory with the appropriate prior images, which initiate the desired sequence of images. “Whenever we recollect, then, we undergo one of the earlier changes, until we undergo the one after which the change in question habitually occurs” (Sorabji, 2006, p. 54).

Aristotle’s analysis of sequences of thought is central to modern mnemonic techniques for remembering ordered lists (Lorayne, 2007; Lorayne & Lucas, 1974). Aristotle noted that recollection via initiating a sequence of mental images could be a deliberate and systematic process. This was because the first image in the sequence could be selected so that it would be recollected fairly easily. Recall of the sequence, or of the target image at the end of the sequence, was then dictated by lawful relationships between adjacent ideas. Thus Aristotle invented laws of association.

Aristotle considered three different kinds of relationships between the starting image and its successor: similarity, opposition, and (temporal) contiguity:

And this is exactly why we hunt for the successor, starting in our thoughts from the present or from something else, and from something similar, or opposite, or neighbouring. By this means recollection occurs. (Sorabji, 2006, p. 54)

In more modern associationist theories, Aristotle’s laws would be called the law of similarity, the law of contrast, and the law of contiguity or the law of habit.

Aristotle’s theory of memory was essentially ignored for many centuries (Warren, 1921). Instead, pre-Renaissance and Renaissance Europe were more interested in the artificial memory—mnemonics—that was the foundation of Greek oratory. These techniques were rediscovered during the Middle Ages in the form of Ad Herennium, a circa 86 BC text on rhetoric that included a section on enhancing the artificial memory (Yates, 1966). Ad Herennium described the mnemonic techniques invented by Simonides circa 500 BC. While the practice of mnemonics flourished during the Middle Ages, it was not until the seventeenth century that advances in associationist theories of memory and thought began to flourish.

The rise of modern associationism begins with Thomas Hobbes (Warren, 1921). Hobbes’ (1967) notion of thought as mental discourse was based on his observation that thinking involved an orderly sequence of ideas. Hobbes was interested in explaining how such sequences occurred. While Hobbes’ own work was very preliminary, it inspired more detailed analyses carried out by the British empiricists who followed him.
Empiricist philosopher John Locke coined the phrase *association of ideas*, which first appeared as a chapter title in the fourth edition of *An Essay Concerning Human Understanding* (Locke, 1977). Locke's work was an explicit reaction against Cartesian philosophy (Thilly, 1900); his goal was to establish experience as the foundation of all thought. He noted that connections between simple ideas might not reflect a natural order. Locke explained this by appealing to experience:

> Ideas that in themselves are not at all of kin, come to be so united in some men's minds that it is very hard to separate them, they always keep in company, and the one no sooner at any time comes into the understanding but its associate appears with it. (Locke, 1977, p. 122)

Eighteenth-century British empiricists expanded Locke's approach by exploring and debating possible laws of association. George Berkeley (1710) reiterated Aristotle's law of contiguity and extended it to account for associations involving different modes of sensation. David Hume (1852) proposed three different laws of association: resemblance, contiguity in time or place, and cause or effect. David Hartley, one of the first philosophers to link associative laws to brain function, saw contiguity as the primary source of associations and ignored Hume's law of resemblance (Warren, 1921).

Debates about the laws of association continued into the nineteenth century. James Mill (1829) only endorsed the law of contiguity, and explicitly denied Hume's laws of cause and effect or resemblance. Mill's ideas were challenged and modified by his son, John Stuart Mill. In his revised version of his father's book (Mill & Mill, 1869), Mill posited a completely different set of associative laws, which included a reintroduction of Hume's law of similarity. He also replaced his father's linear, mechanistic account of complex ideas with a “mental chemistry” that endorsed nonlinear emergence. This is because in this mental chemistry, when complex ideas were created via association, the resulting whole was more than just the sum of its parts. Alexander Bain (1855) refined the associationism of John Stuart Mill, proposing four different laws of association and attempting to reduce all intellectual processes to these laws. Two of these were the familiar laws of contiguity and of similarity.

Bain was the bridge between philosophical and psychological associationism (Boring, 1950). He stood,

> exactly at a corner in the development of psychology, with philosophical psychology stretching out behind, and experimental physiological psychology lying ahead, in a new direction. The psychologists of the twentieth century can read much of Bain with hearty approval; perhaps John Locke could have done the same. (Boring, 1950, p. 240)

One psychologist who approved of Bain was William James; he frequently cited Bain in his *Principles of Psychology* (James, 1890a). Chapter 14 of this work provided
James' own treatment of associationism. James criticized philosophical associationism's emphasis on associations between mental contents. James proposed a mechanistic, biological theory of associationism instead, claiming that associations were made between brain states:

> We ought to talk of the association of objects, not of the association of ideas. And so far as association stands for a cause, it is between processes in the brain—it is these which, by being associated in certain ways, determine what successive objects shall be thought. (James, 1890a, p. 554, original italics)

James (1890a) attempted to reduce other laws of association to the law of contiguity, which he called the law of habit and expressed as follows: “When two elementary brain-processes have been active together or in immediate succession, one of them, on reoccurring, tends to propagate its excitement into the other” (p. 566). He illustrated the action of this law with a figure (James, 1890a, p. 570, Figure 40), a version of which is presented as Figure 4-1.

Figure 4-1 illustrates two ideas, A and B, each represented as a pattern of activity in its own set of neurons. A is represented by activity in neurons a, b, c, d, and e; B is represented by activity in neurons l, m, n, o, and p. The assumption is that A represents an experience that occurred immediately before B. When B occurs, activating its neurons, residual activity in the neurons representing A permits the two patterns to be associated by the law of habit. That is, the “tracts” connecting the neurons (the “modifiable connections” in Figure 4-1) have their strengths modified.
The ability of A’s later activity to reproduce B is due to these modified connections between the two sets of neurons.

The thought of A must awaken that of B, because a, b, c, d, e, will each and all discharge into l through the paths by which their original discharge took place. Similarly they will discharge into m, n, o, and p; and these latter tracts will also each reinforce the other’s action because, in the experience B, they have already vibrated in unison. (James, 1890a, p. 569)

James’ (1890a) biological account of association reveals three properties that are common to modern connectionist networks. First, his system is parallel: more than one neuron can be operating at the same time. Second, his system is convergent: the activity of one of the output neurons depends upon receiving or summing the signals sent by multiple input neurons. Third, his system is distributed: the association between A and B is the set of states of the many “tracts” illustrated in Figure 4-1; there is not just a single associative link.

James’s (1890a) law of habit was central to the basic mechanism proposed by neuroscientist Donald Hebb (1949) for the development of cell assemblies. Hebb provided a famous modern statement of James’ law of habit:

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased. (Hebb, 1949, p. 62)

This makes explicit the modern connectionist idea that learning is modifying the strength of connections between processors. Hebb’s theory inspired the earliest computer simulations of memory systems akin to the one proposed by James (Milner, 1957; Rochester et al., 1956). These simulations revealed a critical role for inhibition that led Hebb (1959) to revise his theory. Modern neuroscience has discovered a phenomenon called long-term potentiation that is often cited as a biologically plausible instantiation of Hebb’s theory (Brown, 1990; Gerstner & Kistler, 2002; Martinez & Derrick, 1996; van Hemmen & Senn, 2002).

The journey from James through Hebb to the first simulations of memory (Milner, 1957; Rochester et al., 1956) produced a modern associative memory system called the standard pattern associator (McClelland, 1986). The standard pattern associator, which is structurally identical to Figure 4-1, is a memory capable of learning associations between pairs of input patterns (Steinbuch, 1961; Taylor, 1956) or learning to associate an input pattern with a categorizing response (Rosenblatt, 1962; Selfridge, 1956; Widrow & Hoff, 1960).

The standard pattern associator is empiricist in the sense that its knowledge is acquired by experience. Usually the memory begins as a blank slate: all of the connections between processors start with weights equal to zero. During a learning
phase, pairs of to-be-associated patterns simultaneously activate the input and output units in Figure 4-1. With each presented pair, all of the connection weights—the strength of each connection between an input and an output processor—are modified by adding a value to them. This value is determined in accordance with some version of Hebb’s (1949) learning rule. Usually, the value added to a weight is equal to the activity of the processor at the input end of the connection, multiplied by the activity of the processor at the output end of the connection, and multiplied by some fractional value called a learning rate. The mathematical details of such learning are provided in Chapter 9 of Dawson (2004).

The standard pattern associator is called a distributed memory because its knowledge is stored throughout all the connections in the network, and because this one set of connections can store several different associations. During a recall phase, a cue pattern is used to activate the input units. This causes signals to be sent through the connections in the network. These signals are equal to the activation value of an input unit multiplied by the weight of the connection through which the activity is being transmitted. Signals received by the output processors are used to compute net input, which is simply the sum of all of the incoming signals. In the standard pattern associator, an output unit’s activity is equal to its net input. If the memory is functioning properly, then the pattern of activation in the output units will be the pattern that was originally associated with the cue pattern.

The standard pattern associator is the cornerstone of many models of memory created after the cognitive revolution (Anderson, 1972; Anderson et al., 1977; Eich, 1982; Hinton & Anderson, 1981; Murdock, 1982; Pike, 1984; Steinbuch, 1961; Taylor, 1956). These models are important, because they use a simple principle—James’ (1890a, 1890b) law of habit—to model many subtle regularities of human memory, including errors in recall. In other words, the standard pattern associator is a kind of memory that has been evaluated with the different kinds of evidence cited in Chapters 2 and 3, in an attempt to establish strong equivalence.

The standard pattern associator also demonstrates another property crucial to modern connectionism, graceful degradation. How does this distributed model behave if it is presented with a noisy cue, or with some other cue that was never tested during training? It generates a response that has the same degree of noise as its input (Dawson, 1998, Table 3-1). That is, there is a match between the quality of the memory’s input and the quality of its output.

The graceful degradation of the standard pattern associator reveals that it is sensitive to the similarity of noisy cues to other cues that were presented during training. Thus modern pattern associators provide some evidence for James’ (1890a) attempt to reduce other associative laws, such as the law of similarity, to the basic law of habit or contiguity.
In spite of the popularity and success of distributed associative memories as models of human learning and recall (Hinton & Anderson, 1981), they are extremely limited in power. When networks learn via the Hebb rule, they produce errors when they are overtrained, are easily confused by correlated training patterns, and do not learn from their errors (Dawson, 2004). An error-correcting rule called the delta rule (Dawson, 2004; Rosenblatt, 1962; Stone, 1986; Widrow & Hoff, 1960) can alleviate some of these problems, but it does not eliminate them. While association is a fundamental notion in connectionist models, other notions are required by modern connectionist cognitive science. One of these additional ideas is nonlinear processing.

4.3 Nonlinear Transformations

John Stuart Mill modified his father’s theory of associationism (Mill & Mill, 1869; Mill, 1848) in many ways, including proposing a mental chemistry “in which it is proper to say that the simple ideas generate, rather than . . . compose, the complex ones” (Mill, 1848, p. 533). Mill’s mental chemistry is an early example of emergence, where the properties of a whole (i.e., a complex idea) are more than the sum of the properties of the parts (i.e., a set of associated simple ideas).

The generation of one class of mental phenomena from another, whenever it can be made out, is a highly interesting fact in mental chemistry; but it no more supersedes the necessity of an experimental study of the generated phenomenon than a knowledge of the properties of oxygen and sulphur enables us to deduce those of sulphuric acid without specific observation and experiment. (Mill, 1848, p. 534)

Mathematically, emergence results from nonlinearity (Luce, 1999). If a system is linear, then its whole behaviour is exactly equal to the sum of the behaviours of its parts. The standard pattern associator that was illustrated in Figure 4-1 is an example of such a system. Each output unit in the standard pattern associator computes a net input, which is the sum of all of the individual signals that it receives from the input units. Output unit activity is exactly equal to net input. In other words, output activity is exactly equal to the sum of input signals in the standard pattern associator. In order to increase the power of this type of pattern associator—in order to facilitate emergence—a nonlinear relationship between input and output must be introduced.

Neurons demonstrate one powerful type of nonlinear processing. The inputs to a neuron are weak electrical signals, called graded potentials, which stimulate and travel through the dendrites of the receiving neuron. If enough of these weak graded potentials arrive at the neuron’s soma at roughly the same time, then their cumulative effect disrupts the neuron’s resting electrical state. This results in a massive
depolarization of the membrane of the neuron’s axon, called an action potential, which is a signal of constant intensity that travels along the axon to eventually stimulate some other neuron.

A crucial property of the action potential is that it is an all-or-none phenomenon, representing a nonlinear transformation of the summed graded potentials. The neuron converts continuously varying inputs into a response that is either on (action potential generated) or off (action potential not generated). This has been called the all-or-none law (Levitan & Kaczmarek, 1991, p. 43): “The all-or-none law guarantees that once an action potential is generated it is always full size, minimizing the possibility that information will be lost along the way.” The all-or-none output of neurons is a nonlinear transformation of summed, continuously varying input, and it is the reason that the brain can be described as digital in nature (von Neumann, 1958).

The all-or-none behaviour of a neuron makes it logically equivalent to the relays or switches that were discussed in Chapter 2. This logical interpretation was exploited in an early mathematical account of the neural information processing (McCulloch & Pitts, 1943). McCulloch and Pitts used the all-or-none law to justify describing neurons very abstractly as devices that made true or false logical assertions about input information:

The all-or-none law of nervous activity is sufficient to insure that the activity of any neuron may be represented as a proposition. Physiological relations existing among nervous activities correspond, of course, to relations among the propositions; and the utility of the representation depends upon the identity of these relations with those of the logical propositions. To each reaction of any neuron there is a corresponding assertion of a simple proposition. (McCulloch & Pitts, 1943, p. 117)

McCulloch and Pitts (1943) invented a connectionist processor, now known as the McCulloch-Pitts neuron (Quinlan, 1991), that used the all-or-none law. Like the output units in the standard pattern associator (Figure 4-1), a McCulloch-Pitts neuron first computes its net input by summing all of its incoming signals. However, it then uses a nonlinear activation function to transform net input into internal activity. The activation function used by McCulloch and Pitts was the Heaviside step function, named after nineteenth-century electrical engineer Oliver Heaviside. This function compares the net input to a threshold. If the net input is less than the threshold, the unit’s activity is equal to 0. Otherwise, the unit’s activity is equal to 1. (In other artificial neural networks [Rosenblatt, 1958, 1962], below-threshold net inputs produced activity of −1.)

The output units in the standard pattern associator (Figure 4-1) can be described as using the linear identity function to convert net input into activity, because output unit activity is equal to net input. If one replaced the identity function with the Heaviside step function in the standard pattern associator, it would
then become a different kind of network, called a perceptron (Dawson, 2004), which was invented by Frank Rosenblatt during the era in which cognitive science was born (Rosenblatt, 1958, 1962).

Perceptrons (Rosenblatt, 1958, 1962) were artificial neural networks that could be trained to be pattern classifiers: given an input pattern, they would use their nonlinear outputs to decide whether or not the pattern belonged to a particular class. In other words, the nonlinear activation function used by perceptrons allowed them to assign perceptual predicates; standard pattern associators do not have this ability. The nature of the perceptual predicates that a perceptron could learn to assign was a central issue in an early debate between classical and connectionist cognitive science (Minsky & Papert, 1969; Papert, 1988).

The Heaviside step function is nonlinear, but it is also discontinuous. This was problematic when modern researchers sought methods to train more complex networks. Both the standard pattern associator and the perceptron are one-layer networks, meaning that they have only one layer of connections, the direct connections between input and output units (Figure 4-1). More powerful networks arise if intermediate processors, called hidden units, are used to preprocess input signals before sending them on to the output layer. However, it was not until the mid-1980s that learning rules capable of training such networks were invented (Ackley, Hinton, & Sejnowski, 1985; Rumelhart, Hinton, & Williams, 1986b). The use of calculus to derive these new learning rules became possible when the discontinuous Heaviside step function was replaced by a continuous approximation of the all-or-none law (Rumelhart, Hinton, & Williams, 1986b).

One continuous approximation of the Heaviside step function is the sigmoid-shaped logistic function. It asymptotes to a value of 0 as its net input approaches negative infinity, and asymptotes to a value of 1 as its net input approaches positive infinity. When the net input is equal to the threshold (or bias) of the logistic, activity is equal to 0.5. Because the logistic function is continuous, its derivative can be calculated, and calculus can be used as a tool to derive new learning rules (Rumelhart, Hinton, & Williams, 1986b). However, it is still nonlinear, so logistic activities can still be interpreted as truth values assigned to propositions.

Modern connectionist networks employ many different nonlinear activation functions. Processing units that employ the logistic activation function have been called integration devices (Ballard, 1986) because they convert a sum (net input) and “squash” it into the range between 0 and 1. Other processing units might be tuned to generate maximum responses to a narrow range of net inputs. Ballard (1986) called such processors value units. A different nonlinear continuous function, the Gaussian equation, can be used to mathematically define a value unit, and calculus can be used to derive a learning rule for this type of artificial neural network (Dawson, 1998, 2004; Dawson & Schopflocher, 1992b).
Many other activation functions exist. One review paper has identified 640 different activation functions employed in connectionist networks (Duch & Jankowski, 1999). One characteristic of the vast majority of all of these activation functions is their nonlinearity. Connectionist cognitive science is associationist, but it is also nonlinear.

4.4 The Connectionist Sandwich

Both the McCulloch-Pitts neuron (McCulloch & Pitts, 1943) and the perceptron (Rosenblatt, 1958, 1962) used the Heaviside step function to implement the all-or-none law. As a result, both of these architectures generated a “true” or “false” judgment about each input pattern. Thus both of these architectures are digital, and their basic function is pattern recognition or pattern classification.

The two-valued logic that was introduced in Chapter 2 can be cast in the context of such digital pattern recognition. In the two-valued logic, functions are computed over two input propositions, \( p \) and \( q \), which themselves can either be true or false. As a result, there are only four possible combinations of \( p \) and \( q \), which are given in the first two columns of Table 4-1. Logical functions in the two-valued logic are themselves judgments of true or false that depend on combinations of the truth values of the input propositions \( p \) and \( q \). As a result, there are 16 different logical operations that can be defined in the two-valued logic; these were provided in Table 2-2.

The truth tables for two of the sixteen possible operations in the two-valued logic are provided in the last two columns of Table 4-1. One is the AND operation \((p \cdot q)\), which is only true when both propositions are true. The other is the XOR operation \((p \lor q)\), which is only true when one or the other of the propositions is true.

<table>
<thead>
<tr>
<th></th>
<th>( p )</th>
<th>( q )</th>
<th>( p \cdot q )</th>
<th>( p \lor q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Table 4-1.* Truth tables for the logical operations AND \((p \cdot q)\) and XOR \((p \lor q)\), where the truth value of each operation is given as a function of the truth of each of two propositions, \( p \) and \( q \). ‘1’ indicates “true” and ‘0’ indicates “false.” The logical notation is taken from McCulloch (1988b).

That AND or XOR are examples of digital pattern recognition can be made more explicit by representing their truth tables graphically as pattern spaces. In a pattern
space, an entire row of a truth table is represented as a point on a graph. The coordinates of a point in a pattern space are determined by the truth values of the input propositions. The colour of the point represents the truth value of the operation computed over the inputs.

Figure 4-2A illustrates the pattern space for the AND operation of Table 4-1. Note that it has four graphed points, one for each row of the truth table. The coordinates of each graphed point—(1,1), (1,0), (0,1), and (0,0)—indicate the truth values of the propositions p and q. The AND operation is only true when both of these propositions are true. This is represented by colouring the point at coordinate (1,1) black. The other three points are coloured white, indicating that the logical operator returns a “false” value for each of them.

Pattern spaces are used for digital pattern recognition by carving them into decision regions. If a point that represents a pattern falls in one decision region, then it is classified in one way. If that point falls in a different decision region, then it is classified in a different way. Learning how to classify a set of patterns involves learning how to correctly carve the pattern space up into the desired decision regions.

The AND problem is an example of a linearly separable problem. This is because a single straight cut through the pattern space divides it into two decision regions that generate the correct pattern classifications. The dashed line in Figure 4-2A indicates the location of this straight cut for the AND problem. Note that the one “true” pattern falls on one side of this cut, and that the three “false” patterns fall on the other side of this cut.

Not all problems are linearly separable. A linearly nonseparable problem is one in which a single straight cut is not sufficient to separate all of the patterns of one type from all of the patterns of another type. An example of a linearly nonseparable problem is the XOR problem, whose pattern space is illustrated in Figure 4-2B. Note that the positions of the four patterns in Figure 4-2B are identical to the positions in Figure 4-2A, because both pattern spaces involve the same propositions.
The only difference is the colouring of the points, indicating that XOR involves making a different judgment than AND. However, this difference between graphs is important, because now it is impossible to separate all of the black points from all of the white points with a single straight cut. Instead, two different cuts are required, as shown by the two dashed lines in Figure 4-2B. This means that XOR is not linearly separable.

Linear separability defines the limits of what can be computed by a Rosenblatt perceptron (Rosenblatt, 1958, 1962) or by a McCulloch-Pitts neuron (McCulloch & Pitts, 1943). That is, if some pattern recognition problem is linearly separable, then either of these architectures is capable of representing a solution to that problem. For instance, because AND is linearly separable, it can be computed by a perceptron, such as the one illustrated in Figure 4-3.

Figure 4-3. A Rosenblatt perceptron that can compute the AND operation.

This perceptron consists of two input units whose activities respectively represent the state (i.e., either 0 or 1) of the propositions $p$ and $q$. Each of these input units sends a signal through a connection to an output unit; the figure indicates that the weight of each connection is 1. The output unit performs two operations. First, it computes its net input by summing the two signals that it receives (the $\Sigma$ component of the output unit). Second, it transforms the net input into activity by applying the Heaviside step function. The figure indicates in the second component of the output unit that the threshold for this activation function ($\theta$) is 1.5. This means that output unit activity will only be 1 if net input is greater than or equal to 1.5; otherwise, output unit activity will be equal to 0.

If one considers the four different combinations of input unit activities that would be presented to this device—(1,1), (1,0), (0,1), and (0,0)—then it is clear that
the only time that output unit activity will equal 1 is when both input units are activated with 1 (i.e., when \( p \) and \( q \) are both true). This is because this situation will produce a net input of 2, which exceeds the threshold. In all other cases, the net input will either be 1 or 0, which will be less than the threshold, and which will therefore produce output unit activity of 0.

The ability of the Figure 4-3 perceptron to compute AND can be described in terms of the pattern space in Figure 4-2A. The threshold and the connection weights of the perceptron provide the location and orientation of the single straight cut that carves the pattern space into decision regions (the dashed line in Figure 4-2A). Activating the input units with some pattern presents a pattern space location to the perceptron. The perceptron examines this location to decide on which side of the cut the location lies, and responds accordingly.

This pattern space account of the Figure 4-3 perceptron also points to a limitation. When the Heaviside step function is used as an activation function, the perceptron only defines a single straight cut through the pattern space and therefore can only deal with linearly separable problems. A perceptron akin to the one illustrated in Figure 4-3 would not be able to compute XOR (Figure 4-2B) because the output unit is incapable of making the two required cuts in the pattern space.

How does one extend computational power beyond the perceptron? One approach is to add additional processing units, called hidden units, which are intermediaries between input and output units. Hidden units can detect additional features that transform the problem by increasing the dimensionality of the pattern space. As a result, the use of hidden units can convert a linearly nonseparable problem into a linearly separable one, permitting a single binary output unit to generate the correct responses.

Figure 4-4 shows how the AND circuit illustrated in Figure 4-3 can be added as a hidden unit to create a multilayer perceptron that can compute the linearly nonseparable XOR operation (Rumelhart, Hinton, & Williams, 1986a). This perceptron also has two input units whose activities respectively represent the state of the propositions \( p \) and \( q \). Each of these input units sends a signal through a connection to an output unit; the figure indicates that the weight of each connection is 1. The threshold of the output’s activation function (\( \theta \)) is 0.5. If we were to ignore the hidden unit in this network, the output unit would be computing OR, turning on when one or both of the input propositions are true.

However, this network does not compute OR, because the input units are also connected to a hidden unit, which in turn sends a third signal to be added into the output unit’s net input. The hidden unit is identical to the AND circuit from Figure 4-3. The signal that it sends to the output unit is strongly inhibitory; the weight of the connection between the two units is –2.
The action of the hidden unit is crucial to the behaviour of the system. When neither or only one of the input units activates, the hidden unit does not respond, so it sends a signal of 0 to the output unit. As a result, in these three situations the output unit turns on when either of the inputs is on (because the net input is over the threshold) and turns off when neither input unit is on. When both input units are on, they send an excitatory signal to the output unit. However, they also send a signal that turns on the hidden unit, causing it to send inhibition to the output unit. In this situation, the net input of the output unit is \(1 + 1 - 2 = 0\) which is below threshold, producing zero output unit activity. The entire circuit therefore performs the XOR operation.

The behaviour of the Figure 4-4 multilayer perceptron can also be related to the pattern space of Figure 4-2B. The lower cut in that pattern space is provided by the output unit. The upper cut in that pattern space is provided by the hidden unit. The coordination of the two units permits the circuit to solve this linearly nonseparable problem.

Interpreting networks in terms of the manner in which they carve a pattern space into decision regions suggests that learning can be described as determining where cuts in a pattern space should be made. Any hidden or output unit that uses a nonlinear, monotonic function like the Heaviside or the logistic can be viewed as making a single cut in a space. The position and orientation of this cut is determined by the weights of the connections feeding into the unit, as well as the threshold or bias (\(\theta\)) of the unit. A learning rule modifies all of these components. (The bias of a unit can be trained as if it were just another connection weight by assuming that it is the signal coming from a special, extra input unit that is always turned on [Dawson, 2004, 2005].)
The multilayer network illustrated in Figure 4-4 is atypical because it directly connects input and output units. Most modern networks eliminate such direct connections by using at least one layer of hidden units to isolate the input units from the output units, as shown in Figure 4-5. In such a network, the hidden units can still be described as carving a pattern space, with point coordinates provided by the input units, into a decision region. However, because the output units do not have direct access to input signals, they do not carve the pattern space. Instead, they divide an alternate space, the hidden unit space, into decision regions. The hidden unit space is similar to the pattern space, with the exception that the coordinates of the points that are placed within it are provided by hidden unit activities.

![Diagram](https://via.placeholder.com/150)

**Figure 4-5.** A typical multilayer perceptron has no direct connections between input and output units.

When there are no direct connections between input and output units, the hidden units provide output units with an internal representation of input unit activity. Thus it is proper to describe a network like the one illustrated in Figure 4-5 as being just as representational (Horgan & Tienson, 1996) as a classical model. That connectionist representations can be described as a nonlinear transformation of the input unit representation, permitting higher-order nonlinear features to be detected, is why a network like the one in Figure 4-5 is far more powerful than one in which no hidden units appear (e.g., Figure 4-3).

When there are no direct connections between input and output units, the representations held by hidden units conform to the classical sandwich that characterized
classical models (Hurley, 2001)—a connectionist sandwich (Calvo & Gomila, 2008, p. 5): “Cognitive sandwiches need not be Fodorian. A feed forward connectionist network conforms equally to the sandwich metaphor. The input layer is identified with a perception module, the output layer with an action one, and hidden space serves to identify metrically, in terms of the distance relations among patterns of activation, the structural relations that obtain among concepts. The hidden layer this time contains the meat of the connectionist sandwich.”

A difference between classical and connectionist cognitive science is not that the former is representational and the latter is not. Both are representational, but they disagree about the nature of mental representations. “The major lesson of neural network research, I believe, has been to thus expand our vision of the ways a physical system like the brain might encode and exploit information and knowledge” (Clark, 1997, p. 58).

4.5 Connectionist Computations: An Overview

In the preceding sections some of the basic characteristics of connectionist networks were presented. These elements of connectionist cognitive science have emerged as a reaction against key assumptions of classical cognitive science. Connectionist cognitive scientists replace rationalism with empiricism, and recursion with chains of associations.

Although connectionism reacts against many of the elements of classical cognitive science, there are many similarities between the two. In particular, the multiple levels of analysis described in Chapter 2 apply to connectionist cognitive science just as well as they do to classical cognitive science (Dawson, 1998). The next two sections of this chapter focus on connectionist research in terms of one of these, the computational level of investigation.

Connectionism’s emphasis on both empiricism and associationism has raised the spectre, at least in the eyes of many classical cognitive scientists, of a return to the behaviourism that cognitivism itself revolted against. When cognitivism arose, some of its early successes involved formal proofs that behaviourist and associationist theories were incapable of accounting for fundamental properties of human languages (Bever, Fodor, & Garrett, 1968; Chomsky, 1957, 1959b, 1965, 1966). With the rise of modern connectionism, similar computational arguments have been made against artificial neural networks, essentially claiming that they are not sophisticated enough to belong to the class of universal machines (Fodor & Pylyshyn, 1988).

In Section 4.6, “Beyond the Terminal Meta-postulate,” we consider the in-principle power of connectionist networks, beginning with two different types of tasks that networks can be used to accomplish. One is pattern classification: assigning an
input pattern in an all-or-none fashion to a particular category. A second is function approximation: generating a continuous response to a set of input values.

Section 4.6 then proceeds to computational analyses of how capable networks are of accomplishing these tasks. These analyses prove that networks are as powerful as need be, provided that they include hidden units. They can serve as arbitrary pattern classifiers, meaning that they can solve any pattern classification problem with which they are faced. They can also serve as universal function approximators, meaning that they can fit any continuous function to an arbitrary degree of precision. This computational power suggests that artificial neural networks belong to the class of universal machines. The section ends with a brief review of computational analyses, which conclude that connectionist networks indeed can serve as universal Turing machines and are therefore computationally sophisticated enough to serve as plausible models for cognitive science.

Computational analyses need not limit themselves to considering the general power of artificial neural networks. Computational analyses can be used to explore more specific questions about networks. This is illustrated in Section 4.7, “What Do Output Unit Activities Represent?” in which we use formal methods to answer the question that serves as the section’s title. The section begins with a general discussion of theories that view biological agents as intuitive statisticians who infer the probability that certain events may occur in the world (Peterson & Beach, 1967; Rescorla, 1967, 1968). An empirical result is reviewed that suggests artificial neural networks are also intuitive statisticians, in the sense that the activity of an output unit matches the probability that a network will be “rewarded” (i.e., trained to turn on) when presented with a particular set of cues (Dawson et al., 2009).

The section then ends by providing an example computational analysis: a formal proof that output unit activity can indeed literally be interpreted as a conditional probability. This proof takes advantage of known formal relations between neural networks and the Rescorla-Wagner learning rule (Dawson, 2008; Gluck & Bower, 1988; Sutton & Barto, 1981), as well as known formal relations between the Rescorla-Wagner learning rule and contingency theory (Chapman & Robbins, 1990).

4.6 Beyond the Terminal Meta-postulate

Connectionist networks are associationist devices that map inputs to outputs, systems that convert stimuli into responses. However, we saw in Chapter 3 that classical cognitive scientists had established that the stimulus-response theories of behaviourist psychology could not adequately deal with the recursive structure of natural language (Chomsky, 1957, 1959b, 1965, 1966). In the terminal meta-postulate argument (Bever, Fodor, and Garrett, 1968), it was noted that the rules of associative theory defined a “terminal vocabulary of a theory, i.e., over the vocabulary in which
behavior is described” (p. 583). Bever, Fodor, and Garrett then proceeded to prove that the terminal vocabulary of associationism is not powerful enough to accept or reject languages that have recursive clausal structure.

If connectionist cognitive science is another instance of associative or behaviourist theory, then it stands to reason that it too is subject to these same problems and therefore lacks the computational power required of cognitive theory. One of the most influential criticisms of connectionism has essentially made this point, arguing against the computational power of artificial neural networks because they lack the componentiality and systematicity associated with recursive rules that operate on components of symbolic expressions (Fodor & Pylyshyn, 1988). If artificial neural networks do not belong to the class of universal machines, then they cannot compete against the physical symbol systems that define classical cognitive science (Newell, 1980; Newell & Simon, 1976).

What tasks can artificial neural networks perform, and how well can they perform them? To begin, let us consider the most frequent kind of problem that artificial neural networks are used to solve: pattern recognition (Pao, 1989; Ripley, 1996). Pattern recognition is a process by which varying input patterns, defined by sets of features which may have continuous values, are assigned to discrete categories in an all-or-none fashion (Harnad, 1987). In other words, it requires that a system perform a mapping from continuous inputs to discrete outputs. Artificial neural networks are clearly capable of performing this kind of mapping, provided either that their output units use a binary activation function like the Heaviside, or that their continuous output is extreme enough to be given a binary interpretation. In this context, the pattern of “on” and “off” responses in a set of output units represents the digital name of the class to which an input pattern has been assigned.

We saw earlier that pattern recognition problems can be represented using pattern spaces (Figure 4.2). To classify patterns, a system carves a pattern space into decision regions that separate all of the patterns belonging to one class from the patterns that belong to others. An arbitrary pattern classifier would be a system that could, in principle, solve any pattern recognition problem with which it was faced. In order to have such ability, such a system must have complete flexibility in carving a pattern space into decision regions: it must be able to slice the space into regions of any required shape or number.

Artificial neural networks can categorize patterns. How well can they do so? It has been shown that a multilayer perceptron with three layers of connections—two layers of hidden units intervening between the input and output layers—is indeed an arbitrary pattern classifier (Lippmann, 1987, 1989). This is because the two layers of hidden units provided the required flexibility in carving pattern spaces into decision regions, assuming that the hidden units use a sigmoid-shaped activation
function such as the logistic. “No more than three layers are required in perceptron-like feed-forward nets” (Lippmann, 1987, p. 16).

When output unit activity is interpreted digitally—as delivering “true” or “false” judgments—artificial neural networks can be interpreted as performing one kind of task, pattern classification. However, modern networks use continuous activation functions that do not need to be interpreted digitally. If one applies an analog interpretation to output unit activity, then networks can be interpreted as performing a second kind of input-output mapping task, function approximation.

In function approximation, an input is a set of numbers that represents the values of variables passed into a function, i.e., the values of the set $x_1, x_2, x_3, \ldots x_N$. The output is a single value $y$ that is the result of computing some function of those variables, i.e., $y = f(x_1, x_2, x_3, \ldots x_N)$. Many artificial neural networks have been trained to approximate functions (Girosi & Poggio, 1990; Hartman, Keeler, & Kowalski, 1989; Moody & Darken, 1989; Poggio & Girosi, 1990; Renals, 1989). In these networks, the value of each input variable is represented by the activity of an input unit, and the continuous value of an output unit’s activity represents the computed value of the function of those input variables.

A system that is most powerful at approximating functions is called a universal function approximator. Consider taking any continuous function and examining a region of this function from a particular starting point (e.g., one set of input values) to a particular ending point (e.g., a different set of input values). A universal function approximator is capable of approximating the shape of the function between these bounds to an arbitrary degree of accuracy.

Artificial neural networks can approximate functions. How well can they do so? A number of proofs have shown that a multilayer perceptron with two layers of connections—in other words, a single layer of hidden units intervening between the input and output layers—is capable of universal function approximation (Cotter, 1990; Cybenko, 1989; Funahashi, 1989; Hartman, Keeler, & Kowalski, 1989; Hornik, Stinchcombe, & White, 1989). “If we have the right connections from the input units to a large enough set of hidden units, we can always find a representation that will perform any mapping from input to output” (Rumelhart, Hinton, & Williams, 1986a, p. 319).

That multilayered networks have the in-principle power to be arbitrary pattern classifiers or universal function approximators suggests that they belong to the class “universal machine,” the same class to which physical symbol systems belong (Newell, 1980). Newell (1980) proved that physical symbol systems belonged to this class by showing how a universal Turing machine could be simulated by a physical symbol system. Similar proofs exist for artificial neural networks, firmly establishing their computational power.
The Turing equivalence of connectionist networks has long been established. McCulloch and Pitts (1943) proved that a network of McCulloch-Pitts neurons could be used to build the machine head of a universal Turing machine; universal power was then achieved by providing this system with an external memory. “To psychology, however defined, specification of the net would contribute all that could be achieved in that field” (p. 131). More modern results have used the analog nature of modern processors to internalize the memory, indicating that an artificial neural network can simulate the entire Turing machine (Siegelmann, 1999; Siegelmann & Sontag, 1991, 1995).

Modern associationist psychologists have been concerned about the implications of the terminal meta-postulate and have argued against it in an attempt to free their theories from its computational shackles (Anderson & Bower, 1973; Paivio, 1986). The hidden units of modern artificial neural networks break these shackles by capturing higher-order associations—associations between associations—that are not defined in a vocabulary restricted to input and output activities. The presence of hidden units provides enough power to modern networks to firmly plant them in the class “universal machine” and to make them viable alternatives to classical simulations.

4.7 What Do Output Unit Activities Represent?

When McCulloch and Pitts (1943) formalized the information processing of neurons, they did so by exploiting the all-or-none law. As a result, whether a neuron responded could be interpreted as assigning a “true” or “false” value to some proposition computed over the neuron’s outputs. McCulloch and Pitts were able to design artificial neurons capable of acting as 14 of the 16 possible primitive functions on the two-valued logic that was described in Chapter 2.

McCulloch and Pitts (1943) formalized the all-or-none law by using the Heaviside step equation as the activation function for their artificial neurons. Modern activation functions such as the logistic equation provide a continuous approximation of the step function. It is also quite common to interpret the logistic function in digital, step function terms. This is done by interpreting a modern unit as being “on” or “off” if its activity is sufficiently extreme. For instance, in simulations conducted with my laboratory software (Dawson, 2005) it is typical to view a unit as being “on” if its activity is 0.9 or higher, or “off” if its activity is 0.1 or lower.

Digital activation functions, or digital interpretations of continuous activation functions, mean that pattern recognition is a primary task for artificial neural networks (Pao, 1989; Ripley, 1996). When a network performs pattern recognition, it is trained to generate a digital or binary response to an input pattern, where this
response is interpreted as representing a class to which the input pattern is unambiguously assigned.

What does the activity of a unit in a connectionist network mean? Under the strict digital interpretation described above, activity is interpreted as the truth value of some proposition represented by the unit. However, modern activation functions such as the logistic or Gaussian equations have continuous values, which permit more flexible kinds of interpretation. Continuous activity might model the frequency with which a real unit (i.e., a neuron) generates action potentials. It could represent a degree of confidence in asserting that a detected feature is present, or it could represent the amount of a feature that is present (Waskan & Bechtel, 1997).

In this section, a computational-level analysis is used to prove that, in the context of modern learning theory, continuous unit activity can be unambiguously interpreted as a candidate measure of degree of confidence with conditional probability (Waskan & Bechtel, 1997).

In experimental psychology, some learning theories are motivated by the ambiguous or noisy nature of the world. Cues in the real world do not signal outcomes with complete certainty (Dewey, 1929). It has been argued that adaptive systems deal with worldly uncertainty by becoming “intuitive statisticians,” whether these systems are humans (Peterson & Beach, 1967) or animals (Gallistel, 1990; Shanks, 1995). An agent that behaves like an intuitive statistician detects contingency in the world, because cues signal the likelihood (and not the certainty) that certain events (such as being rewarded) will occur (Rescorla, 1967, 1968).

Evidence indicates that a variety of organisms are intuitive statisticians. For example, the matching law is a mathematical formalism that was originally used to explain variations in response frequency. It states that the rate of a response reflects the rate of its obtained reinforcement. For instance, if response A is reinforced twice as frequently as response B, then A will appear twice as frequently as B (Herrnstein, 1961). The matching law also predicts how response strength varies with reinforcement frequency (de Villiers & Herrnstein, 1976). Many results show that the matching law governs numerous tasks in psychology and economics (Davison & McCarthy, 1988; de Villiers, 1977; Herrnstein, 1997).

Another phenomenon that is formally related (Herrnstein & Loveland, 1975) to the matching law is probability matching, which concerns choices made by agents faced with competing alternatives. Under probability matching, the likelihood that an agent makes a choice amongst different alternatives mirrors the probability associated with the outcome or reward of that choice (Vulkan, 2000). Probability matching has been demonstrated in a variety of organisms, including insects (Fischer, Couvillon, & Bitterman, 1993; Keasar et al., 2002; Longo, 1964; Niv et al., 2002), fish (Behrend & Bitterman, 1961), turtles (Kirk & Bitterman, 1965), pigeons (Graf, Bullock, & Bitterman, 1964), and humans (Estes & Straughan, 1954).
Perceptrons, too, can match probabilities (Dawson et al., 2009). Dawson et al. used four different cues, or discriminative stimuli (DSs), but did not “reward” them 100 percent of the time. Instead, they rewarded one DS 20 percent of the time, another 40 percent, a third 60 percent, and a fourth 80 percent. After 300 epochs, where each epoch involved presenting each cue alone 10 different times in random order, these contingencies were inverted (i.e., subtracted from 100). The dependent measure was perceptron activity when a cue was presented; the activation function employed was the logistic. Some results of this experiment are presented in Figure 4-6. It shows that after a small number of epochs, the output unit activity becomes equal to the probability that a presented cue was rewarded. It also shows that perceptron responses quickly readjust when contingencies are suddenly modified, as shown by the change in Figure 4-6 around epoch 300. In short, perceptrons are capable of probability matching.

![Figure 4-6.](image)


For instance, consider the simple situation in which a cue can either be presented, \(C\), or not, \(\neg C\). Associated with either of these states is an outcome (e.g., a reward) that can either occur, \(O\), or not, \(\neg O\). In this simple situation, involving a single cue and a single outcome, the contingency between the cue and the outcome is formally defined as the difference in conditional probabilities, \(\Delta P\), where \(\Delta P = P(O|C) - P(O|\neg C)\) (Allan, 1980). More sophisticated models, such as the probabilistic contrast model (e.g., Cheng & Novick, 1990) or the power PC theory (Cheng, 1997),
define more complex probabilistic contrasts that are possible when multiple cues occur and can be affected by the context in which they are presented.

Empirically, the probability matching of perceptrons, illustrated in Figure 4-6, suggests that their behaviour can represent $\Delta P$. When a cue is presented, activity is equal to the probability that the cue signals reinforcement—that is, $P(O|C)$. This implies that the difference between a perceptron's activity when a cue is presented and its activity when a cue is absent must be equal to $\Delta P$. Let us now turn to a computational analysis to prove this claim.

What is the formal relationship between formal contingency theories and theories of associative learning (Shanks, 2007)? Researchers have compared the predictions of an influential account of associative learning, the Rescorla-Wagner model (Rescorla & Wagner, 1972), to formal theories of contingency (Chapman & Robbins, 1990; Cheng, 1997; Cheng & Holyoak, 1995). It has been shown that while in some instances the Rescorla-Wagner model predicts the conditional contrasts defined by a formal contingency theory, in other situations it fails to generate these predictions (Cheng, 1997).

Comparisons between contingency learning and Rescorla-Wagner learning typically involve determining equilibria of the Rescorla-Wagner model. An equilibrium of the Rescorla-Wagner model is a set of associative strengths defined by the model, at the point where the asymptote of changes in error defined by Rescorla-Wagner learning approaches zero (Danks, 2003). In the simple case described earlier, involving a single cue and a single outcome, the Rescorla-Wagner model is identical to contingency theory. This is because at equilibrium, the associative strength between cue and outcome is exactly equal to $\Delta P$ (Chapman & Robbins, 1990).

There is also an established formal relationship between the Rescorla-Wagner model and the delta rule learning of a perceptron (Dawson, 2008; Gluck & Bower, 1988; Sutton & Barto, 1981). Thus by examining the equilibrium state of a perceptron facing a simple contingency problem, we can formally relate this kind of network to contingency theory and arrive at a formal understanding of what output unit activity represents.

When a continuous activation function is used in a perceptron, calculus can be used to determine the equilibrium of the perceptron. Let us do so for a single cue situation in which some cue, $C$, when presented, is rewarded a frequency of $a$ times, and is not rewarded a frequency of $b$ times. Similarly, when the cue is not presented, the perceptron is rewarded a frequency of $c$ times and is not rewarded a frequency of $d$ times. Note that to reward a perceptron is to train it to generate a desired response of 1, and that to not reward a perceptron is to train it to generate a desired response of 0, because the desired response indicates the presence or absence of the unconditioned stimulus (Dawson, 2008).
Assume that when the cue is present, the logistic activation function computes an activation value that we designate as $o_c$, and that when the cue is absent it returns the activation value designated as $o_{c-}$. We can now define the total error of responding for the perceptron, that is, its total error for the $(a + b + c + d)$ number of patterns that represent a single epoch, in which each instance of the contingency problem is presented once. For instance, on a trial in which $C$ is presented and the perceptron is reinforced, the perceptron’s error for that trial is the squared difference between the reward, 1, and $o_c$. As there are $a$ of these trials, the total contribution of this type of trial to overall error is $a(1 - o_c)^2$. Applying this logic to the other three pairings of cue and outcome, total error $E$ can be defined as follows:

$$E = a(1 - o_c)^2 + b(0 - o_c)^2 + c(1 - o_{c-})^2 + d(0 - o_{c-})^2$$

$$E = a(1 - o_c)^2 + b(o_c)^2 + c(1 - o_{c-})^2 + d(o_{c-})^2$$

For a perceptron to be at equilibrium, it must have reached a state in which total error has been optimized, so that the error can no longer be decreased by using the delta rule to alter the perceptron’s weight. To determine the equilibrium of the perceptron for the single cue contingency problem, we begin by taking the derivative of the error equation with respect to the activity of the perceptron when the cue is present, $o_c$:

$$\frac{\partial E}{\partial o_c} = 2(a(o_c - 1) + bo_c)$$

One condition of the perceptron at equilibrium is that $o_c$ is a value that causes this derivative to be equal to 0. The equation below sets the derivative to 0 and solves for $o_c$. The result is $a/(a + b)$, which is equal to the conditional probability $P(O|C)$ if the single cue experiment is represented with a traditional contingency table:

$$0 = 2(a(o_c - 1) + bo_c)$$

$$= a(o_c - 1) + bo_c$$

$$= ao_c - a + bo_c$$

$$a = o_c(a + b)$$

$$\frac{a}{a + b} = o_c$$

$$P(O|C) = o_c$$

Similarly, we can take the derivative of the error equation with respect to the activity of the perceptron when the cue is not present, $o_{c-}$:

$$\frac{\partial E}{\partial o_{c-}} = 2(c(o_{c-} - 1) + do_{c-})$$
A second condition of the perceptron at equilibrium is that $a_c$ is a value that causes the derivative above to be equal to 0. As before, we can set the derivative to 0 and solve for the value of $a_c$. This time the result is $c/(c+d)$, which in a traditional contingency table is equal to the conditional probability $P(O|\neg C)$:

$$
0 = 2(c(a_c - 1) + da_c)
= c(a_c - 1) + da_c
= c a_c - c + da_c
\frac{c}{c+d} = a_c
P(O|\neg C) = a_c
$$

The main implication of the above equations is that they show that perceptron activity is literally a conditional probability. This provides a computational proof for the empirical hypothesis about perceptron activity that was generated from examining Figure 4-6.

A second implication of the proof is that when faced with the same contingency problem, a perceptron’s equilibrium is not the same as that for the Rescorla-Wagner model. At equilibrium, the associative strength for the cue $C$ that is determined by Rescorla-Wagner training is literally $\Delta P$ (Chapman & Robbins, 1990). This is not the case for the perceptron. For the perceptron, $\Delta P$ must be computed by taking the difference between its output when the cue is present and its output when the cue is absent. That is, $\Delta P$ is not directly represented as a connection weight, but instead is the difference between perceptron behaviours under different cue situations—that is, the difference between the conditional probability output by the perceptron when a cue is present and the conditional probability output by the perceptron when the cue is absent.

Importantly, even though the perceptron and the Rescorla-Wagner model achieve different equilibria for the same problem, it is clear that both are sensitive to contingency when it is formally defined as $\Delta P$. Differences between the two reflect an issue that was raised in Chapter 2, that there exist many different possible algorithms for computing the same function. Key differences between the perceptron and the Rescorla-Wagner model—in particular, the fact that the former performs a nonlinear transformation on internal signals, while the latter does not—cause them to adopt very different structures, as indicated by different equilibria. Nonetheless, these very different systems are equally sensitive to exactly the same contingency.

This last observation has implications for the debate between contingency theory and associative learning (Cheng, 1997; Cheng & Holyoak, 1995; Shanks, 2007).
the current phase of this debate, modern contingency theories have been proposed as alternatives to Rescorla-Wagner learning. While in some instances equilibria for the Rescorla-Wagner model predict the conditional contrasts defined by a formal contingency theory like the power PC model, in other situations this is not the case (Cheng, 1997). However, the result above indicates that differences in equilibria do not necessarily reflect differences in system abilities. Clearly equilibrium differences cannot be used as the sole measure when different theories of contingency are compared.

4.8 Connectionist Algorithms: An Overview

In the last several sections we have explored connectionist cognitive science at the computational level of analysis. Claims about linear separability, the in-principle power of multilayer networks, and the interpretation of output unit activity have all been established using formal analyses.

In the next few sections we consider connectionist cognitive science from another perspective that it shares with classical cognitive science: the use of algorithmic-level investigations. The sections that follow explore how modern networks, which develop internal representations with hidden units, are trained, and also describe how one might interpret the internal representations of a network after it has learned to accomplish a task of interest. Such interpretations answer the question *How does a network convert an input pattern into an output response?* — and thus provide information about network algorithms.

The need for algorithmic-level investigations is introduced by noting in Section 4.9 that most modern connectionist networks are multilayered, meaning that they have at least one layer of hidden units lying between the input units and the output units. This section introduces a general technique for training such networks, called the generalized delta rule. This rule extends empiricism to systems that can have powerful internal representations.

Section 4.10 provides one example of how the internal representations created by the generalized delta rule can be interpreted. It describes the analysis of a multilayered network that has learned to classify different types of musical chords. An examination of the connection weights between the input units and the hidden units reveals a number of interesting ways in which this network represents musical regularities. An examination of the network’s hidden unit space shows how these musical regularities permit the network to rearrange different types of chord types so that they may then be carved into appropriate decision regions by the output units.

In section 4.11 a biologically inspired approach to discovering network algorithms is introduced. This approach involves wiretapping the responses of hidden units when the network is presented with various stimuli, and then using these
responses to determine the trigger features that the hidden units detect. It is also shown that changing the activation function of a hidden unit can lead to interesting complexities in defining the notion of a trigger feature, because some kinds of hidden units capture families of trigger features that require further analysis.

In Section 4.12 we describe how interpreting the internal structure of a network begins to shed light on the relationship between algorithms and architectures. Also described is a network that, as a result of training, translates a classical model of a task into a connectionist one. This illustrates an intertheoretic reduction between classical and connectionist theories, raising the possibility that both types of theories can be described in the same architecture.

4.9 Empiricism and Internal Representations

The ability of hidden units to increase the computational power of artificial neural networks was well known to Old Connectionism (McCulloch & Pitts, 1943). Its problem was that while a learning rule could be used to train networks with no hidden units (Rosenblatt, 1958, 1962), no such rule existed for multilayered networks. The reason that a learning rule did not exist for multilayered networks was because learning was defined in terms of minimizing the error of unit responses. While it was straightforward to define output unit error, no parallel definition existed for hidden unit error. A hidden unit’s error could not be defined because it was not related to any directly observable outcome (e.g., external behaviour). If a hidden unit’s error could not be defined, then Old Connectionist rules could not be used to modify its connections.

The need to define and compute hidden unit error is an example of the credit assignment problem:

In playing a complex game such as chess or checkers, or in writing a computer program, one has a definite success criterion—the game is won or lost. But in the course of play, each ultimate success (or failure) is associated with a vast number of internal decisions. If the run is successful, how can we assign credit for the success among the multitude of decisions? (Minsky, 1963, p. 432)

The credit assignment problem that faced Old Connectionism was the inability to assign the appropriate credit—or more to the point, the appropriate blame—to each hidden unit for its contribution to output unit error. Failure to solve this problem prevented Old Connectionism from discovering methods to make their most powerful networks belong to the domain of empiricism and led to its demise (Papert, 1988).

The rebirth of connectionist cognitive science in the 1980s (McClelland & Rumelhart, 1986; Rumelhart & McClelland, 1986c) was caused by the discovery...
of a solution to Old Connectionism’s credit assignment problem. By employing a nonlinear but continuous activation function, calculus could be used to explore changes in network behaviour (Rumelhart, Hinton, & Williams, 1986b). In particular, calculus could reveal how an overall network error was altered, by changing a component deep within the network, such as a single connection between an input unit and a hidden unit. This led to the discovery of the “backpropagation of error” learning rule, sometimes known as the generalized delta rule (Rumelhart, Hinton, & Williams, 1986b). The calculus underlying the generalized delta rule revealed that hidden unit error could be defined as the sum of weighted errors being sent backwards through the network from output units to hidden units.

The generalized delta rule is an error-correcting method for training multilayered networks that shares many characteristics with the original delta rule for perceptrons (Rosenblatt, 1958, 1962; Widrow, 1962; Widrow & Hoff, 1960). A more detailed mathematical treatment of this rule, and its relationship to other connectionist learning rules, is provided by Dawson (2004). A less technical account of the rule is given below.

The generalized delta rule is used to train a multilayer perceptron to mediate a desired input-output mapping. It is a form of supervised learning, in which a finite set of input-output pairs is presented iteratively, in random order, during training. Prior to training, a network is a “pretty blank” slate; all of its connection weights, and all of the biases of its activation functions, are initialized as small, random numbers. The generalized delta rule involves repeatedly presenting input-output pairs and then modifying weights. The purpose of weight modification is to reduce overall network error.

A single presentation of an input-output pair proceeds as follows. First, the input pattern is presented, which causes signals to be sent to hidden units, which in turn activate and send signals to the output units, which finally activate to represent the network’s response to the input pattern. Second, the output unit responses are compared to the desired responses, and an error term is computed for each output unit. Third, an output unit’s error is used to modify the weights of its connections. This is accomplished by adding a weight change to the existing weight. The weight change is computed by multiplying four different numbers together: a learning rate, the derivative of the unit’s activation function, the output unit’s error, and the current activity at the input end of the connection. Up to this point, learning is functionally the same as performing gradient descent training on a perceptron (Dawson, 2004).

The fourth step differentiates the generalized delta rule from older rules: each hidden unit computes its error. This is done by treating an output unit’s error as if it were activity and sending it backwards as a signal through a connection to a hidden unit. As this signal is sent, it is multiplied by the weight of the connection. Each
hidden unit computes its error by summing together all of the error signals that it receives from the output units to which it is connected. Fifth, once the hidden unit error has been computed, the weights of the hidden units can be modified using the same equation that was used to alter the weights of each of the output units.

This procedure can be repeated iteratively if there is more than one layer of hidden units. That is, the error of each hidden unit in one layer can be propagated backwards to an adjacent layer as an error signal once the hidden unit weights have been modified. Learning about this pattern stops once all of the connections have been modified. Then the next training pattern can be presented to the input units, and the learning process occurs again.

There are a variety of different ways in which the generic algorithm given above can be realized. For instance, in stochastic training, connection weights are updated after each pattern is presented (Dawson, 2004). This approach is called stochastic because each pattern is presented once per epoch of training, but the order of presentation is randomized for each epoch. Another approach, batch training, is to accumulate error over an epoch and to only update weights once at the end of the epoch, using accumulated error (Rumelhart, Hinton, & Williams, 1986a). As well, variations of the algorithm exist for different continuous activation functions. For instance, an elaborated error term is required to train units that have Gaussian activation functions, but when this is done, the underlying mathematics are essentially the same as in the original generalized delta rule (Dawson & Schopflocher, 1992b).

New Connectionism was born when the generalized delta rule was invented. Interestingly, the precise date of its birth and the names of its parents are not completely established. The algorithm was independently discovered more than once. Rumelhart, Hinton, and Williams (1986a, 1986b) are its most famous discoverers and popularizers. It was also discovered by David Parker in 1985 and by Yann LeCun in 1986 (Anderson, 1995). More than a decade earlier, the algorithm was reported in Paul Werbos’ (1974) doctoral thesis. The mathematical foundations of the generalized delta rule can be traced to an earlier decade, in a publication by Shun-Ichi Amari (1967).

In an interview (Anderson & Rosenfeld, 1998), neural network pioneer Stephen Grossberg stated that “Paul Werbos, David Parker, and Shun-Ichi Amari should have gotten credit for the backpropagation model, instead of Rumelhart, Hinton, and Williams” (pp. 179–180). Regardless of the credit assignment problem associated with the scientific history of this algorithm, it transformed cognitive science in the mid-1980s, demonstrating “how the lowly concepts of feedback and derivatives are the essential building blocks needed to understand and replicate higher-order phenomena like learning, emotion and intelligence at all levels of the human mind” (Werbos, 1994, p. 1).
4.10 Chord Classification by a Multilayer Perceptron

Artificial neural networks provide a medium in which to explore empiricism, for they acquire knowledge via experience. This knowledge is used to mediate an input-output mapping and usually takes the form of a distributed representation. Distributed representations provide some of the putative connectionist advantages over classical cognitive science: damage resistance, graceful degradation, and so on. Unfortunately, distributed representations are also tricky to interpret, making it difficult for them to provide new theories for cognitive science.

However, interpreting the internal structures of multilayered networks, though difficult, is not impossible. To illustrate this, let us consider a multilayer perceptron trained to classify different types of musical chords. The purpose of this section is to discuss the role of hidden units, to demonstrate that networks that use hidden units can also be interpreted, and to introduce a decidedly connectionist notion called the coarse code.

Chords are combinations of notes that are related to musical scales, where a scale is a sequence of notes that is subject to certain constraints. A chromatic scale is one in which every note played is one semitone higher than the previous note. If one were to play the first thirteen numbered piano keys of Figure 4-7 in order, then the result would be a chromatic scale that begins on a low C and ends on another C an octave higher.

![Figure 4-7. A small piano keyboard with numbered keys. Key 1 is C.](image)

A major scale results by constraining a chromatic scale such that some of its notes are not played. For instance, the C major scale is produced if only the white keys numbered from 1 to 13 in Figure 4-7 are played in sequence (i.e., if the black keys numbered 2, 4, 7, 9, and 11 are not played).

![Figure 4-8. The C major scale and some of its added note chords.](image)
The musical notation for the C major scale is provided in the sequence of notes illustrated in the first part of Figure 4-8. The Greeks defined a variety of modes for each scale; different modes were used to provoke different aesthetic experiences (Hanslick, 1957). The C major scale in the first staff of Figure 4-8 is in the Ionian mode because it begins on the note C, which is the root note, designated I, for the C major key.

One can define various musical chords in the context of C major in two different senses. First, the key signature of each chord is the same as C major (i.e., no sharps or flats). Second, each of these chords is built on the root of the C major scale (the note C). For instance, one basic chord is the major triad. In the key of C major, the root of this chord—the chord’s lowest note—is C (e.g., piano key #1 in Figure 4-7). The major triad for this key is completed by adding two other notes to this root. The second note in the triad is 4 semitones higher than C, which is the note E (the third note in the major scale in Figure 4-8). The third note in the triad is 3 semitones higher than the second note, which in this case is G (the fifth note in the major scale in Figure 4-8). Thus the notes C-E-G define the major triad for the key of C; this is the first chord illustrated in Figure 4-8.

A fourth note can added on to any major triad to create an “added note” tetrachord (Baker, 1982). The type of added note chord that is created depends upon the relationship between the added note and the third note of the major triad. If the added note is 4 semitones higher than the third note, the result is a major 7th chord, such as the Cmaj7 illustrated in Figure 4-8. If the added note is 3 semitones higher than the third note, the result is a dominant 7th chord such as the C7 chord presented in Figure 4-8. If the added note is 2 semitones higher than the third note, then the result is a 6th chord, such as the C6 chord illustrated in Figure 4-8.

The preceding paragraphs described the major triad and some added note chords for the key of C major. In Western music, C major is one of twelve possible major keys. The set of all possible major keys is provided in Figure 4-9, which organizes them in an important cyclic structure, called the circle of fifths.

![Figure 4-9. The circle of fifths.](image-url)
The circle of fifths includes all 12 notes in a chromatic scale, but arranges them so that adjacent notes in the circle are a musical interval of a perfect fifth (i.e., 7 semitones) apart. The circle of fifths is a standard topic for music students, and it is foundational to many concepts in music theory. It is provided here, though, to be contrasted later with “strange circles” that are revealed in the internal structure of a network trained to identify musical chords.

Any one of the notes in the circle of fifths can be used to define a musical key and therefore can serve as the root note of a major scale. Similarly, any one of these notes can be the root of a major triad created using the pattern of root + 4 semitones + 3 semitones that was described earlier for the key of C major (Baker, 1982). Furthermore, the rules described earlier can also be applied to produce added note chords for any of the 12 major key signatures. These possible major triads and added note chords were used as inputs for training a network to correctly classify different types of chords, ignoring musical key.

A training set of 48 chords was created by building the major triad, as well as the major 7th, dominant 7th, and 6th chord for each of the 12 possible major key signatures (i.e., using each of the notes in Figure 4-9 as a root). When presented with a chord, the network was trained to classify it into one of the four types of interest: major triad, major 7th, dominant 7th, or 6th. To do so, the network had 4 output units, one for each type of chord. For any input, the network learned to turn the correct output unit on and to turn the other three output units off.

The input chords were encoded with a pitch class representation (Laden & Keefe, 1989; Yaremchuk & Dawson, 2008). In a pitch class representation, only 12 input units are employed, one for each of the 12 different notes that can appear in a scale. Different versions of the same note (i.e., the same note played at different octaves) are all mapped onto the same input representation. For instance, notes 1, 13, 25, and 37 in Figure 4-7 all correspond to different pitches but belong to the same pitch class—they are all C notes, played at different octaves of the keyboard. In a pitch class representation, the playing of any of these input notes would be encoded by turning on a single input unit—the one unit used to represent the pitch class of C.

A pitch class representation of chords was used for two reasons. First, it requires a very small number of input units to represent all of the possible stimuli. Second, it is a fairly abstract representation that makes the chord classification task difficult, which in turn requires using hidden units in a network faced with this task.

Why chord classification might be difficult for a network when pitch class encoding is employed becomes evident by thinking about how we might approach the problem if faced with it ourselves. Classifying the major chords is simple: they are the only input stimuli that activate three input units instead of four. However, classifying the other chord types is very challenging. One first has to determine what key the stimulus is in, identify which three notes define its major chord component,
and then determine the relationship between the third note of the major chord component and the fourth “added” note. This is particularly difficult because of the pitch class representation, which throws away note-order information that might be useful in identifying chord type.

It was decided that the network that would be trained on the chord classification task would be a network of value units (Dawson & Schopflocher, 1992b). The hidden units and output units in a network of value units use a Gaussian activation function, which means that they behave as if they carve two parallel planes through a pattern space. Such networks can be trained with a variation of the generalized delta rule. This type of network was chosen for this problem for two reasons. First, networks of value units have emergent properties that make them easier to interpret than other types of networks trained on similar problems (Dawson, 2004; Dawson et al., 1994). One reason for this is because value units behave as if they are “tuned” to respond to very particular input signals. Second, previous research on different versions of chord classification problems had produced networks that revealed elegant internal structure (Yaremchuk & Dawson, 2005, 2008).

The simplest network of value units that could learn to solve the chord classification problem required three hidden units. At the start of training, the value of \( m \) for each unit was initialized as 0. (The value of \( m \) for a value unit is analogous to a threshold in other types of units [Dawson, Kremer, & Gannon, 1994; Dawson & Schopflocher, 1992b]; if a value unit’s net input is equal to \( m \) then the unit generates a maximum activity of 1.00.) All connection weights were set to values randomly selected from the range between –0.1 and 0.1. The network was trained with a learning rate of 0.01 until it produced a “hit” for every output unit on every pattern. Because of the continuous nature of the activation function, a hit was defined as follows: a value of 0.9 or higher when the desired output was 1, and a value of 0.1 or lower when the desired output was 0. The network that is interpreted below learned the chord classification task after 299 presentations of the training set.

What is the role of a layer of hidden units? In a perceptron, which has no hidden units, input patterns can only be represented in a pattern space. Recall from the discussion of Figure 4-2 that a pattern space represents each pattern as a point in space. The dimensionality of this space is equal to the number of input units. The coordinates of each pattern’s point in this space are given by the activities of the input units. For some networks, the positioning of the points in the pattern space prevents some patterns from being correctly classified, because the output units are unable to adequately carve the pattern space into the appropriate decision regions.

In a multilayer perceptron, the hidden units serve to solve this problem. They do so by transforming the pattern space into a hidden unit space (Dawson, 2004). The dimensionality of a hidden unit space is equal to the number of hidden units.
in the layer. Patterns are again represented as points in this space; however, in this space their coordinates are determined by the activities they produce in each hidden unit. The hidden unit space is a transformation of the pattern space that involves detecting higher-order features. This usually produces a change in dimensionality—the hidden unit space often has a different number of dimensions than does the pattern space—and a repositioning of the points in the new space. As a result, the output units are able to carve the hidden unit space into a set of decision regions that permit all of the patterns, repositioned in the hidden unit space, to be correctly classified.

This account of the role of hidden units indicates that the interpretation of the internal structure of a multilayer perceptron involves answering two different questions. First, what kinds of features are the hidden units detecting in order to map patterns from the pattern space into the hidden unit space? Second, how do the output units process the hidden unit space to solve the problem of interest? The chord classification network can be used to illustrate how both questions can be addressed.

First, when mapping the input patterns into the hidden unit space, the hidden units must be detecting some sorts of musical regularities. One clue as to what these regularities may be is provided by simply examining the connection weights that feed into them, provided in Table 4-2.

<table>
<thead>
<tr>
<th>Input Note</th>
<th>Hidden 1</th>
<th>Hidden 1 Class</th>
<th>Hidden 2</th>
<th>Hidden 2 Class</th>
<th>Hidden 3</th>
<th>Hidden 3 Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0.53</td>
<td>Circle of Major Thirds 1</td>
<td>0.12</td>
<td>Circle of Major Thirds 1</td>
<td>0.75</td>
<td>Circle of Major Seconds 1</td>
</tr>
<tr>
<td>D#</td>
<td>0.53</td>
<td>Circle of Major Thirds 1</td>
<td>0.12</td>
<td>Circle of Major Thirds 1</td>
<td>0.75</td>
<td>Circle of Major Seconds 1</td>
</tr>
<tr>
<td>G</td>
<td>0.53</td>
<td>Circle of Major Thirds 1</td>
<td>0.12</td>
<td>Circle of Major Thirds 1</td>
<td>0.75</td>
<td>Circle of Major Seconds 1</td>
</tr>
<tr>
<td>A</td>
<td>−0.53</td>
<td>Circle of Major Thirds 2</td>
<td>−0.12</td>
<td>Circle of Major Thirds 2</td>
<td>0.75</td>
<td>Circle of Major Seconds 1</td>
</tr>
<tr>
<td>C#</td>
<td>−0.53</td>
<td>Circle of Major Thirds 2</td>
<td>−0.12</td>
<td>Circle of Major Thirds 2</td>
<td>0.75</td>
<td>Circle of Major Seconds 1</td>
</tr>
<tr>
<td>F</td>
<td>−0.53</td>
<td>Circle of Major Thirds 3</td>
<td>−0.53</td>
<td>Circle of Major Thirds 3</td>
<td>−0.77</td>
<td>Circle of Major Seconds 2</td>
</tr>
<tr>
<td>C</td>
<td>0.12</td>
<td>Circle of Major Thirds 3</td>
<td>−0.53</td>
<td>Circle of Major Thirds 3</td>
<td>−0.77</td>
<td>Circle of Major Seconds 2</td>
</tr>
<tr>
<td>G#</td>
<td>0.12</td>
<td>Circle of Major Thirds 3</td>
<td>−0.53</td>
<td>Circle of Major Thirds 3</td>
<td>−0.77</td>
<td>Circle of Major Seconds 2</td>
</tr>
<tr>
<td>E</td>
<td>0.12</td>
<td>Circle of Major Thirds 3</td>
<td>−0.53</td>
<td>Circle of Major Thirds 3</td>
<td>−0.77</td>
<td>Circle of Major Seconds 2</td>
</tr>
<tr>
<td>F#</td>
<td>−0.12</td>
<td>Circle of Major Thirds 4</td>
<td>0.53</td>
<td>Circle of Major Thirds 4</td>
<td>−0.77</td>
<td>Circle of Major Seconds 2</td>
</tr>
<tr>
<td>A#</td>
<td>−0.12</td>
<td>Circle of Major Thirds 4</td>
<td>0.53</td>
<td>Circle of Major Thirds 4</td>
<td>−0.77</td>
<td>Circle of Major Seconds 2</td>
</tr>
<tr>
<td>D</td>
<td>−0.12</td>
<td>Circle of Major Thirds 4</td>
<td>0.53</td>
<td>Circle of Major Thirds 4</td>
<td>−0.77</td>
<td>Circle of Major Seconds 2</td>
</tr>
</tbody>
</table>

Table 4-2. Connection weights from the 12 input units to each of the three hidden units. Note that the first two hidden units adopt weights that assign input notes to the four circles of major thirds. The third hidden unit adopts weights that assign input notes to the two circles of major seconds.
In the pitch class representation used for this network, each input unit stands for a distinct musical note. As far as the hidden units are concerned, the “name” of each note is provided by the connection weight between the input unit and the hidden unit. Interestingly, Table 4-2 reveals that all three hidden units take input notes that we would take as being different (because they have different names, as in the circle of fifths in Figure 4-9) and treat them as being identical. That is, the hidden units assign the same “name,” or connection weight, to input notes that we would give different names to.

Furthermore, assigning the same “name” to different notes by the hidden units is not done randomly. Notes are assigned according to strange circles, that is, circles of major thirds and circles of major seconds. Let us briefly describe these circles, and then return to an analysis of Table 4-2.

The circle of fifths (Figure 4-9) is not the only way in which notes can be arranged geometrically. One can produce other circular arrangements by exploiting other musical intervals. These are strange circles in the sense that they would very rarely be taught to music students as part of a music theory curriculum. However, these strange circles are formal devices that can be as easily defined as can be the circle of fifths.

For instance, if one starts with the note C and moves up a major second (2 semitones) then one arrives at the note D. From here, moving up another major second arrives at the note E. This can continue until one circles back to C but an octave higher than the original, which is a major second higher than A#. This circle of major seconds captures half of the notes in the chromatic scale, as is shown in the top part of Figure 4-10. A complementary circle of major seconds can also be constructed (bottom circle of Figure 4-10); this circle contains all the remaining notes that are not part of the first circle.

![Figure 4-10. The two circles of major seconds.](image)

An alternative set of musical circles can be defined by exploiting a different musical interval. In each circle depicted in Figure 4-11, adjacent notes are a major third (4 semitones) apart. As shown in Figure 4-11 four such circles are possible.
What do these strange circles have to do with the internal structure of the network trained to classify the different types of chords? A close examination of Table 4-2 indicates that these strange circles are reflected in the connection weights that feed into the network's hidden units. For Hidden Units 1 and 2, if notes belong to the same circle of major thirds (Figure 4-11), then they are assigned the same connection weight. For Hidden Unit 3, if notes belong to the same circle of major seconds (Figure 4-10), then they are assigned the same connection weight. In short, each of the hidden units replaces the 12 possible different note names with a much smaller set, which equates notes that belong to the same circle of intervals and differentiates notes that belong to different circles.

Further inspection of Table 4-2 reveals additional regularities of interest. Qualitatively, both Hidden Units 1 and 2 assign input notes to equivalence classes based on circles of major thirds. They do so by using the same note “names”: 0.53, 0.12, –0.12, and –0.53. However, the two hidden units have an important difference: they assign the same names to different sets of input notes. That is, notes that are assigned one connection weight by Hidden Unit 1 are assigned a different connection weight by Hidden Unit 2.

The reason that the difference in weight assignment between the two hidden units is important is that the behaviour of each hidden unit is not governed by a single incoming signal, but is instead governed by a combination of three or four
input signals coming from all of the units. The connection weights used by the hidden units place meaningful constraints on how these signals are combined.

Let us consider the role of the particular connection weights used by the hidden units. Given the binary nature of the input encoding, the net input of any hidden unit is simply the sum of the weights associated with each of the activated input units. For a value unit, if the net input is equal to the value of the unit’s $m$ then the output generates a maximum value of 1.00. As the net input moves away from $m$ in either a positive or negative direction, activity quickly decreases. At the end of training, the values of $m$ for the three hidden units were 0.00, 0.00, and $-0.03$ for Hidden Units 1, 2, and 3, respectively. Thus for each hidden unit, if the incoming signals are essentially zero—that is if all the incoming signals cancel each other out—then high activity will be produced.

Why then do Hidden Units 1 and 2 use the same set of four connection weights but assign these weights to different sets of input notes? The answer is that these hidden units capture similar chord relationships but do so using notes from different strange circles.

This is shown by examining the responses of each hidden unit to each input chord after training. Table 4-3 summarizes these responses, and shows that each hidden unit generated identical responses to different subsets of input chords.

<table>
<thead>
<tr>
<th>Chord</th>
<th>Input Chord</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Chord Root</td>
<td>Hid1</td>
</tr>
<tr>
<td>Major</td>
<td>C, D, A, F#, G#, A#</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>C#, D#, F, G, A, B</td>
<td>0.06</td>
</tr>
<tr>
<td>Major7</td>
<td>C, D, A, F#, G#, A#</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>C#, D#, F, G, A, B</td>
<td>0.12</td>
</tr>
<tr>
<td>Dom7</td>
<td>C, D, A, F#, G#, A#</td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td>C#, D#, F, G, A, B</td>
<td>0.59</td>
</tr>
<tr>
<td>6th</td>
<td>C, D, A, F#, G#, A#</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td>C#, D#, F, G, A, B</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Table 4-3. The activations produced in each hidden unit by different subsets of input chords.

From Table 4-3, one can see that the activity of Hidden Unit 3 is simplest to describe: when presented with a dominant 7th chord, it produces an activation of 0 and a weak activation to a major triad. When presented with either a major 7th or a 6th chord, it produces maximum activity. This pattern of activation is easily explained by considering the weights that feed into Hidden Unit 3 (Table 4-2). Any major 7th or 6th chord is created out of two notes from one circle of major seconds and two notes from the
other circle. The sums of pairs of weights from different circles cancel each other out, producing near-zero net input and causing maximum activation.

In contrast, the dominant 7th chords use three notes from one circle of major seconds and only one from the other circle. As a result, the signals do not cancel out completely, given the weights in Table 4-2. Instead, a strong non-zero net input is produced, and the result is zero activity.

Finally, any major triad involves only three notes: two from one circle of major seconds and one from the other. Because of the odd number of input signals, cancellation to zero is not possible. However, the weights have been selected so that the net input produced by a major triad is close enough to m to produce weak activity.

The activation patterns for Hidden Units 1 and 2 are more complex. It is possible to explain all of them in terms of balancing (or failing to balance) signals associated with different circles of major thirds. However, it is more enlightening to consider these two units at a more general level, focusing on the relationship between their activations.

In general terms, Hidden Units 1 and 2 generate activations of different intensities to different classes of chords. In general, they produce the highest activity to 6th chords and the lowest activity to major 7th chords. Importantly, they do not generate the same activity to all chords of the same type. For instance, for the 12 possible 6th chords, Hidden Unit 1 generates activity of 0.84 to 6 of them but activity of only 0.03 to the other 6 chords. An inspection of Table 4-3 indicates that for every chord type, both Hidden Units 1 and 2 generate one level of activity with half of them, but produce another level of activity with the other half.

The varied responses of these two hidden units to different chords of the same type are related to the circle of major seconds (Figure 4-10). For example, Hidden Unit 1 generates a response of 0.84 to 6th chords whose root note belongs to the top circle of Figure 4-10, and a response of 0.03 to 6th chords whose root note belongs to the bottom circle of Figure 4-10. Indeed, for all of the chord types, both of these hidden units generate one response if the root note belongs to one circle of major seconds and a different response if the root note belongs to the other circle.

Furthermore, the responses of Hidden Units 1 and 2 complement one another: for any chord type, those chords that produce low activity in Hidden Unit 1 produce higher activity in Hidden Unit 2. As well, those chords that produce low activity in Hidden Unit 2 produce higher activity in Hidden Unit 1. This complementing is again related to the circles of major seconds: Hidden Unit 1 generates higher responses to chords whose root belongs to one circle, while Hidden Unit 2 generates higher responses to chords whose roots belong to the other. Which circle is “preferred” by a hidden unit depends on chord type.

Clearly each of the three hidden units is sensitive to musical properties. However, it is not clear how these properties support the network’s ability to classify
chords. For instance, none of the hidden units by themselves pick out a set of properties that uniquely define a particular type of chord. Instead, hidden units generate some activity to different chord types, suggesting the existence of a coarse code.

In order to see how the activities of the hidden units serve as a distributed representation that mediates chord classification, we must examine the hidden unit space. The hidden unit space plots each input pattern as a point in a space whose dimensionality is determined by the number of hidden units. The coordinates of the point in the hidden unit space are the activities produced by an input pattern in each hidden unit. The three-dimensional hidden unit space for the chord classification network is illustrated in Figure 4-12.

![Figure 4-12](image.png)

**Figure 4-12.** The hidden unit space for the chord classification network. H1, H2, and H3 provide the activity of hidden units 1, 2, and 3 respectively.

Because the hidden units generate identical responses to many of the chords, instead of 48 different visible points in this graph (one for each input pattern), there are only 8. Each point represents 6 different chords that fall in exactly the same location in the hidden unit space.

The hidden unit space reveals that each chord type is represented by two different points. That these points capture the same class is represented in Figure 4-12 by joining a chord type’s points with a dashed line. Two points are involved in defining a chord class in this space because, as already discussed, each hidden unit is sensitive to the organization of notes according to the two circles of major seconds. For each chord type, chords whose root belongs to one of these circles are mapped
to one point, and chords whose root belongs to the other are mapped to the other point. Interestingly, there is no systematic relationship in the graph that maps onto the two circles. For instance, it is not the case that the four points toward the back of the Figure 4-12 cube all map onto the same circle of major seconds.

Figure 4-13 illustrates how the output units can partition the points in the hidden unit space in order to classify chords. Each output unit in this network is a value unit, which carves two parallel hyperplanes through a pattern space. To solve the chord classification problem, the connection weights and the bias of each output unit must take on values that permit these two planes to isolate the two points associated with one chord type from all of the other points in the space. Figure 4-13 shows how this would be accomplished by the output unit that signals that a 6th chord has been detected.

4.11 Trigger Features

For more than half a century, neuroscientists have studied vision by mapping the receptive fields of individual neurons (Hubel & Wiesel, 1959; Lettvin, Maturana, McCulloch, & Pitts, 1959). To do this, they use a method called microelectrode
recording or wiretapping (Calvin & Ojemann, 1994), in which the responses of single neurons are measured while stimuli are being presented to an animal. With this technique, it is possible to describe a neuron as being sensitive to a trigger feature, a specific pattern that when detected produces maximum activity in the cell.

That individual neurons may be described as detecting trigger features has led some to endorse a neuron doctrine for perceptual psychology. This doctrine has the goal of discovering the trigger features for all neurons (Barlow, 1972, 1995). This is because,

\[
\text{a description of that activity of a single nerve cell which is transmitted to and influences other nerve cells, and of a nerve cell's response to such influences from other cells, is a complete enough description for functional understanding of the nervous system. (Barlow, 1972, p. 380)}
\]

The validity of the neuron doctrine is a controversial issue (Bowers, 2009; Gross, 2002). Regardless, there is a possibility that identifying trigger features can help to interpret the internal workings of artificial neural networks.

For some types of hidden units, trigger features can be identified analytically, without requiring any wiretapping of hidden unit activities (Dawson, 2004). For instance, the activation function for an integration device (e.g., the logistic equation) is monotonic, which means that increases in net input always produce increases in activity. As a result, if one knows the maximum and minimum possible values for input signals, then one can define an integration device's trigger feature simply by inspecting the connection weights that feed into it (Dawson, Kremer, & Gannon, 1994). The trigger feature is that pattern which sends the minimum signal through every inhibitory connection and the maximum signal through every excitatory connection. The monotonicity of an integration device's activation function ensures that it will have only one trigger feature.

The notion of a trigger feature for other kinds of hidden units is more complex. Consider a value unit whose bias, \( m \), in its Gaussian activation function is equal to 0. The trigger feature for this unit will be the feature that causes it to produce maximum activation. For this value unit, this will occur when the net input to the unit is equal to 0 (i.e., equal to the value of \( m \)) (Dawson & Schopflocher, 1992b). The net input of a value unit is defined by a particular linear algebra operation, called the inner product, between a vector that represents a stimulus and a vector that represents the connection weights that fan into the unit (Dawson, 2004). So, when net input equals 0, this means that the inner product is equal to 0.

However, when an inner product is equal to 0, this indicates that the two vectors being combined are orthogonal to one another (that is, there is an angle of 90° between the two vectors). Geometrically speaking, then, the trigger feature for a value unit is an input pattern represented by a vector of activities that is at a right angle to the vector of connection weights.
This geometric observation raises complications, because it implies that a hidden value unit will *not* have a single trigger feature. This is because there are many input patterns that are orthogonal to a vector of connection weights. *Any* input vector that lies in the hyperplane that is perpendicular to the vector of connection weights will serve as a trigger feature for the hidden value unit (Dawson, 2004); this is illustrated in Figure 4-14.

Another consequence of the geometric account provided above is that there should be families of other input patterns that share the property of producing the same hidden unit activity, but one that is lower than the maximum activity produced by one of the trigger features. These will be patterns that all fall into the same hyperplane, but this hyperplane is *not* orthogonal to the vector of connection weights.

![Figure 4-14. Any input pattern (dashed lines) whose vector falls in the plane orthogonal to the vector of connection weights (solid line) will be a trigger feature for a hidden value unit.](image)

The upshot of all of this is that if one trains a network of value units and then wiretaps its hidden units, the resulting hidden unit activities should be highly organized. Instead of having a rectangular distribution of activation values, there should be regular groups of activations, where each group is related to a different family of input patterns (i.e., families related to different hyperplanes of input patterns).

Empirical support for this analysis was provided by the discovery of activity...
banding when a hidden unit’s activities were plotted using a jittered density plot (Berkeley et al., 1995). A jittered density plot is a two-dimensional scatterplot of points; one such plot can be created for each hidden unit in a network. Each plotted point represents one of the patterns presented to the hidden unit during wiretapping. The $x$-value of the point’s position in the graph is the activity produced in that hidden unit by the pattern. The $y$-value of the point’s position in the scatterplot is a random value that is assigned to reduce overlap between points.

An example of a jittered density plot for a hidden value unit is provided in Figure 4-15. Note that the points in this plot are organized into distinct bands, which is consistent with the geometric analysis. This particular unit belongs to a network of value units trained on a logic problem discussed in slightly more detail below (Bechtel & Abrahamsen, 1991), and was part of a study that examined some of the implications of activity banding (Dawson & Piercey, 2001).

![Figure 4-15](image)

**Figure 4-15.** An example of banding in a jittered density plot of a hidden value unit in a network that was trained on a logic problem.

Bands in jittered density plots of hidden value units can be used to reveal the kinds of features that are being detected by these units. For instance, Berkeley et al. (1995) reported that all of the patterns that fell into the same band on a single jittered density plot in the networks did so because they shared certain local properties or features, which are called definite features.

There are two types of definite features. The first is called a definite unary feature. When a definite unary feature exists, it means that a single feature has the same value for every pattern in the band. The second is called a definite binary feature. With this kind of definite feature, an individual feature is not constant within
a band. However, its relationship to some other feature is constant—variations in one feature are perfectly correlated with variations in another. Berkeley et al. (1995) showed how definite features could be both objectively defined and easily discovered using simple descriptive statistics (see also Dawson, 2005).

Definite features are always expressed in terms of the values of input unit activities. As a result, they can be assigned meanings using knowledge of a network’s input unit encoding scheme.

One example of using this approach was presented in Berkeley et al.’s (1995) analysis of a network on the Bechtel and Abrahamsen (1991) logic task. This task consists of a set of 576 logical syllogisms, each of which can be expressed as a pattern of binary activities using 14 input units. Each problem is represented as a first sentence that uses two variables, a connective or a second sentence that states a variable, and a conclusion that states a variable. Four different problem types were created in this format: modus ponens, modus tollens, disjunctive syllogism, and alternative syllogism. Each problem type was created using one of three different connectives and four different variables: the connectives were If…then, Or, or Not Both…And; the variables were A, B, C, and D. An example of a valid modus ponens argument in this format is “Sentence 1: ‘If A then B’; Sentence 2: ‘A’; Conclusion: ‘B’.”

For this problem, a network’s task is to classify an input problem into one of the four types and to classify it as being either a valid or an invalid example of that problem type. Berkeley et al. (1995) successfully trained a network of value units that employed 10 hidden units. After training, each of these units were wiretapped using the entire training set as stimulus patterns, and a jittered density plot was produced for each hidden unit. All but one of these plots revealed distinct banding. Berkeley et al. were able to provide a very detailed set of definite features for each of the bands.

After assigning definite features, Berkeley et al. (1995) used them to explore how the internal structure of the network was responsible for making the correct logical judgments. They expressed input logic problems in terms of which band of activity they belonged to for each jittered density plot. They then described each pattern as the combination of definite features from each of these bands, and they found that the internal structure of the network represented rules that were very classical in nature.

For example, Berkeley et al. (1995) found that every valid modus ponens problem was represented as the following features: having the connective If…then, having the first variable in Sentence 1 identical to Sentence 2, and having the second variable in Sentence 1 identical to the Conclusion. This is essentially the rule for valid modus ponens that could be taught in an introductory logic class (Bergmann, Moor, & Nelson, 1990). Berkeley et al. found several such rules; they also found a number that were not so traditional, but which could still be expressed in a
classical form. This result suggests that artificial neural networks might be more symbolic in nature than connectionist cognitive scientists care to admit (Dawson, Medler, & Berkeley, 1997).

Importantly, the Berkeley et al. (1995) analysis was successful because the definite features that they identified were local. That is, by examining a single band in a single jittered density plot, one could determine a semantically interpretable set of features. However, activity bands are not always local. In some instances hidden value units produce nicely banded jittered density plots that possess definite features, but these features are difficult to interpret semantically (Dawson & Piercey, 2001). This occurs when the semantic interpretation is itself distributed across different bands for different hidden units; an interpretation of such a network requires definite features from multiple bands to be considered in concert.

While the geometric argument provided earlier motivated a search for the existence of bands in the hidden units of value unit networks, banding has been observed in networks of integration devices as well (Berkeley & Gunay, 2004). That being said, banding is not seen in every value unit network either. The existence of banding is likely an interaction between network architecture and problem representation; banding is useful when discovered, but it is only one tool available for network interpretation.

The important point is that practical tools exist for interpreting the internal structure of connectionist networks. Many of the technical issues concerning the relationship between classical and connectionist cognitive science may hinge upon network interpretations: “In our view, questions like ‘What is a classical rule?’ and ‘Can connectionist networks be classical in nature?’ are also hopelessly unconstrained. Detailed analyses of the internal structure of particular connectionist networks provide a specific framework in which these questions can be fruitfully pursued” (Dawson, Medler, & Berkeley, 1997, p. 39).

4.12 A Parallel Distributed Production System

One of the prototypical architectures for classical cognitive science is the production system (Anderson, 1983; Kieras & Meyer, 1997; Meyer et al., 2001; Meyer & Kieras, 1977a, 1977b; Newell, 1973, 1990; Newell & Simon, 1972). A production system is a set of condition-action pairs. Each production works in parallel, scanning working memory for a pattern that matches its condition. If a production finds such a match, then it takes control, momentarily disabling the other productions, and performs its action, which typically involves adding, deleting, copying, or moving symbols in the working memory.

Production systems have been proposed as a lingua franca for cognitive science, capable of describing any connectionist or embodied cognitive science theory and
therefore of subsuming such theories under the umbrella of classical cognitive science (Vera & Simon, 1993). This is because Vera and Simon (1993) argued that any situation-action pairing can be represented either as a single production in a production system or, for complicated situations, as a set of productions. “Productions provide an essentially neutral language for describing the linkages between information and action at any desired (sufficiently high) level of aggregation” (p. 42). Other philosophers of cognitive science have endorsed similar positions. For instance, von Eckardt (1995) suggested that if one considers distributed representations in artificial neural networks as being “higher-level” representations, then connectionist networks can be viewed as being analogous to classical architectures. This is because when examined at this level, connectionist networks have the capacity to input and output represented information, to store represented information, and to manipulate represented information. In other words, the symbolic properties of classical architectures may emerge from what are known as the subsymbolic properties of networks (Smolensky, 1988).

However, the view that artificial neural networks are classical in general or examples of production systems in particular is not accepted by all connectionists. It has been claimed that connectionism represents a Kuhnian paradigm shift away from classical cognitive science (Schneider, 1987). With respect to Vera and Simon’s (1993) particular analysis, their definition of symbol has been deemed too liberal by some neural network researchers (Touretzky & Pomerleau, 1994). Touretzky and Pomerleau (1994) claimed of a particular neural network discussed by Vera and Simon, ALVINN (Pomerleau, 1991), that its hidden unit “patterns are not arbitrarily shaped symbols, and they are not combinatorial. Its hidden unit feature detectors are tuned filters” (Touretzky & Pomerleau, 1994, p. 348). Others have viewed ALVINN from a position of compromise, noting that “some of the processes are symbolic and some are not” (Greeno & Moore, 1993, p. 54).

Are artificial neural networks equivalent to production systems? In the philosophy of science, if two apparently different theories are in fact identical, then one theory can be translated into the other. This is called intertheoretic reduction (Churchland, 1985, 1988; Hooker, 1979, 1981). The widely accepted view that classical and connectionist cognitive science are fundamentally different (Schneider, 1987) amounts to the claim that intertheoretic reduction between a symbolic model and a connectionist network is impossible. One research project (Dawson et al., 2000) directly examined this issue by investigating whether a production system model could be translated into an artificial neural network.

Dawson et al. (2000) investigated intertheoretic reduction using a benchmark problem in the machine learning literature, classifying a very large number (8,124) of mushrooms as being either edible or poisonous on the basis of 21 different features (Schlimmer, 1987). Dawson et al. (2000) used a standard machine learning
technique, the ID₃ algorithm (Quinlan, 1986) to induce a decision tree for the mushroom problem. A decision tree is a set of tests that are performed in sequence to classify patterns. After performing a test, one either reaches a terminal branch of the tree, at which point the pattern being tested can be classified, or a node of the decision tree, which is to say another test that must be performed. The decision tree is complete for a pattern set if every pattern eventually leads the user to a terminal branch. Dawson et al. (2000) discovered that a decision tree consisting of only five different tests could solve the Schlimmer mushroom classification task. Their decision tree is provided in Table 4-4.

<table>
<thead>
<tr>
<th>Step</th>
<th>Tests and Decision Points</th>
</tr>
</thead>
</table>
| 1    | *What is the mushroom’s odour?*  
   If it is almond or anise then it is edible. **(Rule 1 Edible)**  
   If it is creosote or fishy or foul or musty or pungent or spicy then it is poisonous. **(Rule 1 Poisonous)**  
   If it has no odour then proceed to Step 2. |
| 2    | *Obtain the spore print of the mushroom.*  
   If the spore print is black or brown or buff or chocolate or orange or yellow then it is edible. **(Rule 2 Edible)**  
   If the spore print is green or purple then it is poisonous. **(Rule 2 Poisonous)**  
   If the spore print is white then proceed to Step 3. |
| 3    | *Examine the gill size of the mushroom.*  
   If the gill size is broad, then it is edible. **(Rule 3 Edible)**  
   If the gill size is narrow, then proceed to Step 4. |
| 4    | *Examine the stalk surface above the mushroom’s ring.*  
   If the surface is fibrous then it is edible. **(Rule 4 Edible)**  
   If the surface is silky or scaly then it is poisonous. **(Rule 4 Poisonous)**  
   If the surface is smooth then proceed to Step 5. |
| 5    | *Examine the mushroom for bruises.*  
   If it has no bruises then it is edible. **(Rule 5 Edible)**  
   If it has bruises then it is poisonous. **(Rule 5 Poisonous)** |

**Table 4-4.** Dawson et al’s (2000) step decision tree for classifying mushrooms.  
Decision points in this tree where mushrooms are classified (e.g., **Rule 1 Edible**) are given in bold.
The decision tree provided in Table 4-4 is a classical theory of how mushrooms can be classified. It is not surprising, then, that one can translate this decision tree into the lingua franca: Dawson et al. (2000) rewrote the decision tree as an equivalent set of production rules. They did so by using the features of mushrooms that must be true at each terminal branch of the decision tree as the conditions for a production. The action of this production is to classify the mushroom (i.e., to assert that a mushroom is either edible or poisonous). For instance, at the Rule 1 Edible decision point in Table 4-4, one could create the following production rule: “If the odour is anise or almond, then the mushroom is edible.” Similar productions can be created for later decision points in the algorithm; these productions will involve a longer list of mushroom features. The complete set of productions that were created for the decision tree algorithm is provided in Table 4-5.

Dawson et al. (2000) trained a network of value units to solve the mushroom classification problem and to determine whether a classical model (such as the decision tree from Table 4-4 or the production system from Table 4-5) could be translated into a network. To encode mushroom features, their network used 21 input units, 5 hidden value units, and 10 output value units. One output unit encoded the edible/poisonous classification—if a mushroom was edible, this unit was trained to turn on; otherwise this unit was trained to turn off.

<table>
<thead>
<tr>
<th>Decision Point From Table 4-4</th>
<th>Equivalent Production</th>
<th>Network Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1 Edible</td>
<td>P1: if (odor = anise) ∨ (odor = almond) → edible</td>
<td>2 or 3</td>
</tr>
<tr>
<td>Rule 1 Poisonous</td>
<td>P2: if (odor ≠ anise) ∧ (odor ≠ almond) ∧ (odor ≠ none) → not edible</td>
<td>1</td>
</tr>
<tr>
<td>Rule 2 Edible</td>
<td>P3: if (odor = none) ∧ (spore print colour ≠ green) ∧ (spore print colour ≠ purple) ∧ (spore print colour ¹ white) → edible</td>
<td>9</td>
</tr>
<tr>
<td>Rule 2 Poisonous</td>
<td>P4: if (odor = none) ∧ ((spore print colour = green) ∨ (spore print colour = purple)) → not edible</td>
<td>6</td>
</tr>
<tr>
<td>Rule 3 Edible</td>
<td>P5: if (odor = none) ∧ (spore print colour = white) ∧ (gill size = broad) → edible</td>
<td>4</td>
</tr>
<tr>
<td>Rule 4 Edible</td>
<td>P6: if (odor = none) ∧ (spore print colour = white) ∧ (gill size = narrow) ∧ (stalk surface above ring = fibrous) → edible</td>
<td>7 or 11</td>
</tr>
</tbody>
</table>
### Table 4-5. Dawson et al.'s (2000) production system translation of Table 4-4.

Conditions are given as sets of features. The Network Cluster column pertains to their artificial neural network trained on the mushroom problem and is described later in the text.

<table>
<thead>
<tr>
<th>Decision Point From Table 4-4</th>
<th>Equivalent Production</th>
<th>Network Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 4 Poisonous</td>
<td>P7: if (odor = none) ∧ (spore print colour = white) ∧ (gill size = narrow) ∧ ((stalk surface above ring = silky) ∨ (stalk surface above ring = scaly)) → not edible</td>
<td>5</td>
</tr>
<tr>
<td>Rule 5 Edible</td>
<td>P8: if (odor = none) ∧ (spore print colour = white) ∧ (gill size = narrow) ∧ (stalk surface above ring = smooth) ∧ (bruises = no) → edible</td>
<td>8 or 12</td>
</tr>
<tr>
<td>Rule 5 Poisonous</td>
<td>P9: if (odor = none) ∧ (spore print colour = white) ∧ (gill size = narrow) ∧ (stalk surface above ring = smooth) ∧ (bruises = yes) → not edible</td>
<td>10</td>
</tr>
</tbody>
</table>

The other nine output units were used to provide extra output learning, which was the technique employed to insert a classical theory into the network. Normally, a pattern classification system is only provided with information about what correct pattern labels to assign. For instance, in the mushroom problem, the system would typically only be taught to generate the label *edible* or the label *poisonous*. However, more information about the pattern classification task is frequently available. In particular, it is often known *why* an input pattern belongs to one class or another. It is possible to incorporate this information to the pattern classification problem by teaching the system not only to assign a pattern to a class (e.g., “edible”, “poisonous”) but to also generate a reason for making this classification (e.g., “passed Rule 1”, “failed Rule 4”). Elaborating a classification task along such lines is called the injection of hints or extra output learning (Abu-Mostafa, 1990; Suddarth & Kergosien, 1990).

Dawson et al. (2000) hypothesized that extra output learning could be used to insert the decision tree from Table 4-4 into a network. Table 4-4 provides nine different terminal branches of the decision tree at which mushrooms are assigned to categories (“Rule 1 edible”, “Rule 1 poisonous”, “Rule 2 edible”, etc.). The network learned to “explain” why it classified an input pattern in a particular way by turning on one of the nine extra output units to indicate which terminal branch of the decision tree was involved. In other words, the network (which required 8,699 epochs of training on the 8,124 different input patterns!) classified networks “for the same
reasons” as would the decision tree. This is why Dawson et al. hoped that this classical theory would literally be translated into the network.

Apart from the output unit behaviour, how could one support the claim that a classical theory had been translated into a connectionist network? Dawson et al. (2000) interpreted the internal structure of the network in an attempt to see whether such a network analysis would reveal an internal representation of the classical algorithm. If this were the case, then standard training practices would have succeeded in translating the classical algorithm into a PDP network.

One method that Dawson et al. (2000) used to interpret the trained network was a multivariate analysis of the network's hidden unit space. They represented each mushroom as the vector of five hidden unit activation values that it produced when presented to the network. They then performed a k-means clustering of this data. The k-means clustering is an iterative procedure that assigns data points to k different clusters in such a way that each member of a cluster is closer to the centroid of that cluster than to the centroid of any other cluster to which other data points have been assigned.

However, whenever cluster analysis is performed, one question that must be answered is How many clusters should be used?—in other words, what should the value of k be?. An answer to this question is called a stopping rule. Unfortunately, no single stopping rule has been agreed upon (Aldenderfer & Blashfield, 1984; Everitt, 1980). As a result, there exist many different types of methods for determining k (Milligan & Cooper, 1985).

While no general method exists for determining the optimal number of clusters, one can take advantage of heuristic information concerning the domain being clustered in order to come up with a satisfactory stopping rule for this domain. Dawson et al. (2000) argued that when the hidden unit activities of a trained network are being clustered, there must be a correct mapping from these activities to output responses, because one trained network itself has discovered one such mapping. They used this position to create the following stopping rule: “Extract the smallest number of clusters such that every hidden unit activity vector assigned to the same cluster produces the same output response in the network.” They used this rule to determine that the k-means analysis of the network’s hidden unit activity patterns required the use of 12 different clusters.

Dawson et al. (2000) then proceeded to examine the mushroom patterns that belonged to each cluster in order to determine what they had in common. For each cluster, they determined the set of descriptive features that each mushroom shared. They realized that each set of shared features they identified could be thought of as a condition, represented internally by the network as a vector of hidden unit activities, which results in the network producing a particular action, in particular, the edible/poisonous judgement represented by the first output unit.
For example, mushrooms that were assigned to Cluster 2 had an odour that was either almond or anise, which is represented by the network's five hidden units adopting a particular vector of activities. These activities serve as a condition that causes the network to assert that the mushroom is edible.

By interpreting a hidden unit vector in terms of condition features that are prerequisites to network responses, Dawson et al. (2000) discovered an amazing relationship between the clusters and the set of productions in Table 4-5. They determined that each distinct class of hidden unit activities (i.e., each cluster) corresponded to one, and only one, of the productions listed in the table. This mapping is provided in the last column of Table 4-5. In other words, when one describes the network as generating a response because its hidden units are in one state of activity, one can translate this into the claim that the network is executing a particular production. This shows that the extra output learning translated the classical algorithm into a network model.

The translation of a network into a production system, or vice versa, is an example of new wave reductionism (Bickle, 1996; Endicott, 1998). In new wave reductionism, one does not reduce a secondary theory directly to a primary theory. Instead, one takes the primary theory and constructs from it a structure that is analogous to the secondary theory, but which is created in the vocabulary of the primary theory. Theory reduction involves constructing a mapping between the secondary theory and its image constructed from the primary theory. “The older theory, accordingly, is never deduced; it is just the target of a relevantly adequate mimicry” (Churchland, 1985, p. 10).

Dawson et al.’s (2000) interpretation is a new wave intertheoretic reduction because the production system of Table 4-5 represents the intermediate structure that is analogous to the decision tree of Table 4-4. “Adequate mimicry” was established by mapping different classes of hidden unit states to the execution of particular productions. In turn, there is a direct mapping from any of the productions back to the decision tree algorithm. Dawson et al. concluded that they had provided an exact translation of a classical algorithm into a network of value units.

The relationship between hidden unit activities and productions in Dawson et al.’s (2000) mushroom network is in essence an example of equivalence between symbolic and subsymbolic accounts. This implies that one cannot assume that classical models and connectionist networks are fundamentally different at the algorithmic level, because one type of model can be translated into the other. It is possible to have a classical model that is exactly equivalent to a PDP network.

This result provides very strong support for the position proposed by Vera and Simon (1993). The detailed analysis provided by Dawson et al. (2000) permitted them to make claims of the type “Network State \( x \) is equivalent to Production \( y \).” Of course, this one result cannot by itself validate Vera and Simon’s argument. For
instance, can any classical theory be translated into a network? This is one type of algorithmic-level issue that requires a great deal of additional research. As well, the translation works both ways: perhaps artificial neural networks provide a biologically plausible lingua franca for classical architectures!

4.13 Of Coarse Codes

The notion of representation in classical cognitive science is tightly linked to the structure/process distinction that is itself inspired by the digital computer. An explicit set of rules is proposed to operate on a set of symbols that permits its components to be identified, digitally, as tokens that belong to particular symbol types.

In contrast, artificial neural networks dispense (at first glance) with the sharp distinction between structure and process that characterizes classical cognitive science. Instead, networks themselves take the form of dynamic symbols that represent information at the same time as they transform it. The dynamic, distributed nature of artificial neural networks appears to make them more likely to be explained using statistical mechanics than using propositional logic.

One of the putative advantages of connectionist cognitive science is that it can inspire alternative notions of representation. The blurring of the structure/process distinction, the seemingly amorphous nature of the internal structure that characterizes many multilayer networks, leads to one such proposal, called coarse coding.

A coarse code is one in which an individual unit is very broadly tuned, sensitive to either a wide range of features or at least to a wide range of values for an individual feature (Churchland & Sejnowski, 1992; Hinton, McClelland, & Rumelhart, 1986). In other words, individual processors are themselves very inaccurate devices for measuring or detecting a feature. The accurate representation of a feature can become possible, though, by pooling or combining the responses of many such inaccurate detectors, particularly if their perspectives are slightly different (e.g., if they are sensitive to different ranges of features, or if they detect features from different input locations).

A familiar example of coarse coding is provided by the nineteenth trichromatic theory of colour perception (Helmholtz, 1968; Wasserman, 1978). According to this theory, colour perception is mediated by three types of retinal cone receptors. One is maximally sensitive to short (blue) wavelengths of light, another is maximally sensitive to medium (green) wavelengths, and the third is maximally sensitive to long (red) wavelengths. Thus none of these types of receptors are capable of representing, by themselves, the rich rainbow of perceptible hues.

However, these receptors are broadly tuned and have overlapping sensitivities. As a result, most light will activate all three channels simultaneously, but to different degrees. Actual colored light does not produce sensations of absolutely pure
color; that red, for instance, even when completely freed from all admixture of white light, still does not excite those nervous fibers which alone are sensitive to impressions of red, but also, to a very slight degree, those which are sensitive to green, and perhaps to a still smaller extent those which are sensitive to violet rays. (Helmholtz, 1968, p. 97)

The pooling of different activities of the three channels permits a much greater variety of colours to be represented and perceived.

We have already seen examples of coarse coding in some of the network analyses that were presented earlier in this chapter. For instance, consider the chord recognition network. It was shown in Table 4-3 that none of its hidden units were accurate chord detectors. Hidden Units 1 and 2 did not achieve maximum activity when presented with any chord. When Hidden Unit 3 achieved maximum activity, this did not distinguish a 6th chord from a major 7th chord. However, when patterns were represented as points in a three-dimensional space, where the coordinates of each point were defined by a pattern’s activity in each of the three hidden units (Figures 4-12 and 4-13), perfect chord classification was possible.

Other connectionist examples of coarse coding are found in studies of networks trained to accomplish navigational tasks, such as making judgments about the distance or direction between pairs of cities on a map (Dawson & Boechler, 2007; Dawson, Boechler, & Orsten, 2005; Dawson, Boechler, & Valsangkar-Smyth, 2000). For instance, Dawson and Boechler (2007) trained a network to judge the heading from one city on a map of Alberta to another. Seven hidden value units were required to accomplish this task. Each of these hidden units could be described as being sensitive to heading. However, this sensitivity was extremely coarse—some hidden units could resolve directions only to the nearest 180°. Nevertheless, a linear combination of the activities of all seven hidden units represented the desired direction between cities with a high degree of accuracy.

Similarly, Dawson, Boechler, and Valsangkar-Smyth (2000) trained a network of value units to make distance judgments between all possible pairs of 13 Albertan cities. This network required six hidden units to accomplish this task. Again, these units provided a coarse coding solution to the problem. Each hidden unit could be described as occupying a location on the map of Alberta through which a line was drawn at a particular orientation. This oriented line provided a one-dimensional map of the cities: connection weights encoded the projections of the cities from the two-dimensional map onto each hidden unit’s one-dimensional representation. However, because the hidden units provided maps of reduced dimensionality, they were wildly inaccurate. Depending on the position of the oriented line, two cities that were far apart in the actual map could lie close together on a hidden unit’s representation. Fortunately, because each of these inaccurate hidden unit maps encoded projections from different perspectives, the combination of their activities...
was able to represent the actual distance between all city pairs with a high degree of accuracy.

The discovery of coarse coding in navigational networks has important theoretical implications. Since the discovery of place cells in the hippocampus (O’Keefe & Dostrovsky, 1971), it has been thought that one function of the hippocampus is to instantiate a cognitive map (O’Keefe & Nadel, 1978). One analogy used to explain cognitive maps is that they are like graphical maps (Kitchin, 1994). From this, one might predict that the cognitive map is a metric, topographically organized, two-dimensional array in which each location in the map (i.e., each place in the external world) is associated with the firing of a particular place cell, and neighbouring place cells represent neighbouring places in the external world.

However, this prediction is not supported by anatomical evidence. First, place cells do not appear to be topographically organized (Burgess, Recce, & O’Keefe, 1995; McNaughton et al., 1996). Second, the receptive fields of place cells are at best locally metric, because one cannot measure the distance between points that are more than about a dozen body lengths apart because of a lack of receptive field overlap (Touretzky, Wan, & Redish, 1994). Some researchers now propose that the cognitive map doesn’t really exist, but that map-like properties emerge when place cells are coordinated with other types of cells, such as head direction cells, which fire when an animal’s head is pointed in a particular direction, regardless of the animal’s location in space (McNaughton et al., 1996; Redish, 1999; Redish & Touretzky, 1999; Touretzky, Wan, & Redish, 1994).

Dawson et al. (2000) observed that their navigational network is also subject to the same criticisms that have been levied against the notion of a topographically organized cognitive map. The hidden units did not exhibit topographic organization, and their inaccurate responses suggest that they are at best locally metric.

Nevertheless, the behaviour of the Dawson et al. (2000) network indicated that it represented information about a metric space. That such behaviour can be supported by the type of coarse coding discovered in this network suggests that metric, spatial information can be encoded in a representational scheme that is not isomorphic to a graphical map. This raises the possibility that place cells represent spatial information using a coarse code which, when its individual components are inspected, is not very map-like at all. O’Keefe and Nadel (1978, p. 78) were explicitly aware of this kind of possibility: "The cognitive map is not a picture or image which ‘looks like’ what it represents; rather, it is an information structure from which map-like images can be reconstructed and from which behaviour dependent upon place information can be generated."

What are the implications of the ability to interpret the internal structure of artificial neural networks to the practice of connectionist cognitive science?
When New Connectionism arose in the 1980s, interest in it was fuelled by two complementary perspectives (Medler, 1998). First, there was growing dissatisfaction with the progress being made in classical cognitive science and symbolic artificial intelligence (Dreyfus, 1992; Dreyfus & Dreyfus, 1988). Second, seminal introductions to artificial neural networks (McClelland & Rumelhart, 1986; Rumelhart & McClelland, 1986c) gave the sense that the connectionist architecture was a radical alternative to its classical counterpart (Schneider, 1987).

The apparent differences between artificial neural networks and classical models led to an early period of research in which networks were trained to accomplish tasks that had typically been viewed as prototypical examples of classical cognitive science (Bechtel, 1994; Rumelhart & McClelland, 1986a; Seidenberg & McClelland, 1989; Sejnowski & Rosenberg, 1988). These networks were then used as “existence proofs” to support the claim that non-classical models of classical phenomena are possible. However, detailed analyses of these networks were not provided, which meant that, apart from intuitions that connectionism is not classical, there was no evidence to support claims about the non-classical nature of the networks’ solutions to the classical problems. Because of this, this research perspective has been called gee whiz connectionism (Dawson, 2004, 2009).

Of course, at around the same time, prominent classical researchers were criticizing the computational power of connectionist networks (Fodor & Pylyshyn, 1988), arguing that connectionism was a throwback to less powerful notions of associationism that classical cognitive science had already vanquished (Bever, Fodor, & Garrett, 1968; Chomsky, 1957, 1959b, 1965). Thus gee whiz connectionism served an important purpose: providing empirical demonstrations that connectionism might be a plausible medium in which cognitive science can be fruitfully pursued.

However, it was noted earlier that there exists a great deal of research on the computational power of artificial neural networks (Girosi & Poggio, 1990; Hartman, Keeler, & Kowalski, 1989; Lippmann, 1989; McCulloch & Pitts, 1943; Moody & Darken, 1989; Poggio & Girosi, 1990; Renals, 1989; Siegelmann, 1999; Siegelmann & Sontag, 1991); the conclusion from this research is that multilayered networks have the same in-principle power as any universal machine. This leads, though, to the demise of gee whiz connectionism, because if connectionist systems belong to the class of universal machines, “it is neither interesting nor surprising to demonstrate that a network can learn a task of interest” (Dawson, 2004, p. 118). If a network’s ability to learn to perform a task is not of interest, then what is?

It can be extremely interesting, surprising, and informative to determine what regularities the network exploits. What kinds of regularities in the input patterns has the network discovered? How does it represent these regularities? How are these regularities combined to govern the response of the network? (Dawson, 2004, p. 118)
By uncovering the properties of representations that networks have discovered for mediating an input-output relationship, connectionist cognitive scientists can discover new properties of cognitive phenomena.

4.14 Architectural Connectionism: An Overview

In the last several sections, we have been concerned with interpreting the internal structure of multilayered artificial neural networks. While some have claimed that all that can be found within brains and networks is goo (Mozer & Smolensky, 1989), the preceding examples have shown that detailed interpretations of internal network structure are both possible and informative. These interpretations reveal algorithmic-level details about how artificial neural networks use their hidden units to mediate mappings from inputs to outputs.

If the goal of connectionist cognitive science is to make new representational discoveries, then this suggests that it be practised as a form of synthetic psychology (Braitenberg, 1984; Dawson, 2004) that incorporates both synthesis and analysis, and that involves both forward engineering and reverse engineering.

The analytic aspect of connectionist cognitive science involves peering inside a network in order to determine how its internal structure represents solutions to problems. The preceding pages of this chapter have provided several examples of this approach, which seems identical to the reverse engineering practised by classical cognitive scientists.

The reverse engineering phase of connectionist cognitive science is also linked to classical cognitive science, in the sense that the results of these analyses are likely to provide the questions that drive algorithmic-level investigations. Once a novel representational format is discovered in a network, a key issue is to determine whether it also characterizes human or animal cognition. One would expect that when connectionist cognitive scientists evaluate their representational discoveries, they should do so by gathering the same kind of relative complexity, intermediate state, and error evidence that classical cognitive scientists gather when seeking strong equivalence.

Before one can reverse engineer a network, one must create it. And if the goal of such a network is to discover surprising representational regularities, then it should be created by minimizing representational assumptions as much as possible. One takes the building blocks available in a particular connectionist architecture, creates a network from them, encodes a problem for this network in some way, and attempts to train the network to map inputs to outputs.

This synthetic phase of research involves exploring different network structures (e.g., different design decisions about numbers of hidden units, or types of activation functions) and different approaches to encoding inputs and outputs. The
idea is to give the network as many degrees of freedom as possible to discover representational regularities that have not been imposed or predicted by the researcher. These decisions all involve the architectural level of investigation.

One issue, though, is that networks are greedy, in the sense that they will exploit whatever resources are available to them. As a result, fairly idiosyncratic and specialized detectors are likely to be found if too many hidden units are provided to the network, and the network’s performance may not transfer well when presented with novel stimuli. To deal with this, one must impose constraints by looking for the simplest network that will reliably learn the mapping of interest. The idea here is that such a network might be the one most likely to discover a representation general enough to transfer the network’s ability to new patterns.

Importantly, sometimes when one makes architectural decisions to seek the simplest network capable of solving a problem, one discovers that the required network is merely a perceptron that does not employ any hidden units. In the remaining sections of this chapter I provide some examples of simple networks that are capable of performing interesting tasks. In section 4.15 the relevance of perceptrons to modern theories of associative learning is described. In section 4.16 I present a perceptron model of the reorientation task. In section 4.17 an interpretation is given for the structure of a perceptron that learns a seemingly complicated progression of musical chords.

4.15 New Powers of Old Networks

The history of artificial neural networks can be divided into two periods, Old Connectionism and New Connectionism (Medler, 1998). New Connectionism studies powerful networks consisting of multiple layers of units, and connections are trained to perform complex tasks. Old Connectionism studied networks that belonged to one of two classes. One was powerful multilayer networks that were hand wired, not trained (McCulloch & Pitts, 1943). The other was less powerful networks that did not have hidden units but were trained (Rosenblatt, 1958, 1962; Widrow, 1962; Widrow & Hoff, 1960).

Perceptrons (Rosenblatt, 1958, 1962) belong to Old Connectionism. A perceptron is a standard pattern associator whose output units employ a nonlinear activation function. Rosenblatt’s perceptrons used the Heaviside step function to convert net input into output unit activity. Modern perceptrons use continuous nonlinear activation functions, such as the logistic or the Gaussian (Dawson, 2004, 2005, 2008; Dawson et al., 2009; Dawson et al., 2010).

Perceptrons are trained using an error-correcting variant of Hebb-style learning (Dawson, 2004). Perceptron training associates input activity with output unit error as follows. First, a pattern is presented to the input units, producing output
unit activity via the existing connection weights. Second, output unit error is computed by taking the difference between actual output unit activity and desired output unit activity for each output unit in the network. This kind of training is called supervised learning, because it requires an external trainer to provide the desired output unit activities. Third, Hebb-style learning is used to associate input unit activity with output unit error: weight change is equal to a learning rate times input unit activity times output unit error. (In modern perceptrons, this triple product can also be multiplied by the derivative of the output unit’s activation function, resulting in gradient descent learning [Dawson, 2004]).

The supervised learning of a perceptron is designed to reduce output unit errors as training proceeds. Weight changes are proportional to the amount of generated error. If no errors occur, then weights are not changed. If a task’s solution can be represented by a perceptron, then repeated training using pairs of input-output stimuli is guaranteed to eventually produce zero error, as proven in Rosenblatt’s perceptron convergence theorem (Rosenblatt, 1962).

Being a product of Old Connectionism, there are limits to the range of input-output mappings that can be mediated by perceptrons. In their famous computational analyses of what perceptrons could and could not learn to compute, Minsky and Papert (1969) demonstrated that perceptrons could not learn to distinguish some basic topological properties easily discriminated by humans, such as the difference between connected and unconnected figures. As a result, interest in and funding for Old Connectionist research decreased dramatically (Medler, 1998; Papert, 1988).

However, perceptrons are still capable of providing new insights into phenomena of interest to cognitive science. The remainder of this section illustrates this by exploring the relationship between perceptron learning and classical conditioning.

The primary reason that connectionist cognitive science is related to empiricism is that the knowledge of an artificial neural network is typically acquired via experience. For instance, in supervised learning a network is presented with pairs of patterns that define an input-output mapping of interest, and a learning rule is used to adjust connection weights until the network generates the desired response to a given input pattern.

In the twentieth century, prior to the birth of artificial neural networks (McCulloch & Pitts, 1943), empiricism was the province of experimental psychology. A detailed study of classical conditioning (Pavlov, 1927) explored the subtle regularities of the law of contiguity. Pavlovian, or classical, conditioning begins with an unconditioned stimulus (US) that is capable, without training, of producing an unconditioned response (UR). Also of interest is a conditioned stimulus (CS) that when presented will not produce the UR. In classical conditioning, the CS is paired with the US for a number of trials. As a result of this pairing, which places the CS
in contiguity with the UR, the CS becomes capable of eliciting the UR on its own. When this occurs, the UR is then known as the conditioned response (CR).

Classical conditioning is a very basic kind of learning, but experiments revealed that the mechanisms underlying it were more complex than the simple law of contiguity. For example, one phenomenon found in classical conditioning is blocking (Kamin, 1968). Blocking involves two conditioned stimuli, CS_A and CS_B. Either stimulus is capable of being conditioned to produce the CR. However, if training begins with a phase in which only CS_A is paired with the US and is then followed by a phase in which both CS_A and CS_B are paired with the US, then CS_B fails to produce the CR. The prior conditioning involving CS_A blocks the conditioning of CS_B, even though in the second phase of training CS_B is contiguous with the UR.

The explanation of phenomena such as blocking required a new model of associative learning. Such a model was proposed in the early 1970s by Robert Rescorla and Allen Wagner (Rescorla & Wagner, 1972). This mathematical model of learning has been described as being cognitive, because it defines associative learning in terms of expectation. Its basic idea is that a CS is a signal about the likelihood that a US will soon occur. Thus the CS sets up expectations of future events. If these expectations are met, then no learning will occur. However, if these expectations are not met, then associations between stimuli and responses will be modified. “Certain expectations are built up about the events following a stimulus complex; expectations initiated by that complex and its component stimuli are then only modified when consequent events disagree with the composite expectation” (p. 75).

The expectation-driven learning that was formalized in the Rescorla-Wagner model explained phenomena such as blocking. In the second phase of learning in the blocking paradigm, the coming US was already signaled by CS_A. Because there was no surprise, no conditioning of CS_B occurred. The Rescorla-Wagner model has had many other successes; though it is far from perfect (Miller, Barnet, & Grahame, 1995; Walkenbach & Haddad, 1980), it remains an extremely influential, if not the most influential, mathematical model of learning.

The Rescorla-Wagner proposal that learning depends on the amount of surprise parallels the notion in supervised training of networks that learning depends on the amount of error. What is the relationship between Rescorla-Wagner learning and perceptron learning?

Proofs of the equivalence between the mathematics of Rescorla-Wagner learning and the mathematics of perceptron learning have a long history. Early proofs demonstrated that one learning rule could be translated into the other (Gluck & Bower, 1988; Sutton & Barto, 1981). However, these proofs assumed that the networks had linear activation functions. Recently, it has been proven that if when it is more properly assumed that networks employ a nonlinear activation
function, one can still translate Rescorla-Wagner learning into perceptron learning, and vice versa (Dawson, 2008).

One would imagine that the existence of proofs of the computational equivalence between Rescorla-Wagner learning and perceptron learning would mean that perceptrons would not be able to provide any new insights into classical conditioning. However, this is not correct. Dawson (2008) has shown that if one puts aside the formal comparison of the two types of learning and uses perceptrons to simulate a wide variety of different classical conditioning paradigms, then some puzzling results occur. On the one hand, perceptrons generate the same results as the Rescorla-Wagner model for many different paradigms. Given the formal equivalence between the two types of learning, this is not surprising. On the other hand, for some paradigms, perceptrons generate different results than those predicted from the Rescorla-Wagner model (Dawson, 2008, Chapter 7). Furthermore, in many cases these differences represent improvements over Rescorla-Wagner learning. If the two types of learning are formally equivalent, then how is it possible for such differences to occur?

Dawson (2008) used this perceptron paradox to motivate a more detailed comparison between Rescorla-Wagner learning and perceptron learning. He found that while these two models of learning were equivalent at the computational level of investigation, there were crucial differences between them at the algorithmic level. In order to train a perceptron, the network must first behave (i.e., respond to an input pattern) in order for error to be computed to determine weight changes. In contrast, Dawson showed that the Rescorla-Wagner model defines learning in such a way that behaviour is not required!

Dawson’s (2008) algorithmic analysis of Rescorla-Wagner learning is consistent with Rescorla and Wagner’s (1972) own understanding of their model: “Independent assumptions will necessarily have to be made about the mapping of associative strengths into responding in any particular situation” (p. 75). Later, they make this same point much more explicitly:

We need to provide some mapping of [associative] values into behavior. We are not prepared to make detailed assumptions in this instance. In fact, we would assume that any such mapping would necessarily be peculiar to each experimental situation, and depend upon a large number of ‘performance’ variables.

(Rescorla & Wagner, 1972, p. 77)

Some knowledge is tacit: we can know more than we can tell (Polanyi, 1966). Dawson (2008) noted that the Rescorla-Wagner model presents an interesting variant of this theme, where if there is no explicit need for a behavioural theory, then there is no need to specify it explicitly. Instead, researchers can ignore Rescorla and Wagner’s (1972) call for explicit models to convert associative strengths into behaviour and instead assume unstated, tacit theories such as “strong associations produce
 Researchers evaluate the Rescorla-Wagner model (Miller, Barnet, & Grahame, 1995; Walkenbach & Haddad, 1980) by agreeing that associations will eventually lead to behavior, without actually stating how this is done. In the Rescorla-Wagner model, learning comes first and behavior comes later—maybe.

Using perceptrons to study classical conditioning paradigms contributes to the psychological understanding of such learning in three ways. First, at the computational level, it demonstrates equivalences between independent work on learning conducted in computer science, electrical engineering, and psychology (Dawson, 2008; Gluck & Bower, 1988; Sutton & Barto, 1981).

Second, the results of training perceptrons in these paradigms raise issues that lead to a more sophisticated understanding of learning theories. For instance, the perceptron paradox led to the realization that when the Rescorla-Wagner model is typically used, accounts of converting associations into behavior are unspecified. Recall that one of the advantages of computer simulation research is exposing tacit assumptions (Lewandowsky, 1993).

Third, the activation functions that are a required property of a perceptron serve as explicit theories of behavior to be incorporated into the Rescorla-Wagner model. More precisely, changes in activation function result in changes to how the perceptron responds to stimuli, indicating the importance of choosing a particular architecture (Dawson & Spetch, 2005). The wide variety of activation functions that are available for artificial neural networks (Duch & Jankowski, 1999) offers a great opportunity to explore how changing theories of behavior—or altering architectures—affect the nature of associative learning.

The preceding paragraphs have shown how the perceptron can be used to inform theories of a very old psychological phenomenon, classical conditioning. We now consider how perceptrons can play a role in exploring a more modern topic, reorientation, which was described from a classical perspective in Chapter 3 (Section 3.12).

4.16 Connectionist Reorientation

In the reorientation task, an agent learns that a particular place—usually a corner of a rectangular arena—is a goal location. The agent is then removed from the arena, disoriented, and returned to an arena. Its task is to use the available cues to relocate the goal. Theories of reorientation assume that there are two types of cues available for reorienting: local feature cues and relational geometric cues. Studies indicate that both types of cues are used for reorienting, even in cases where geometric cues are irrelevant (Cheng & Newcombe, 2005). As a result, some
theories have proposed that a geometric module guides reorienting behaviour (Cheng, 1986; Gallistel, 1990).

The existence of a geometric module has been proposed because different kinds of results indicate that the processing of geometric cues is mandatory. First, in some cases agents continue to make rotational errors (i.e., the agent does not go to the goal location, but goes instead to an incorrect location that is geometrically identical to the goal location) even when a feature disambiguates the correct corner (Cheng, 1986; Hermer & Spelke, 1994). Second, when features are removed following training, agents typically revert to choosing both of the geometrically correct locations (Kelly et al., 1998; Sovrano et al., 2003). Third, when features are moved, agents generate behaviours that indicate that both types of cues were processed (Brown, Spetch, & Hurd, 2007; Kelly, Spetch, & Heth, 1998).

Recently, some researchers have begun to question the existence of geometric modules. One reason for this is that the most compelling evidence for claims of modularity comes from neuroscience (Dawson, 1998; Fodor, 1983), but such evidence about the modularity of geometry in the reorientation task is admittedly sparse (Cheng & Newcombe, 2005). This has led some researchers to propose alternative notions of modularity when explaining reorientation task regularities (Cheng, 2005, 2008; Cheng & Newcombe, 2005).

Still other researchers have explored how to abandon the notion of the geometric module altogether. They have proceeded by creating models that produce the main findings from the reorientation task, but they do so without using a geometric module. A modern perceptron that uses the logistic activation function has been shown to provide just such a model (Dawson et al., 2010).

The perceptrons used by Dawson et al. (2010) used a single output unit that, when the perceptron was “placed” in the original arena, was trained to turn on to the goal location and turn off to all of the other locations. A set of input units was used to represent the various cues—featural and geometric—available at each location. Both feature cues and geometric cues were treated in an identical fashion by the network; no geometric module was built into it.

After training, the perceptron was “placed” into a new arena; this approach was used to simulate the standard variations of the reorientation task in which geometric cues and feature cues could be placed in conflict. In the new arena, the perceptron was “shown” all of the possible goal locations by activating its input units with the features available at each location. The resulting output unit activity was interpreted as representing the likelihood that there was a reward at any of the locations in the new arena.

The results of the Dawson et al. (2010) simulations replicated the standard reorientation task findings that have been used to argue for the existence of a geometric module. However, this was accomplished without using such a module. These
simulations also revealed new phenomena that have typically not been explored in
the reorientation task that relate to the difference between excitatory cues, which
indicate the presence of a reward, and inhibitory cues, which indicate the absence of
a reward. In short, perceptrons have been used to create an associative, nonmodular
theory of reorientation.

4.17 Perceptrons and Jazz Progressions

We have seen that a particular type of network from Old Connectionism, the per-
ceptron, can be usefully applied in the studies of classical conditioning and reori-
entation. In the current section we see that it can also be used to explore musical
regularities. Also illustrated is the interpretation of the internal structure of such a
network, which demonstrates that even simple networks can reveal some interesting
algorithmic properties.

Jazz progressions are sequences of chords. Consider the C major scale pre-
sented earlier, in Figure 4-8. If one takes the first note of the scale, C, as the root
and adds every second note in the scale—E, G, and B)—the result is a four-note
chord—a tetrachord—called the C major 7th chord (Cmaj7). Because the root of this
chord is the first note of the scale, this is identified as the I chord for C major. Other
tetrachords can also be built for this key. Starting with the second note in the scale,
D, and adding the notes F, A, and C produces D minor 7th (Dm7). Because its root is
the second note of the scale, this is identified as the II chord for the key of C major.
Using G as the root and adding the notes B, D, and F creates the G dominant 7th
chord (G7). It is the V chord of the key of C major because its root is the fifth note
of the C major scale.

The I, II, and V chords are the three most commonly played jazz chords, and in
jazz they often appear in the context of the II-V-I progression (Levine, 1989). This
chord progression involves playing these chords in a sequence that begins with the
II chord, moves to the V chord, and ends on the I chord. The II-V-I progression is
important for several reasons.

First, chord progressions are used to establish tonality, that is, to specify to the
listener the musical key in which a piece is being played. They do so by setting up
expectancies about what is to be played next. For any major key, the most stable
tones are notes I, IV, and V (Krumhansl, 1990), and the most stable chords are the
ones built on those three notes.

Second, in the perception of chord sequences there are definite preferences for
the IV chord to resolve into the V chord and for the V chord to resolve into the I chord,
producing the IV-V-I progression that is common in cadences in classical music
(Bharucha, 1984; Jarvinen, 1995; Katz, 1995; Krumhansl, Bharucha, & Kessler, 1982;
Rosner & Narmour, 1992). There is a similar relationship between the IV chord and
the II chord if the latter is minor (Steedman, 1984). Thus the II-V-I progression is a powerful tool for establishing the tonality of a musical piece.

Third, the II-V-I progression lends itself to a further set of chord progressions that move from key to key, providing variety but also establishing tonality. After playing the Cmaj7 chord to end the II-V-I progression for C major, one can change two notes to transform Cmaj7 into Cm7, which is the II chord of a different musical key, A# major. As a result, one can move from performing the II-V-I progression in C major to performing the same progression in a major key one tone lower. This process can be repeated; the full set of chord changes is provided in Table 4-6. Note that this progression eventually returns to the starting key of C major, providing another powerful cue of tonality.

<table>
<thead>
<tr>
<th>Chord Progression For Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>A#</td>
</tr>
<tr>
<td>G#</td>
</tr>
<tr>
<td>F#</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

Table 4-6. A progression of II-V-I progressions, descending from the key of C major. The chords in each row are played in sequence, and after playing one row, the next row is played.

A connectionist network can be taught the II-V-I chord progression. During training, one presents, in pitch class format, a chord belonging to the progression. The network learns to output the next chord to be played in the progression, again using pitch class format. Surprisingly, this problem is very simple: it is linearly separable and can be solved by a perceptron!

How does a perceptron represent this jazz progression? Because a perceptron has no hidden units, its representation must be stored in the set of connection weights between the input and output units. However, this matrix of connection weights is too complex to reveal its musical representations simply by inspecting it. Instead, multivariate statistics must be used.

First, one can convert the raw connection weights into a correlation matrix. That is, one can compute the similarity of each pair of output units by computing
the correlation between the connection weights that feed into them. Once the weights have been converted into correlations, further analyses are then available to interpret network representations. Multidimensional scaling (MDS) can summarize the relationships within a correlation matrix made visible by creating a map (Kruskal & Wish, 1978; Romney, Shepard, & Nerlove, 1972; Shepard, Romney, & Nerlove, 1972). Items are positioned in the map in such a way that the more similar items are, the closer together they are in the map.

The MDS of the jazz progression network’s correlations produced a one-dimensional map that provided a striking representation of musical relationships amongst the notes. In a one-dimensional MDS solution, each data point is assigned a single number, which is its coordinate on the single axis that is the map. The coordinate for each note is presented in a bar chart in Figure 4-16.

![Figure 4-16](image)

**Figure 4-16.** Coordinates associated with each output note, taken from an MDS of the Table 4-8 correlations. Shading reflects groupings of notes as circles of major thirds.

The first regularity evident from Figure 4-16 is that half of the notes have negative coordinates, while the other half have positive coordinates. That is, the perceptron’s connection weights separate musical notes into two equal-sized classes. These classes reflect a basic property of the chord progressions learned by the network: all of the notes that have positive coordinates were also used as major keys in which the II-V-I progression was defined, while none of the notes with negative coordinates were used in this fashion.
Another way to view the two classes of notes revealed by this analysis is in terms of the two circles of major seconds that were presented in Figure 4-10. The first circle of major seconds contains only those notes that have positive coordinates in Figure 4-16. The other circle of major seconds captures the set of notes that have negative coordinates in Figure 4-16. In other words, the jazz progression network acts as if it has classified notes in terms of the circles of major seconds!

The order in which the notes are arranged in the one-dimensional map is also related to the four circles of major thirds that were presented in Figure 4-11. The bars in Figure 4-16 have been coloured to reveal four sets of three notes each. Each of these sets of notes defines a circle of major thirds. The MDS map places notes in such a way that the notes of one such circle are listed in order, followed by the notes of another circle of major thirds.

To summarize, one musical formalism is the II-V-I jazz progression. Interestingly, this formalism can be learned by a network from Old Connectionism, the perceptron. Even though this network is simple, interpreting its representations is not straightforward and requires the use of multivariate statistics. However, when such analysis is performed, it appears that the network captures the regularities of this jazz progression using the strange circles that were encountered in the earlier section on chord classification. That is, the connection weights of the perceptron reveal circles of major seconds and circles of major thirds.

4.18 What Is Connectionist Cognitive Science?

The purpose of the current chapter was to introduce the elements of connectionist cognitive science, the “flavour” of cognitive science that was seen first as Old Connectionism in the 1940s (McCulloch & Pitts, 1943) and which peaked by the late 1950s (Rosenblatt, 1958, 1962; Widrow, 1962; Widrow & Hoff, 1960). Criticisms concerning the limitations of such networks (Minsky & Papert, 1969) caused connectionist research to almost completely disappear until the mid-1980s (Papert, 1988), when New Connectionism arose in the form of techniques capable of training powerful multilayered networks (McClelland & Rumelhart, 1986; Rumelhart & McClelland, 1986c).

Connectionism is now well established as part of mainstream cognitive science, although its relationship to classical cognitive science is far from clear. Artificial neural networks have been used to model a dizzying variety of phenomena including animal learning (Enquist & Ghirlanda, 2005; Schmajuk, 1997), cognitive development (Elman et al., 1996), expert systems (Gallant, 1993), language (Mammone, 1993; Sharkey, 1992), pattern recognition and perception (Pao, 1989; Ripley, 1996; Wechsler, 1992), and musical cognition (Griffith & Todd, 1999; Todd & Loy, 1991).
Given the breadth of connectionist cognitive science, only a selection of its elements have been introduced in this chapter; capturing all of the important contributions of connectionism in a single chapter is not possible. A proper treatment of connectionism requires a great deal of further reading; fortunately connectionism is described in a rich and growing literature (Amit, 1989; Anderson, 1995; Anderson & Rosenfeld, 1998; Bechtel & Abrahamson, 2002; Carpenter & Grossberg, 1992; Caudill & Butler, 1992a, 1992b; Churchland, 1986; Churchland & Sejnowski, 1992; Clark, 1989, 1993; Dawson, 2004, 2005; Grossberg, 1988; Horgan & Tienson, 1996; Quinlan, 1991; Ramsey, Stich, & Rumelhart, 1991; Ripley, 1996; Rojas, 1996).

Connectionist cognitive science is frequently described as a reaction against the foundational assumptions of classical cognitive science. The roots of classical cognitive science draw inspiration from the rationalist philosophy of Descartes, with an emphasis on nativism and logicism (Chomsky, 1966; Devlin, 1996). In contrast, the foundations of connectionist cognitive science are the empiricist philosophy of Locke and the associationist psychology that can be traced from the early British empiricists to the more modern American behaviourists. Connectionist networks acquire structure or knowledge via experience; they often begin as blank slates (Pinker, 2002) and acquire structure as they learn about their environments (Bechtel, 1985; Clark, 1989, 1993; Hillis, 1988).

Classical cognitive science departed from Cartesian philosophy by seeking materialist accounts of mentality. This view was inspired by the digital computer and the fact that electronic switches could be assigned abstract logical interpretations (Shannon, 1938).

Connectionism is materialist as well, but arguably in a more restricted sense than classical cognitive science. The classical approach appeals to the multiple realization argument when it notes that under the proper interpretation, almost any physical substrate could instantiate information processing or symbol manipulation (Hillis, 1998). In contrast, connectionism views the digital computer metaphor as mistaken. Connectionists claim that the operations of such a device—regardless of its material nature—are too slow, brittle, and inflexible to be appropriate for modelling cognition. Connectionism posits instead that the brain is the only appropriate material for realizing the mind and researchers attempt to frame its theories in terms of information processing that is biologically plausible or neuronally inspired (Amit, 1989; Burnod, 1990; Gluck & Myers, 2001).

In adopting the digital computer metaphor and the accompanying logicist view that cognition is the result of rule-governed symbol manipulation, classical cognitive science is characterized by a marked structure/process distinction. That is, classical models—typified by Turing machines (Turing, 1936) or production systems (Newell & Simon, 1972)—distinguish between the symbols being manipulated...
and the explicit rules doing the manipulating. This distinction is usually marked in models by having separate locations for structure and process, such as a memory that holds symbols and a central controller that holds the processes.

In abandoning the digital computer metaphor and adopting a notion of information processing that is biologically inspired, connectionist cognitive science abandons or blurs the structure/process distinction. Neural networks can be viewed as both structure and process; they have been called active data structures (Hillis, 1985). This has led to an extensive debate about whether theories of cognition require explicit rules (Ramsey, Stich, & Rumelhart, 1991).

The digital computer metaphor adopted by classical cognitive science leads it to also adopt a particular notion of control. In particular, classical models invoke a notion of serial control in which representations can only be manipulated one rule at a time. When classical problem solvers search a problem space in order to solve a problem (Newell & Simon, 1972), they do so to discover a sequence of operations to perform.

In contrast, when connectionist cognitive science abandons the digital computer metaphor, it abandons with it the assumption of centralized serial control. It does so because it views this as a fatal flaw in classical models, generating a “von Neumann bottleneck” that makes classical theories too slow to be useful in real time (Feldman & Ballard, 1982; Hillis, 1985). In the stead of centralized serial control, connectionists propose decentralized control in which many simple processes can be operating in parallel (see Dawson & Schopflocher, 1992a).

Clearly, from one perspective, there are obvious and important differences between connectionist and classical cognitive science. However, a shift in perspective can reveal a view in which striking similarities between these two approaches are evident. We saw earlier that classical cognitive science is performed at multiple levels of analysis, using formal methods to explore the computational level, behavioural methods to investigate the algorithmic level, and a variety of behavioural and biological techniques to elaborate the architectural and implementational levels. It is when connectionist cognitive science is examined from this same multiple-levels viewpoint that its relationship to classical cognitive science is made apparent (Dawson, 1998).

Analyses at the computational level involve using some formal language to make proofs about cognitive systems. Usually these proofs concern statements about what kind of computation is being performed or what the general capabilities of a system are. Computational-level analyses have had a long and important history in connectionist cognitive science, and they have been responsible, for example, for proofs that particular learning rules will converge to desired least-energy or low-error states (Ackley, Hinton, & Sejnowski, 1985; Hopfield, 1982; Rosenblatt, 1962; Rumelhart, Hinton, & Williams, 1986b). Other examples of computational analyses
were provided earlier in this chapter, in the discussion of carving pattern spaces into decision regions and the determination that output unit activities could be interpreted as being conditional probabilities.

That computational analysis is possible for both connectionist and classical cognitive science highlights one similarity between these two approaches. The results of some computational analyses, though, reveal a more striking similarity. One debate in the literature has concerned whether the associationist nature of artificial neural networks limits their computational power, to the extent that they are not appropriate for cognitive science. For instance, there has been considerable debate about whether PDP networks demonstrate appropriate systematicity and componentiality (Fodor & McLaughlin, 1990; Fodor & Pylyshyn, 1988; Hadley, 1994a, 1994b, 1997; Hadley & Hayward, 1997), two characteristics important for the use of recursion in classical models. However, beginning with the mathematical analyses of Warren McCulloch (McCulloch & Pitts, 1943) and continuing with modern computational analyses (Girosi & Poggio, 1990; Hartman, Keeler, & Kowalski, 1989; Lippmann, 1989; McCulloch & Pitts, 1943; Moody & Darken, 1989; Poggio & Girosi, 1990; Renals, 1989; Siegelmann, 1999; Siegelmann & Sontag, 1991), we have seen that artificial neural networks belong to the class of universal machines. Classical and connectionist cognitive science are not distinguishable at the computational level of analysis (Dawson, 1998, 2009).

Let us now turn to the next level of analysis, the algorithmic level. For classical cognitive science, the algorithmic level involves detailing the specific information processing steps that are involved in solving a problem. In general, this almost always involves analyzing behaving systems in order to determine how representations are being manipulated, an approach typified by examining human problem solving with the use of protocol analysis (Ericsson & Simon, 1984; Newell & Simon, 1972). Algorithmic-level analyses for connectionists also involve analyzing the internal structure of intact systems—trained networks—in order to determine how they mediate stimulus-response regularities. We have seen examples of a variety of techniques that can and have been used to uncover the representations that are hidden within network structures, and which permit networks to perform desired input-output mappings. Some of these representations, such as coarse codes, look like alternatives to classical representations. Thus one of classical cognitive science's contributions may be to permit new kinds of representations to be discovered and explored.

Nevertheless, algorithmic-level analyses also reveal further similarities between connectionist and classical cognitive science. While these two approaches may propose different kinds of representations, they still are both representational. There is no principled difference between the classical sandwich and the connectionist sandwich (Calvo & Gomila, 2008). Furthermore, it is not even guaranteed that the
contents of these two types of sandwiches will differ. One can peer inside an artificial neural network and find classical rules for logic (Berkeley et al., 1995) or even an entire production system (Dawson et al., 2000).

At the architectural level of analysis, stronger differences between connectionist and classical cognitive science can be established. Indeed, the debate between these two approaches is in essence a debate about architecture. This is because many of the dichotomies introduced earlier—rationalism vs. empiricism, digital computer vs. analog brain, structure/process vs. dynamic data, serialism vs. parallelism—are differences in opinion about cognitive architecture.

In spite of these differences, and in spite of connectionism’s search for biologically plausible information processing, there is a key similarity at the architectural level between connectionist and classical cognitive science: at this level, both propose architectures that are functional, not physical. The connectionist architecture consists of a set of building blocks: units and their activation functions, modifiable connections, learning rules. But these building blocks are functional accounts of the information processing properties of neurons; other brain-like properties are ignored. Consider one response (Churchland & Churchland, 1990) to the claim that the mind is the product of the causal powers of the brain (Searle, 1990):

> We presume that Searle is not claiming that a successful artificial mind must have all the causal powers of the brain, such as the power to smell bad when rotting, to harbor slow viruses such as kuru, to stain yellow with horseradish peroxidase and so forth. Requiring perfect parity would be like requiring that an artificial flying device lay eggs. (Churchland & Churchland, 1990, p. 37)

It is the functional nature of the connectionist architecture that enables it to be almost always studied by simulating it—on a digital computer!

The functional nature of the connectionist architecture raises some complications when the implementational level of analysis is considered. On the one hand, many researchers view connectionism as providing implementational-level theories of cognitive phenomena. At this level, one finds researchers exploring relationships between biological receptive fields and patterns of connectivity and similar properties of artificial networks (Ballard, 1986; Bankes & Margoliash, 1993; Bowers, 2009; Guzik, Eaton, & Mathis, 1999; Keith, Blohm, & Crawford, 2010; Moorhead, Haig, & Clement, 1989; Poggio, Torre, & Koch, 1985; Zipser & Andersen, 1988). One also encounters researchers finding biological mechanisms that map onto architectural properties such as learning rules. For example, there is a great deal of interest in relating the actions of certain neurotransmitters to Hebb learning (Brown, 1990; Gerstner & Kistler, 2002; van Hemmen & Senn, 2002). Similarly, it has been argued that connectionist networks provide an implementational account of associative learning (Shanks, 1995), a position that ignores its potential contributions at other levels of analysis (Dawson, 2008).
On the other hand, the functional nature of the connectionist architecture has resulted in its biological status being questioned or challenged. There are many important differences between biological and artificial neural networks (Crick & Asanuma, 1986; Douglas & Martin, 1991; McCloskey, 1991). There is very little biological evidence in support of important connectionist learning rules such as backpropagation of error (Mazzoni, Andersen, & Jordan, 1991; O’Reilly, 1996; Shimansky, 2009). Douglas and Martin (1991, p. 292) dismissed artificial neural networks as merely being “stick and ball models.” Thus whether connectionist cognitive science is a biologically plausible alternative to classical cognitive science remains an open issue.

That connectionist cognitive science has established itself as a reaction against classical cognitive science cannot be denied. However, as we have seen in this section, it is not completely clear that connectionism represents a radical alternative to the classical approach (Schneider, 1987), or that it is rather much more closely related to classical cognitive science than a brief glance at some of the literature might suggest (Dawson, 1998). It is certainly the case that connectionist cognitive science has provided important criticisms of the classical approach and has therefore been an important contributor to theory of mind.

Interestingly, many of the criticisms that have been highlighted by connectionist cognitive science—slowness, brittleness, biological implausibility, overemphasis of logicism and disembodiment—have been echoed by a third school, embodied cognitive science. Furthermore, related criticisms have been applied by embodied cognitive scientists against connectionist cognitive science. Not surprisingly, then, embodied cognitive science has generated a very different approach to deal with these issues than has connectionist cognitive science.

In Chapter 5 we turn to the elements of this third “flavour” of cognitive science. As has been noted in this final section of Chapter 4, there appears to be ample room for finding relationships between connectionism and classicism such that the umbrella cognitive science can be aptly applied to both. We see that embodied cognitive science poses some interesting and radical challenges, and that its existence calls many of the core features shared by connectionism and classicism into question.