Burge, T. (1984). Individualism in psychology. *Philosophical Review, 95,* 3–45.
Chalmers, D. J. (1990). Syntactic transformations on distributed representations. *Connection Science, 2,* 53–62.
Chalmers, D. J. (1991). *In and out of the Chinese Room.* Manuscript in preparation.
Dennett, D. C. (1982). Beyond belief. In A. Woodfield (Ed.), *Thought and object* (pp. 1–95). Oxford: Oxford University Press.
Dyer, M. G. (1990). Distributed symbol formation and processing in connectionist networks. *Journal of Experimental and Theoretical Artificial Intelligence, 2,* 215–239.
Fodor, J. A. (1975). *The language of thought.* Cambridge, MA: Harvard University Press.
Fodor, J. A. (1980). Methodological solipsism considered as a research strategy in cognitive psychology. *Behavioral and Brain Sciences, 3,* 63–109.
Fodor, J. A., & Pylyshyn, Z. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition, 28,* 3–71.
Franklin, S., & Garzon, M. (1990). Neural computability. In O. Omidvar (Ed.), *Progress in Neural Networks,* (Vol. 1., pp. 127–145). Norwood, NJ: Ablex.
Harnad, S. (1990). The symbol grounding problem. *Physica D, 42,* 335–46.
Hinton, G. E., McClelland, J. L., & Rumelhart, D. E. (1986). Distributed representations. In D. Rumelhart, J. McClelland, & the PDP Research Group, *Parallel Distributed Processing* (pp. 77–109). Cambridge, MA: MIT Press.
Hofstadter, D. R. (1985). Waking up from the Boolean dream, or, subcognition as computation. In *Metamagical themas* (pp. 631–665). New York: Basic Books.
Holland, J. H. (1986). Escaping brittleness: The possibilities of general purpose machine learning algorithms applied to parallel rule-based systems. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning* (Vol. 2, pp. 593–623). Los Altos, CA: Morgan Kaufmann.
Kaplan, S., Weaver, M. E., & French, R. M. (1990). Active symbols and internal models: Towards a cognitive connectionism. *AI and Society, 4,* 51–71.
Lakoff, G. (1987). *Women, fire, and dangerous things.* Chicago: University of Chicago Press.
Lewis, D. (1970). Psychophysical and theoretical identifications. *Australasian Journal of Philosophy, 50,* 249–258.
Mitchell, M., & Hofstadter, D. R. (1990). The emergence of understanding in a computer model of concepts and analogy-making. *Physica D, 42,* 322–334.
Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry. *Communications of the Association for Computing Machinery, 19,* 113–126.
Penrose, R. (1989). *The emperor's new mind.* Oxford: Oxford University Press.
Pollack, J. B. (1990). Recursive distributed representation. *Artificial Intelligence, 46,* 77–105.
Putnam, H. (1960). Minds and machines. In S. Hook (Ed.), *Dimensions of mind* (pp. 138–164). New York: New York University Press.
Putnam, H. (1975). The meaning of "meaning". In K. Gunderson (Ed.), *Language, mind, and knowledge* (Minnesota Studies in the Philosophy of Science, Vol. 7, pp. 131–193). Minneapolis: University of Minnesota Press.
Rumelhart, D. E., McClelland, J. L., & the PDP Research Group (1986). *Parallel distributed processing.* Cambridge, MA: MIT Press.
Searle, J. R. (1980a). Minds, brains and programs. *Behavioral and Brain Sciences, 3,* 417–424.
Searle, J. R. (1980b). Intrinsic intentionality. *Behavioral and Brain Sciences, 3,* 450–457.
Searle, J. R. (1984). *Minds, brains, and science.* Cambridge, MA: Harvard University Press.
Searle, J. R. (1987). Minds and brains without programs. In C. Blakemore & S. Greenfield (Eds.), *Mindwaves* (pp. 209–233). Oxford: Blackwell.
Searle, J. R. (1990). Consciousness, explanatory inversion, and cognitive science. *Behavioral and Brain Sciences, 13,* 585–596.
Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences, 11,* 1–74.
Turing, A. M. (1950). Computing machinery and intelligence. *Mind, 59,* 433–460.

3 Rules in Programming Languages and Networks

Fred Adams
Kenneth Aizawa
Gary Fuller
Central Michigan University

Debates concerning the relative merits of programming languages and connectionist networks frequently discuss the role of rules in the different computational devices (Fodor & Pylyshyn, 1988; Gasser & Lee, chap. this volume; Horgan & Tienson, 1988, 1990; Lachter & Bever, 1988; Pinker & Prince, 1988; Rumelhart & McClelland, 1986a, 1986b). Among the specific questions that arise are

1. Do models formulated in programming languages use explicit rules where connectionist models do not (Gasser & Lee, chap. this volume; Rumelhart & McClelland, 1986)?
2. Are rules as found in programming languages hard, precise, and exceptionless, where connectionist rules are not (Horgan & Tienson, 1988, 1990; Smolensky, 1988b)?
3. Do connectionist models use rules operating on *distributed representations* where models formulated in programming languages do not (Horgan & Tienson, 1988, 1990; Smolensky, 1988a, 1988b)?
4. Do connectionist models fail to use structure sensitive rules of the sort found in "classical" computer architectures (Fodor & Pylyshyn, 1988; Smolensky, 1988b)?

In this chapter we argue that the answer to each of these questions is negative. We argue that hard, explicit, exceptionless, structure-sensitive rules are only contingently associated with programming languages, such as Turing machines, production systems, LISP, FORTRAN, Pascal, and only contingently excluded

from connectionist models.[1] In other words, although there is some truth in saying that a particular connectionist model does not use explicit rules, or does use soft rules, and so forth, there appear to be other connectionist models that do use explicit rules, do use hard rules, and so forth. Similarly, whereas there are particular programs in programming languages, such as Turing machines, production systems, and so forth, that do use explicit rules, and so forth, there appear to be other programs that do not use explicit rules and so forth. We conclude, therefore, that it is possible to dissociate the various questions concerning rules from the issues separating programming languages and networks of units and connections. Part of what emerges in the course of this chapter is that there are networks that are symbolic processors and networks that are subsymbolic processors, and there are programs that are symbolic processors and programs that are subsymbolic processors (See Dinsmore (chap. 1 in this volume) and Blank, Meeden, and Marshall (chap. 6 in this volume) for more on the symbolic-subsymbolic distinction).

Having found these numerous failed attempts to contrast networks and programming languages in terms of rules, we are moved to question the motivation underlying these attempts. What reason is there to suppose that there is some grand difference between connectionist models and programming language models in their use of rules? Why have people thought it important for there to be greater differences between networks and Turing machines than there are between, say, Turing machines and LISP programs? Why are we not merely being deceived by the fact that units and connections do not look like computer programs?

We should note at the outset that the issue of the use of rules is not to be settled merely by appeal to standard proofs of the equivalence of Turing machines and connectionist networks. These simply show that any Turing-computable function is also computable by a network. They are input–output equivalent. The claims concerning the use of rules are not about differences in the functions computed by the two types of devices, but about differences in the way in which these functions are computed. Programming languages compute functions using explicit rules, where connectionist networks do not.

Our case for this view begins with a brief review of the basics of connectionism and Turing machines. For many purposes, Turing machines will provide a simple means of generating clear counterexamples. If the various *rule-based* distinctions between programming language models and activation propagating/weight changing networks are to be vindicated, at the least, they should separate networks from Turing machines. Such a separation would appear to be a necessary condition for making good the "rule-based" distinction between programming language and connectionist models. After this review, we treat explicit

[1]*Programming language* is here taken in the standard sense from computation theory, for example, what Machtey and Young (1978, chap. 3) referred to as an acceptable programming system.

rules, *soft* rules, rules operating on distributed representations, and rules operating on *structured representations*.

1. THE NUTS AND BOLTS OF CONNECTIONISM AND TURING MACHINES

For a more accessible introduction to connectionism (one whose philosophical analyses we do not necessarily endorse) the reader should refer to the editor's introduction to this volume. Here we offer only a bare bones sketch to fix terminology and specify what we mean by connectionism. Our introduction to Turing machines is somewhat longer than our introduction to connectionism, because we need to remind our readers of a number of features of Turing machines that are important for subsequent discussion.

We describe connectionism in terminology taken from Rumelhart, Hinton, and McClelland (1986). A connectionist model consists of a set of units with positively and negatively weighted connections between them. In the Rumelhart, Hinton, and McClelland vocabulary, each unit has an associated activation value and output. The activation value of unit i determines the output of unit i by a so-called *output function*. In many models, the output of a unit is simply its activation value, so that the output function is merely the identity function. The output of unit i is weighted as it is sent out along various connections and joins with the weighted outputs of other units to determine the net input to other units j. This functional relationship is specified by a *propagation function*.[2] The net input to unit j, possibly in conjunction with earlier activation values of unit j, determines a new activation value for j.[3] This functional relationship is specified by an *activation function*. Thus, units, connections, weights, activation values, outputs, and net inputs, form the basic ontology of connectionism. Their interrelations during their "computational" mode is specified by the activation functions, propagation functions, and output functions. Learning takes place in networks via procedures that specify changes in weights as a function of the results of preceding computations. Learning will not figure in our later discussion so we do not elaborate any further. Notice that we are indifferent here to the particular functional relationships between activation values, and so forth, and the manner in which units are arranged into networks. For us these are all connectionist networks. The Boltzmann machine (Hinton & Sejnowksi, 1986), Hopfield nets (Hopfield, 1982), interactive activation models (McClelland & Rumelhart, 1981; Rumelhart & McClelland, 1982), networks that use the stan-

[2]Actually, Rumelhart, Hinton, and McClelland (1986) used *propagation rule*, but because of the present topic, we choose to change the terminology slightly.

[3]In stochastic models, such as the Boltzmann machine, this net input determines the probability distribution over different possible activation values of j.

dard back-propagation feedforward computation phase (Rumelhart, Hinton, & Williams, 1986), the RAAM model (Blank, Meeden and Marshall, chap. 6 in this volume) are all examples of networks.

A Turing machine may be divided into two distinct parts, a finite state control unit and an infinitely long tape divided into squares. Consider, first, the tape and what appears there. Each square contains a letter from an alphabet over which a Turing machine computes. The alphabet may consist of any arbitrarily chosen finite set of symbols. Two examples of alphabets are the letters in the set {0,1} and the set of English letters along with punctuation marks such as the comma, the period, and the question mark. In addition to the arbitrariness in the choice of alphabet, there is latitude in the choice of language conventions one uses in computations. First of all, the items on the tape may denote nothing at all.[4] More commonly, however, they are assumed to denote, represent, or refer to various things. For example, the number one may be denoted by the string "1," two may be denoted by "11," three may be denoted by "111," and so on. Alternatively, the number one may be denoted by "11," two may be denoted by "1111," three denoted by "111111," and so on.

The finite state control unit has a so-called *read-write head* that scans a single tape square at a time. It also contains a program consisting of a set of instructions. A Turing machine instruction is an ordered four-tuple consisting of

1. a state symbol s_i,
2. a tape symbol t_j,
3. another tape symbol t_k or a direction symbol from the set {L,R}, and
4. a second state symbol s_m.

So, one Turing machine instruction might be "q_5 1 0 q_1," and another "q_{12} 1 L q_3," and another "q_{31} 1 L q_{33}." The first instruction may be read in English to mean, "If the finite state control unit is in state q_5, scanning a '1,' then it should write a '0' and change to state q_1." The second may be read "If the finite state control unit is in state q_{12}, scanning a '1,' then it should move one square to the left and change to state q_3."

[4]We do not here mean simply that it is possible to have names or descriptions for nonexistent objects, such as Pegasus or the golden mountain. That is, of course, true, but it is not our point. We mean that none of the tape symbols, or combinations of tape symbols, need denote; they need not even "purport to denote." For example, when Turing machines are construed as *acceptors* of recursive languages, one may say that they accept a type of string in virtue of accepting an instance of the string. So, a Turing machine may accept the string type "000" in virtue of accepting a token of the string "000" written on its tape. In this account, no mention is made of the denotation, or reference, or meaning relation. It uses the instantiation relation; something obviously distinct from the denotation, reference, or meaning relation. For example, Descartes is an instance of a philosopher, but he does not denote (refer to, mean) a philosopher. We, thus, reject Pylyshyn's maxim "No computation without representation" (Pylyshyn, 1984, pp. 62f).

It is well known that the mathematics of Turing machines can, in principle, describe a great diversity of physical, chemical, and biological processes. Here it is perhaps worth noting an important caution this fact suggests, namely, that we must not construe such phrases as *read-write head* and *symbol manipulation* too narrowly. To describe an object as the read-write head of a Turing machine, one need not mean that the object literally writes, as with pen and paper or with laser printing technology. Nor should one think of symbol manipulation as necessarily akin to something like having the finite state control unit lifting labelled blocks or tickets from one square of the tape and carrying them to another (cf. Feldman, 1981, pp. 53–54). Symbol manipulation can describe such processes, but it may describe others. For example, if one were to suppose that the brain is a Turing machine, a biologically inspired suggestion as to the instantiation of, for example, the 0s and 1s of a Turing machine tape would be that the quiescence of a neuron corresponds to a particular square of the tape having a "0" in it, while the firing of the neuron could correspond to the square having a "1" in it. A biologically inspired suggestion as to the instantiation of a larger alphabet might have different firing frequencies corresponding to different letters. Although this picture of the instantiation of Turing machines may sound vaguely like connectionism, certainly some such biological instantiation of programming language ideas must have been in the back of investigators' minds all along.

Here we suppose that computation theory and Turing machines provide a mathematical theory that may be used to describe various physical objects. These physical objects may be electronic, chemical, biological, or whatever. This is analogous to the way in which one might suppose that differential geometry or a particular set of geometrical objects might be used to describe various sorts of masses distributed throughout a spacetime. As with computation theory and Turing machines, the objects described by differential geometry may have various electronic, chemical, or biological properties. In general practice, scientists and philosophers do not bother to distinguish the mathematical apparatus for describing the physical world and the physical world. Thus, spacetime is often said to be a differentiable manifold or an object is said to be a point mass. This casual manner of speech is generally unproblematic, but because it is potentially misleading here, we draw attention to it. This much said, we add that we adopt a similar treatment of connectionism. Units, connections, weights, and so forth, are mathematical objects that may be used to describe various physical, chemical, or biological objects. Thus, we take the usual assumption that units and connections are somehow instantiated by neurons or clusters of neurons to be an assumption over and above the assumptions concerning the mathematics that will ultimately describe (sub)cognitive processing. Although this may seem surprising to some, it in fact appears to be a view tacitly held by important practitioners of connectionism.

2. EXPLICIT RULES

With technical matters aside, let us consider one possible rule-based line of demarcation between connectionist networks and Turing machines. Sometimes *rule* is used to mean simply an input–output mapping. Rumelhart and McClelland (1986b) used rule in this sense at one point, when they speak of the "rule of 78." Lachter and Bever (1988, pp. 196, 216) also described a related sense when they speak of a rule as a physical object that computes an input–output mapping. This, however, is not the sense of rule at issue. When one asserts that a Turing machine uses explicit rules where a network does not, one might mean that there is something in a Turing machine that *instantiates an instruction,* where there is nothing in a connectionist model that does. More specifically, there are instructions in the Turing machine finite state control unit, but there is nothing like an instruction in a network. We are not sure if any connectionist explicitly holds such a view. We consider it primarily for the sake of completeness and expository clarity.

An instruction, here, means something like the primitive changes that are to be made as defined by the basic ontology of a computational device. But, it is clear that connectionist models specify primitive changes in the basic ontology. They are the equations relating activation values, outputs, and weights. Standard back-propagation learners, for example, use instructions such as the following:

1. if unit i is an input unit and the net input is 0, then the activation value of unit i is 0,
2. if unit i is a hidden unit or an output unit and the net input is x, then the activation value of unit i is $1/(1 + \exp(-x))$, and
3. if unit i is an input unit or a hidden unit and its activation value is y, then the output of unit i is y.

Clearly these instructions govern the behavior of standard back-propagation networks during their computation phase, just as four-tuples, such as "q_0 1 L q_3" and "q_3 0 1 q_8," govern the behavior of Turing machines, and whereas the back-propagation instructions differ from Turing machine instructions, Pascal instructions and FORTRAN instructions differ from Turing machine instructions and are nonetheless instructions.

Here it may be thought that network equations specify something like laws of nature governing the interactions of activation values and outputs, where the instructions in the finite state control unit of a Turing machine can be multiply instantiated in a great diversity of physical, chemical, and biological objects. But, as was mentioned in the previous section and as many advocates of connectionism realize, the basic ontology of networks consists of the mathematical units, connections, and so forth, which may also be used to described things constructed from something other than neurons or clusters of neurons.

Consider, now, a second possibility, the possibility we think most investigators have in mind. Let us admit that both networks and Turing machines instantiate instructions, that the four-tuple instructions of Turing machines are comparable to the equations relating activation values, outputs, weights, and net inputs, but in the face of this maintain that nets do not consult rules in their processing like programming language models. To use an explicit rule is, thus, to consult a rule. This seems to be the view found in Rumelhart and McClelland (1986a, 1986b), Davies (1990), and Gasser and Lee (chap. 8 in this volume). We might clarify this idea using Turing machines. A physical object that corresponds to the program of a Turing machine instantiates a set of instructions. An explicit rule, however, is instantiated by a physical object that corresponds to the data that appears on the tape of a Turing machine. A Turing machine with a program for checking the spelling of English words that has the string of symbols, "*i* before *e*, except after *c*," written on its tape might be said to consult a rule in the course of its computation. Of course, not every piece of data that appears on the Turing machine tape constitutes a rule. Consider a Turing machine that computes over the alphabet {0,1}, that uses the language convention that "1" denotes one, "11" denotes two, "111" denotes three, and so on, and begins its computations on the leftmost "1," if there is one, in a finite string of 1s surrounded by 0s. If this Turing machine has the program,

$$s_0\ 0\ 1\ s_1$$
$$s_0\ 1\ L\ s_2$$
$$s_2\ 0\ 1\ s_3,$$

for computing the successor function, $S(x) = x + 1$, it evidently does not consult rules written on its tape when computing $S(x)$. Neither "$S(x) = x + 1$," nor an equivalent, appear on the tape. All that appears on the tape are unary representations of numbers. Rules are part of the data a program uses in order to process information, whereas instructions process the information. To use a rule, that is, to consult a rule, is thus to use a certain type of data.

This account brings us to the truth in the connectionist's claims that particular models do not use explicit rules. A number of connectionist models, such as the Rumelhart–McClelland past-tense model (Rumelhart & McClelland, 1986b) and the Gasser–Lee phonological model only instantiate instructions (Gasser & Lee, chap. this volume), but do not use explicit rules in the sense that they do not consult rules. The data they receive as input, the input pattern of activation, do not constitute rules such as *i* before *e*, except after *c*. Instead, the data consist of codings of such things as a representation of the present tense of English verbs. The data are, thus, like that for the Turing machine that computes the successor function, rather than our spellchecker. It codes things other than rules.

Although we believe that Rumelhart and McClelland, Gasser and Lee, and Davies are correct in asserting that particular connectionist models do not consult rules, our point is to specify the significance of this observation. It is important to

note that just as some Turing machines may consult explicit rules where others do not, so some networks may consult explicit rules where others do not. There is no reason to suppose that one cannot produce a large network with a module that takes as input a pattern of activation constituting a rule. That is, one can feed patterns of activation representing *i* before *e*, except after *c* into networks. Exactly how and why one might produce a network of this sort is an open question, but there is nothing in the connectionist literature that justifies what would be an astonishing discovery, namely, that the class of all networks cannot use explicit rules in this sense.[5,6]

We have approached this last sense of using explicit rules, the sense of consulting a rule, from a somewhat technical direction. It is, however, possible to approach it from a more intuitive direction as well. We have found what we take to be this more intuitive approach in (Rumelhart & McClelland, 1986b, p. 217), as well as in conversations. It seems natural to say that humans differ from rocks and planets in an important way. Rocks and planets may be governed by Newton's laws, but they do not follow or use a representation of Newton's laws to determine how they shall move. Humans, in contrast, often appear to use representations of rules, such as, "When driving a car, merge right after passing only after you see the headlights of the passed car in the rear-view mirror," or "In English spelling, *i* before *e*, except after *c*."[7] As commonly as this view is heard, it is evidently a nonstarter for the desired characteristic difference between connectionist models and programming language models. It would seem that no advocate of connectionism would wish to say that it is a good thing that connectionist models as a class do not use explicit rules in just the way rocks and planets do not use explicit rules. That would ensure that networks are necessarily like inanimate objects, rather than humans. The reasonable theorist would seem to want to maintain, as we do, that although there are some networks that do not use explicit rules, there are some that do.

3. HARD, PRECISE, EXCEPTIONLESS RULES

The preceding section considered distinguishing connectionism and programming language models in terms of the use of explicit rules. An alternative rules

[5]We would point out that if PDP networks can use explicit rules in this sense, this undercuts the motivation Davies (1990) offered for his *intermediate* theory of rules.

[6]It might be thought here that we have made the connectionist claim too strong. Rather than suppose the connectionist to believe that nets cannot use rules, we should take them to believe that nets need not use explicit rules. This claim, however, would apply equally to Turing machines. Turing machines, like nets, need not use explicit rules. Our example of the Turing machine for computing the successor function is but one example. This alteration, therefore, does not save the connectionist's position.

[7]The rule is, of course, not strictly correct as evidenced by, say, *weigh* and *neighbor*.

approach is to admit that both types of computing device use rules, but they differ in the types of rules used. In this section we examine the connectionist's claim that programming language models use "hard," precise, and exceptionless rules, where connectionist models use "soft" rules (For example, Horgan and Tienson, 1988, 1990; Smolensky, 1988a, 1988b).

To begin, we cannot accept Smolensky's (1988b, p. 138) suggestion that *soft* means stochastic and *hard* means deterministic and that the difference between Turing machines and connectionist networks is that nets are stochastic, whereas Turing machines are deterministic. This is, first, because there are both deterministic and stochastic networks; backpropagation learners, interactive activation nets, and perceptrons, for example, are deterministic, whereas Hopfield nets, the Boltzmann machine, and Harmony theory models are stochastic.[8] The stochastic models are sometimes treated as Markov random fields. Moreover, we observe that it is possible to define both nondeterministic and probabilistic Turing machines alongside deterministic Turing machines. A nondeterministic Turing machine is like a deterministic Turing machine save for the fact that it is permitted to have two or more instructions beginning with the same state symbol s_i and the same tape symbol t_j. Such nondeterministic computing devices are at the heart of the vast computer science literature on NP-completeness. A probabilistic Turing machine is, then, like a nondeterministic Turing machine, but each instruction has a fifth element specifying the probability of taking that instruction when given the chance. Probabilistic Turing machine instructions might then look like this:

$$q_0\ 1\ L\ q_3\ 0.25$$
$$q_0\ 1\ R\ q_3\ 0.25$$
$$q_0\ 1\ R\ q_4\ 0.50.$$

When the Turing machine with these instructions is in state q_0 scanning a 1, it will move left and go into state q_3 25% of the time, it will move right and go into state q_3 25% of the time, and it will move right and go into state q_4 50% of the time.

We also challenge Smolensky's suggestion that *soft* means *fuzzy*, as in fuzzy logic (Zadeh, 1965, 1975, 1983), and in this sense connectionist systems are soft (Smolensky, 1988b, p. 139). Although some networks, such as say the interactive activation models, may be treated as fuzzy models, McCulloch–Pitts nets are not fuzzy. Similarly, whereas the Turing machine programs one usually thinks of for computing number theoretic functions do not use fuzzy values, there are Turing machines programs that could work with them.

Another sense of hard rules versus soft rules may be found in the distinction between *constraints* and *costs* (Hinton & Sejnowski, 1986, p. 284). Here the

[8]We might note that it is possible to generate stochastic networks that use standard backpropagation or perceptron convergence learning, as is done in Rumelhart and McClelland (1986).

distinction is between rules that must be satisfied and those that are usually good or helpful to satisfy. There is a difference between rules that must be obeyed and those that it is in some sense usually better to obey. The distinction may be made using an example from chess. A hard rule, or hard constraint, is a rule that must always be obeyed, for example, that a rook cannot move diagonally. Another hard rule is that the King may not move into check. A soft rule, a rule of thumb that is usually wise to follow, is that one ought to attempt to control the center of the board. Another soft rule is that it is wise to get your pieces out early. Horgan and Tienson (1988, 1990) suggested this sense of soft rules when they suggested that the cognitive processing found in professional basketball players involves the simultaneous satisfaction or consideration of multiple soft constraints.

It is worth noting that the distinction between rules of the game and rules of strategy, if you will, is not the same as the difference between rules with exceptions and rules without exceptions. There may, for example, be rules of strategy that are always, without exception, good rules. Perhaps making one's first move taking the center square is such a rule in tic-tac-toe.

In order to investigate the previous suggestion, we should see how it is implemented in some connectionist models. In nets, the weights between units may be thought of as weak constraints between two hypotheses. Many network models might serve to illustrate this point, but we will consider the principles of interactive activation and a concrete example of its application, the interactive activation model of letter perception (McClelland & Rumelhart, 1982; Rumelhart & McClelland, 1986b). In *interactive activation theory*, the activation values of units may be considered a sort of measure of the credibility of an hypothesis, the credibility of the hypothesis associated with the unit.[9] The equations relating activation values, outputs, and weights are such that, if unit *i* is connected to unit *j* with a positive weight, a large activation value for unit *i* will support a large activation value for unit *j*. In other words, a high degree of credibility for the hypothesis associated with unit *i* will confer a high degree of credibility on the hypothesis corresponding to unit *j*. The equations are also such that if unit *i* is connected to unit *j* with a negative weight, a large activation value for unit *i* will support a small activation value for unit *j*. In other words, a high degree of credibility for the hypothesis corresponding to unit *i* confers a low degree of credibility for the hypothesis corresponding to unit *j*. A small activation value for unit *i* means that it will have little effect on *j* either way.[10] In the interactive activation model, all the constraints are weak in the sense that all weights are finite, and whereas the credibility of hypothesis *i* supports hypothesis

[9]The measure of credibility embodied in the activation value is not always a probability measure insofar as it sometimes ranges from −0.2 to 1, as in the interactive activation model of letter perception.

[10]We might note that there is a slight interpretive problem with this theory when we come to negative activation values of the sort found in the model of letter perception, because the hypotheses with negative credibility will end up supporting hypotheses with which they are inconsistent.

j, the credibility of *j* may be undermined by other hypotheses, so that although supports *j*, *j* does not end up being very credible. The constraints in the netwo (e.g., if the hypothesis associated with unit *i* is credible and consistent with th hypothesis associated with unit *j*, then hypothesis *j* should be credible) are soft the sense that, while this constraint is actually true and useful in coming to conclusion about the hypothesis associated with unit *j*, the weights and activatio values with units *k*, *l*, *m*, and *n*, may lead to a low activation value for unit *j*, that unit *j* is not credible.

We might make these ideas more concrete by appeal to McClelland ar Rumelhart's model of letter perception. In this model, there are three levels units: *feature* units, *letter-position* units, and *word* units (See Fig. 3.1). Thus, the lowest level, there are units that register the presence of parts of letters, suc as the lower left leg of the letter *A*. The set of possible features and ways in whic they may be organized into letters is shown in Figs. 3.2 and 3.3. At a high level, the letter-position level, there is a unit representing the letter *A* occurring the first position in a word being scanned, a unit representing the letter *B* at th first position in a word being scanned, and so forth for each letter. There is similar array of 26 units for each of the four positions in a fixed visual field. the highest level, the word level, there is also a unit for the word *read*, a unit f

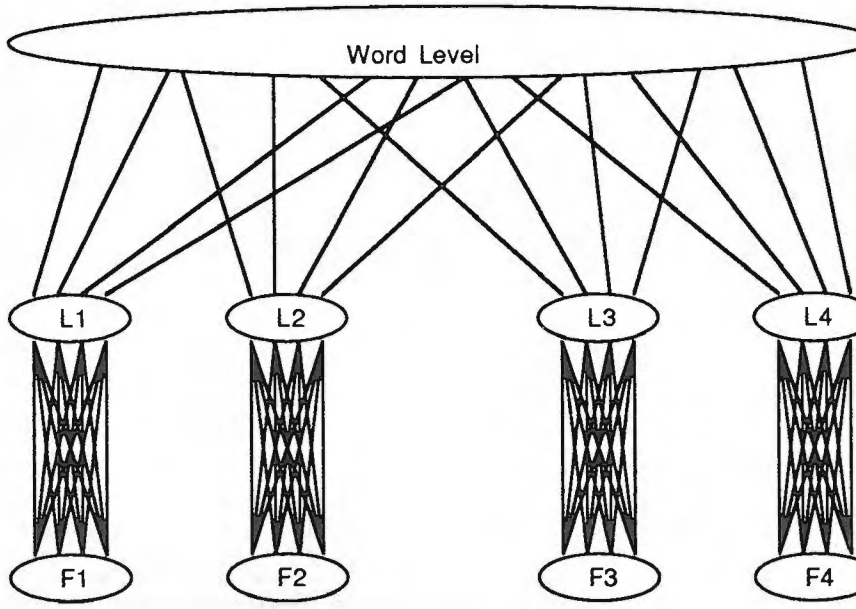

FIG. 3.1. General structure of the Interactive Activation Model of Letter Perception.

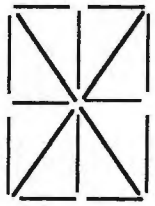

FIG. 3.2. The set of features constituting letters in the Interactive Activation Model of Letter Perception.

the word *bead,* and so forth, for various four-letter words of the English language. In this model, there is a positively weighted connection between the unit corresponding to *B* at the first position and the unit corresponding to the word *bead,* and a negatively weighted connection between the unit corresponding to *B* at the first position and the unit corresponding to the word *read.* Each of the weights in an interactive activation model, such as the model of letter perception, constitutes a soft constraint. If there is something corresponding to the lower left leg of the letter *A* in the first position of the visual field, this makes it more credible to suppose that the first letter in the visual field is an *A.* So, the net has the constraint: If there is something in the position of the lower left leg of the letter *A* in the visual field, then the object in the visual field is an *A.* This constraint is soft, because it may be outweighed by the other feature units. Thus, if the two units indicating the line along the bottom of a *B* both have high activation value, then their negative weights to the letter-position unit *A* may outweigh the single unit for the *A* letter-position unit, and the model will not believe that there is a letter *A* at the first position in the visual field.[11]

We think this theory of soft constraints makes perfect sense, but again this difference does not fall along the line between Turing machines and connectionism. Existing chess-playing programs use soft constraints when they take into account various features of a situation in order to decide on a move. Turing machines that use the same system of evaluating moves would, then, presumably use the same soft constraints as the available chess-playing programs. Perhaps the important general point to make is that there is no inconsistency between determinism and the use of soft rules. The interactive activation model is deter-

[11]We note in passing that Horgan and Tienson suggested that programming languages cannot be used to model the play of professional basketball players because of "the need to deal simultaneously with numerous constraints and the resultant threat of computation explosion. Anything having to do with basketball might appropriately be a factor, and it is at least a practical impossibility to list all of the factors." The passage appears in the context of the difficulty of the frame problem. It is not at all clear, however, that the existing connectionist constraint satisfaction picture offers a solution to the problems. No reason is given for thinking that, whereas there is not enough time to take all the relevant constraints into account, there are enough brain cells to take all the relevant constraints into consideration. In other words, what reason is there to think an explosion in *spatial* complexity is any less a problem than an explosion in *time* complexity?

FIG. 3.3. Sample letters for the Letter Perception Model.

ministic, but it uses soft rules or soft constraints, so the mere fact of the determinism of a Turing machine does not preclude the use of soft constraints.

Another way of drawing attention to the fact that soft constraints may be implemented in conventional computing devices is to note the large literature in nonmonotonic reasoning (See, for example, Kimbrough & Adams, 1988, for numerous references). In work on nonmonotonic reasoning, the principal idea is to develop an inferential system that accommodates the defeasibility of various inferences.

It might be objected, here, that we still have not correctly interpreted the claims about the soft nature of connectionist inferences. One might claim that while Turing machines fundamentally use hard constraints, but can be programmed to use soft constraints, networks are fundamentally soft. That is, the fundamental operations of programming language models are exceptionless, where the fundamental operations of networks are not exceptionless. This may be what Smolensky had in mind when he talked of traditional expert systems that use many, many lines of code in such a way that "softness emerges from hardness," where in networks, "the cognitive system is fundamentally a soft machine that is so complex it sometimes appears hard when viewed at higher levels" (Smolensky, 1988b, pp. 139–140). This response, however, is belied by the observation that the relations between the activation values, outputs, and weights in existing network models are as hard, exceptionless, and precise as you please. The equations for backpropagation learners introduced earlier, for example, clearly show this. Interactive activation and perceptron equations also show this. Note that even taking *fundamentally soft* to mean *fundamentally probabilistic* does not work for the reasons cited earlier: Both connectionist models and Turing machines admit of probabilistic and deterministic strains.

4. RULES OPERATING ON DISTRIBUTED REPRESENTATIONS

Advocates of connectionism frequently assert that their models use distributed representations, often with the suggestion that programming language models do not. One might then say that both connectionist models and programming language models use rules, but the rules in connectionist models operate on distributed representations, where the rules in programming language models oper-

ate on localist representations. The phrase *distributed representations* has a number of meanings. Four of these are relevant to the present discussion. We discuss these in turn. First, representations may be said to be distributed in the sense that many units are used in the representation of a concept from ordinary language discourse. The concept of a chair, for example, might involve the activation values of a set of units, rather than only a single unit. Although it is true that various connectionist models use distributed representations in this sense, it would be a mistake to conclude that this is an essential characteristic of connectionism. There are obviously models that use localist representations in the sense that for each concept or hypothesis the model might be said to possess, there is a unit. In the interactive activation model of letter perception described earlier, there are single units for words such as *read* and *bead*. Each of these units corresponds to a proposition such as *The word "read" was seen* with the activation value of the unit constituting a sort of measure of the confidence in the hypothesis. This sort of understanding is implicit in the very theory of interactive activation.

From well-known examples, it is clear that there are networks that use both localist and distributed representations in the sense of using many units per concept or proposition. It is also clear that Turing machines can use the equivalent of distributed or localist representations, that is, several tape squares per concept or proposition. Distributed representations in Turing machines have markers in numerous tape squares to denote objects or stand for propositions. For example, suppose we wish to have a representation of chairs. One way to prepare a Turing machine to have these representations would be to use a Turing machine that computes over the standard alphabet of English with a few punctuation marks thrown in, that is, {A, a, B, b, . . . , ?, ,, ., "}. To perform computations concerning chairs the Turing machine might manipulate strings consisting of a *c*, an *h*, an *a*, an *i*, and an *r*, where *c*, *h*, *a*, *i*, and *r* each appear in a distinct tape square. Propositions could be given a similar treatment with sentences written in the tape squares. These sorts of representations would apparently count as *distributed Turing machine representations*. A Turing machine using a localist representation might compute over a different alphabet, for example, letters from the set {dog, cat, house, chair, boy, girl, . . . }. Each of these representations is a letter in the sense that each occupies only one tape square. Propositions might be treated similarly using sentences. Thus, whereas particular connectionist models may use rules defined over distributed, rather than localist, representations it is also possible to have Turing machines that use either distributed or localist representations in the sense sketched earlier.

Second, *distributed representations* at times, means something like *redundant representations*. In the Rumelhart–McClelland past-tense model, for example, *went* is represented by many different real-valued activation values of a set of output units. Turing machines, however, can also avail themselves of redundant representations. To take a simple example, it is possible to write a computer

program that will compute the successor function given the language conven that "1," "11," and "111" denote one, that "1111," "11111," and "11111 denote two, and so on.[12]

Third, sometimes one means by distributed representations, representation which the same units and activation values can be used to represent diffe objects. Thus, for example, an activation value of 0.95 on unit *i* could be par the representation of the color of a cat when it is part of one pattern of activat and then later be part of the representation of the weight of an automobile whe is part of another pattern of activation. The distinct representations are t *superimposed* on the same activation value for the same unit.

Granted there can be this sort of superimposition. But it should also granted that nets need not have this sort of superimposition. The activat values of units can be assigned meanings in such a way as not to vary v context. So much for connectionism. Turing machines, likewise, can have let whose denotations do not vary with their context. But, of course, the letters vary from context to context. For example, the letter *a* has no meaning of its o in certain contexts—it means *not* in its first occurrence of *atypical* but nothin all in its second occurrence. Larger strings of symbols have this property as w The string *bank* could be given different meanings in different contexts a p gramming language uses.

Finally, representations may be said to be *spatially distributed* that is, symbols are spread diffusely through space. In connectionism, the neurons se ing to support a mental representation are spread throughout the cortex in suc way that a lesion of a small cluster of cells will not completely obliterate representation, hence causing loss of memory about the object represented. T

[12]A program for doing this is the following:

$s_0\ 0\ 1\ s_1$
$s_0\ 1\ R\ s_2$
$s_2\ 0\ 1\ s_4$
$s_2\ 1\ R\ s_3$
$s_3\ 0\ 1\ s_9$
$s_3\ 1\ R\ s_0$
$s_4\ 1\ R\ s_5$
$s_5\ 0\ 1\ s_6$
$s_6\ 1\ R\ s_7$
$s_7\ 0\ 1\ s_8$
$s_9\ 1\ R\ s_{10}$
$s_{10}\ 0\ 1\ s_{11}$.

The read-write runs across the input string of 1s noting blocks of three 1s. When it comes acr block of less than three 1s, it rounds the total number of 1s up to an even multiple of three, then a single additional 1.

sense of distributed representations, representations spatially spread out, does not contrast networks with programming languages, because the object described with the mathematics of Turing machines can be diffused through space just as easily as can units, or the neurons underlying units. There is no reason to suppose that neurons underlying a "1" on a tape square cannot be spread throughout the cortex. This sense of distributed representations is perhaps a less popular means of contrasting networks and programming languages than the senses considered earlier, but we include it for completeness.

For these reasons, we believe there is no difference between the class of Turing machines and the class of connectionist networks when it comes to rules that apply to distributed versus localist representations.

5. STRUCTURE-SENSITIVE RULES

It might be thought that Fodor and Pylyshyn (1988) put forth a view that is at odds with the sort of view we have so far developed, namely, that connectionist models do not use rules sensitive to the structure of representations, where programming language models do. This way of putting things is not entirely satisfactory. The Fodor–Pylyshyn view is a bit more subtle, and is in fact very much in keeping with the view we have taken of other purported contrasts between programming languages and connectionism. Fodor and Pylyshyn do not reject the idea that what takes place in human brains may involve something like the activation propagation and weight change processes described in section 1. Rather, they take exception to a particular view of mental representation that has contingently grown up around networks and conflicts with a way of using Turing machines and their kin, those that have what may be called *classical cognitive architecture*. By a *classical* architecture, Fodor and Pylyshyn meant a computing device working with certain constraints on the way symbols, or markers, are manipulated and on the way the manipulated markers denote objects. In other words, Fodor and Pylyshyn did not offer a distinction between activation propagating/weight changing devices on the one hand and programming languages on the other.

According to Fodor and Pylyshyn, a computing device with a classical architecture has (a) representations with a combinatorial syntax and (b) rules that are sensitive to that combinatorial syntax. In Turing machines, for example, (a) amounts to the assertion that tape symbols are organized in such a way that all representations are either atomic or built up by combinations of atomic symbols. Thus, one may be denoted by "1," two by "11," and so on, with representations of larger numbers built up from representations of smaller numbers. Claim (b) amounts to the assertion that the instructions in the finite state control unit are sensitive to the way in which the symbols are organized on the tape. Thus, it matters to the computation of the successor function how symbols fill the tape

squares. If the language conventions a program presupposes are not used, then the program may compute a different function. So, if we change one of the language conventions mentioned earlier so that "111" denotes four rather than three, then a program for computing successor that adds a single "1" to the left of the string "11" will not compute the successor of two, but the successor of three.

It is clear from the following passage that Fodor and Pylyshyn (1988) did not believe that activation propagation and weight changing models are inconsistent with a classical architecture,

> But we are not claiming that you can't reconcile a Connectionist architecture with an adequate theory of mental representation (specifically with a combinatorial syntax and semantics for mental representations). On the contrary, of course you can: All that's required is that you use your network to implement a Turing machine, and specify a combinatorial structure for its computational language. (p. 28)

According to Fodor and Pylyshyn, if a network were outfitted with a classical architecture then it would have the resources they believe necessary in order to account for what they call productivity, systematicity, compositionality, and inferential coherence.

So, it is fairly clear on inspection, that Fodor and Pylyshyn thought there is no contradiction in speaking of an activation propagating/weight change network that has a classical architecture. Moreover, it is also rather clear that Fodor believed that there can be Turing machines that do not use classical architecture. He discussed these in Fodor (1987). There he distinguished between representational theories of mind that postulate a language of thought from representational theories of mind that reject a language of thought.

6. CONCLUSION

In this chapter we have surveyed the leading attempts to draw a contrast between the class of programming language models and the class of network models in terms of the use of various types of rules. In each case, we have found the purported distinction to be spurious. We have not, of course, proved that there can be no such distinction; that was not our aim. Nor was it our aim to show that there is no difference at all between networks and programming languages. Our point has been merely that existing explications of the differences between nets and programming languages made in terms of rule use are inadequate.

It is, of course, inevitable that someone will try to invent a new characterization of the difference between nets and programming languages in terms of rules, but we wish to discourage such attempts. First, we note that it may or may not be possible to come up with an accurate rule-based distinction between nets and

programming languages. In either case, however, there remains a further hurdle that must be overcome. If one finds a difference between networks and programs in terms of rules, we should like to know why it is a difference that makes a difference. What empirical, rather than purely philosophical, reason is there to suppose that a purported difference is not a trivial difference and in fact a key to understanding what takes place in the human brain? Why would a difference in rule use be an empirical difference that makes a difference? Second, we observe that much of the philosophical work on networks and artificial intelligence, by both cognitive scientists and by philosophers, has been preoccupied with efforts to draw deep distinctions between networks and programming languages. The appeal to talk of rules constitutes one such attempt. We must, however, ask ourselves why deep distinctions are necessary here. Why suppose that there is some great divide between networks and, say, LISP programs on the one hand, yet no great divide between LISP programs and production systems? If there need be no such divide, then perhaps there need be no rule-based distinction.

REFERENCES

Davies, M. (1990). Knowledge of rules in connectionist networks. *Intellectica, 9–10,* 81–126.

Feldman, J. A. (1981). A connectionist model of visual memory. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel model of associative memory* (pp. 49–82). Hillsdale, NJ: Lawrence Erlbaum Associates.

Fodor, J. A. (1987). *Psychosemantics*. Cambridge, MA: MIT Press.

Fodor, J. A., & Pylyshyn, Z. (1988). Connectionism and cognitive architecture: A critical analysis. In S. Pinker & J. Mehler (Eds.), *Connections and symbols* (pp. 3–71). Cambridge, MA: MIT Press.

Gasser, M., & Lee, C. (This volume). Where do "underlying representations" come from?

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group, (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1, pp. 282–317). Cambridge, MA: MIT Press.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, 79,* 2554–2558.

Horgan, T., & Tienson, J. (1988). Settling into a new paradigm. In T. Horgan & J. Tienson, (Eds.), *Spindel Conference 1987: Connectionism and the philosophy of mind* (pp. 97–114). Memphis, TN: Department of Philosophy, Memphis State University.

Horgan, T., & Tienson, J. (1990). Representations without rules. *Philosophical Topics, 17*(1), 147–174.

Kimbrough, S. O., & Adams, F. (1988). Why nonmonotomic logic? *Decision Support Systems, 4,* 111–127.

Lachter, J., & Bever, T. G. (1988). The relation between linguistic structure and associative theories of language learning: A constructive critique of some connectionist learning models. In S. Pinker & J. Mehler (Eds.), *Connections and symbols* (pp. 195–247). Cambridge, MA: MIT Press.

Machtey, M., & Young, P. (1978). *An introduction to the general theory of algorithms*. New York: North Holland.

McClelland, J. L., & Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: Part I. An account of basic findings. *Psychological Review, 88,* 375–407.

McClelland, J. L., Rumelhart, D. E., & Hinton, G. E. (1986). The appeal of parallel distributed processing. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1, pp. 3–44). Cambridge, MA: MIT Press.

McClelland, J. L., Rumelhart, D. E., & the PDP Research Group (1986). *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 2). Cambridge, MA: MIT Press.

Pinker, S., & Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. In S. Pinker & J. Mehler (Eds.), *Connections and symbols* (pp. 73–193). Cambridge, MA: MIT Press.

Pylyshyn, Z. (1984). *Cognition and computation*. Cambridge, MA: MIT Press.

Rumelhart, D. E., Hinton, G. E., & McClelland, J. L. (1986). A general framework for parallel distributed processing. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1, pp. 45–76). Cambridge, MA: MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1, pp. 318–362). Cambridge, MA: MIT Press.

Rumelhart, D. E., & McClelland, J. L. (1982). An interactive activation model of context effects in letter perception: Part 2. The contextual enhancement effect and some tests and extensions of the model. *Psychological Review, 89,* 60–94.

Rumelhart, D. E., & McClelland, J. L. (1986a). PDP models and general issues in cognitive science. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1, pp. 110–146). Cambridge, MA: MIT Press.

Rumelhart, D. E., & McClelland, J. L. (1986b). On learning the past tense of English verbs. In J. L. McClelland, D. E. Rumelhart, & the PDP Research Group. *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 2, pp. 216–271). Cambridge, MA: MIT Press.

Rumelhart, D. E., McClelland, J. L., & the PDP Research Group (1986). (Eds.). *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1). Cambridge, MA: MIT Press.

Smolensky, P. (1988a). On the proper treatment of connectionism. *Behavioral and Brain Sciences, 11,* 1–74.

Smolensky, P. (1988b). The constituent structure of connectionist mental states: A reply to Fodor and Pylyshyn. In T. Horgan & J. Tienson (Eds.), *Spindel conference 1987: Connectionism and the philosophy of mind* (pp. 97–114). Memphis, TN: Department of Philosophy, Memphis State University.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, 8,* 338–353.

Zadeh, L. A. (1975). Fuzzy logic and approximate reasoning. *Synthese, 30,* 4-7–428.

Zadeh, L. A. (1983). Role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy sets and systems, 11,* 199–227.