1

Introduction

Psychological development is a difficult process to understand. The basic problems in understanding how children represent knowledge and how these representations change over development have been with us for a long time. Despite a century of intriguing scientific evidence on child development, comprehensive theoretical understanding has remained elusive. Part of the reason for this is that the problems of psychological development are too complex for traditional verbal theories of development. One of the main points of this book is that considerable leverage on these problems can be gained by applying computational modeling to developmental phenomena. This is because computational modeling is a good way to capture complex processes, and such models can be examined in detail to discover insights into the phenomena that they simulate. In short, this book presents an argument for a new subfield of developmental psychology, *computational* developmental psychology.

The present chapter sets the stage by reviewing the main issues in psychological development and justifying the use of modeling in the study of such issues, the use of computational models, and the use of neural-network models in particular. Along the way, it is essential to describe the principal features of artificial neural networks.

Issues in Psychological Development

Although there are several possible takes on issues in psychological development, there is wide agreement among developmental researchers that the primary issues concern the *what* and the *how* of development. What is it that develops, and how does it develop? The distinction

between these two issues has often been discussed in terms of the difference between structure and transition. What are the principal structures that develop at each particular age or stage, and what is the transition mechanism that moves a person from one stage to the next? These primary developmental issues go back a long way. Aristotle, for example, discussed the distinction as the difference between *being* and *becoming*.

Contemporary cognitive science provides a more precise way of discussing the issue of what develops. Cognition can be analyzed as a distinction between representation and processing (Thagard, 1996). How is knowledge represented, and what processes occur over those representations?

Further unpacking of the transition issue also leads to a number of secondary issues. How do innate and experiential determinants operate in producing development? How is it possible for anything genuinely new be learned? What is the relation between learning and development?

Because stages are often mentioned in discussions of development, particular other issues arise. Is development continuous or discontinuous? To the extent that it is discontinuous, why are there plateaus or stages in psychological development? What accounts for the particular orders of stages? In what sense can there be developmental precursors of psychological stages? Why is there a prolonged period of development? And why does psychological development slow and eventually stop?

Because of the apparent importance of learning in psychological development and the suspicion that current knowledge affects the course of new learning, a number of other issues arise. How does current knowledge affect new learning and development? What happens to old knowledge after new acquisitions?

Concerning such issues, most developmental researchers would agree on two points: first, that these issues are important to resolve, and second, that currently there is no good agreement on their proper resolution. As this book adopts an issue-oriented approach, I attempt to sketch answers to these issues in the course of the book. The issues are sufficiently deep and longstanding that definitive resolutions remain elusive, but the point is that these issues can be made more tractable from the perspective of artificial neural networks. Some applications to education and to disordered development are also derived from this approach.

Why Use Models?

Because there can be substantial resistance to the very idea of modeling psychological development, it seems appropriate to justify a modeling perspective. The basic justification is that modeling has been repeatedly demonstrated to be extremely useful in a variety of other scientific disciplines that are considerably more advanced than psychology and cognitive science. It is worth documenting this claim in a bit of detail in order to clarify the relation between models and the phenomena being modeled.

Models have a long and productive history in various branches of science. A scientific model is basically a concrete representation of some hidden reality. Hidden realities are typically the target of models by virtue of being difficult to study by direct observation. Scientific models themselves take various forms. A model could be, for example, a drawing, a three-dimensional scale model, a set of symbols, a mathematical equation, or a computer program. Typically, a model is constructed after observing the behavior of a real system whose internal functioning is obscure. Among the qualities of a good scientific model are these:

- The model implements a theory in a precise, concrete, easy-to-manipulate way.
- The model covers (or generates) the phenomena of interest.
- The model helps to explain properties or behavior of the reality that it represents.
- The model links several different observations together, making them easier to understand.
- The model predicts new phenomena that can be tested.
- The model can be improved after observations or experiments reveal new facts.
- The model is expressed in a concrete form so that it can be easily manipulated and measured.

Some of the best examples of successful models come from the physical sciences. It is worth reminding readers about some of these models in physics and chemistry because they demonstrate principles that can illuminate the notion of modeling psychological processing.

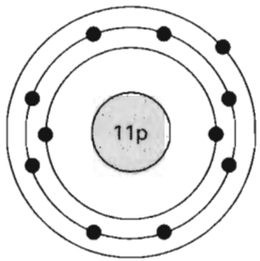

Figure 1.1
An atomic model of a sodium atom.

Atomic models

One of the most useful and influential of these physical-science models is the simplified atomic model of Bohr based on earlier work by Rutherford and others. This model views the atom as a structure consisting of a spatially small but massive nucleus containing protons and neutrons, surrounded by shells in which electrons orbit around the nucleus. A proton is a positively charged particle, and a neutron is a particle with no charge, just slightly more massive than a proton. An electron is a negatively charged particle with negligible mass. Atoms of the same element have the same number of protons, and this forms the unique atomic number of the element. The sum of protons and neutrons is the number of nucleons, the mass number of the atom. The number of protons equals the number of electrons.

An atomic model of a sodium atom is shown in figure 1.1. Sodium is a soft, light metal that reacts rather quickly with air and water. The nucleus has 11 protons and is surrounded by 11 electrons orbiting in three shells: 2 in the inner shell, 8 in the middle shell, and 1 in the outer shell. Such drawings are more realistically done in three dimensions because the electron orbits are not all in the same plane.

A similar model of a neon atom is shown in figure 1.2. Neon is a relatively inert gas, meaning that it is unlikely to be involved in chemical reactions. The neon nucleus has 10 protons and is surrounded by 10 electrons orbiting in two shells: 2 in the inner shell and 8 in the outer shell.

The number of electron shells possessed by an atom ranges from 1 to 7 and corresponds to the periods (rows) of the familiar periodic table. The

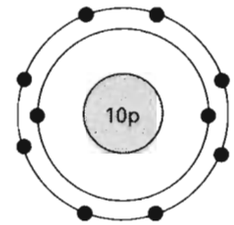

Figure 1.2
An atomic model of a neon atom.

periodic table summarizes the relations between the atomic structures and chemical characteristics of the elements. The columns of the periodic table correspond to the number of electrons in the outermost shell of an atom of an element.

This model of the atomic structure of elements integrates several centuries of physical data, from the finding that atoms combine to form molecules in definite proportions, to the discoveries of negatively charged, positively charged, and neutral particles. Chemists found that they could use this atomic model to systematically represent the chemical properties of elements.

It turns out that the key to understanding many chemical reactions is the number of electrons in the outermost shell. Because these outer-shell electrons are farthest from the nucleus, they are the most likely to be involved in chemical reactions. The maximum capacity of the outermost electron shell is eight electrons, and atoms with eight outer electrons are extremely stable and thus unlikely to enter into chemical reactions. This is why gasses such as neon are termed inert.

Lewis's (1966) *rule of eight* holds that an atom with less than eight outer electrons tends to combine with another atom to fill its outer shell to the capacity of eight electrons. Lewis devised a simple diagrammatic model of elements and molecules based on the atomic model and the rule of eight. In a Lewis diagram, outer-shell electrons are represented by a set of dots orbiting around the nucleus, which is represented by the symbol of the element. As noted in the top row of figure 1.3, a hydrogen atom (H) has a single outer electron, while an oxygen atom (O) has six outer electrons. Similarly, the top row of figure 1.4 indicates that a

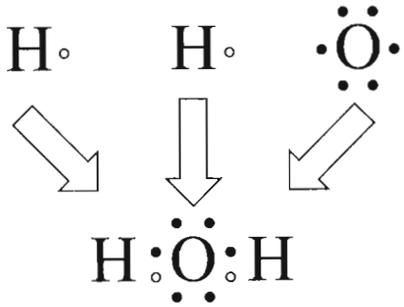

Figure 1.3
The Lewis atomic model of the formation of a water molecule (H_2O) from hydrogen (H) and oxygen (O) atoms. Outer electrons from the hydrogen atoms are shown as open circles, and outer electrons from the oxygen atom are shown as filled circles to keep them distinct in the representation of the water molecule.

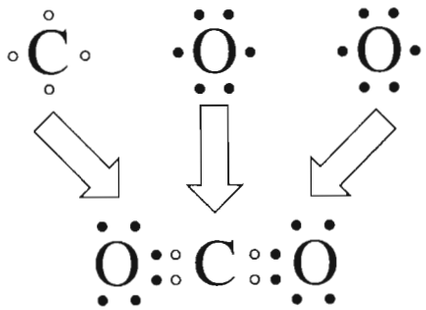

Figure 1.4
The Lewis atomic model of the formation of a molecule of carbon dioxide (CO_2) from carbon (C) and oxygen (O) atoms.

carbon atom (C) has four outer electrons. The number of chemical bonds that a nonmetal element can form with another element is 8 minus the number of its outer electrons. Thus, an oxygen atom can form 2 chemical bonds; a carbon atom can form 4 chemical bonds. Hydrogen is an exception to the rule of eight in that it can form a single bond, either by giving up its single electron or adding another single electron.

Figure 1.3 shows a Lewis-atomic-model diagram of the formation of a water molecule (H_2O) from hydrogen and oxygen atoms. The water molecule is formed when the oxygen atom shares an electron pair with

each of the two hydrogen atoms. At that point, the oxygen atom has 8 outer electrons, and each hydrogen atom has 2 outer electrons. This is a relatively stable chemical configuration.

Figure 1.4 shows a Lewis-atomic-model diagram of the formation of a carbon dioxide molecule from carbon and oxygen atoms. In this case, two electrons from each of two oxygen atoms bond with two of the carbon electrons. This creates a stable chemical structure in which each oxygen atom has a full complement of 8 outer electrons (6 of its own plus 2 from the carbon atom), as does the carbon atom (4 of its own plus 2 more from each oxygen atom).

The Lewis variant of the atomic model covers about 80 percent of all compounds in which atoms share some electrons between them. In its simplicity, Lewis's model emphasizes the importance of the outer electron shell in the chemical behavior of elements. Like all good models, it explains, links, and predicts the phenomena to which it applies. That is, it explains and predicts the nature of the molecules that are formed when elements combine in chemical reactions. It does this in a uniform fashion that links together vast numbers of chemical reactions, which would otherwise seem quite diverse.

If one traces back the historical roots of the Lewis model, it can be viewed as the result of incremental improvements in earlier models. In the fourth century B.C., the classical Greek philosopher Democritus described each kind of matter as being made up of identical, discrete, indivisible particles—atoms. In the eighteenth century, Dalton elaborated this idea in a model that described how atoms combine in particular proportions to create chemical reactions. Following the discovery of electrons in the nineteenth century, Thomson presented a model of divisible atoms containing negatively charged particles in a positively charged sphere, a kind of plum-pudding model. Then, to account for the behavior of protons emitted from radioactive materials, Rutherford created a model close to the contemporary atomic model by assuming divisible atoms composed of positively charged particles concentrated in a spatially small but massive nucleus, with negatively charged particles orbiting around it. After the discovery of the neutron, Bohr modified the Rutherford model to include neutral particles inside the nucleus and quantified electron orbits. As with most useful models, those of Bohr and

Lewis have undergone progressive improvements. For example, Schrödinger described electrons, not by their precise paths, but by regions in which they are most likely to travel. Further refinements in these models continue to account for additional data.

Models in other disciplines

Similar examples of useful models, many of them computational, can be found in other scientific fields, such as econometrics, microbiology, and meteorology. For example, numerical techniques for predicting weather use mathematical functions of current information on atmospheric temperature, pressure, and moisture to predict the state of the atmosphere a bit later. The predicted atmospheric state is used as input to the next predictive cycle, and this process is iterated for as long as it is useful. Such weather forecasts for up to the next 24 hours can be quite accurate. Even forecasts for the next two or three days can be somewhat useful. After that, forecasting the weather, even with sophisticated computer simulations, becomes hazardous. Still, the point is that accurate forecasting without a model would not be feasible. Current modelers of the weather are hoping to boost their computer capacity to enable about 2.5 trillion calculations per second, in an effort to increase the accuracy of their model's predictions.

Computational psychology

Computational psychology was born in the late 1950s as part of the multidisciplinary approach to cognition known as cognitive science. Newell, Shaw, and Simon (1958) built the first artificial-intelligence program, which executed proofs in formal logic. It was meant not just as an engineering feat but also as a cognitive model of how people perform logic. This effort was later generalized into a cognitive model for human problem solving in terms of rules (Newell & Simon, 1972). Contemporary versions of this idea of modeling human cognition in rules include the Soar (Newell, 1990) and ACT-R (J. R. Anderson, 1993) cognitive architectures. Klahr and Wallace (1976) pioneered the application of rule-based computation to developmental psychology.

Beginning another line of work in computational psychology, Minsky (1975) proposed that human knowledge can be represented in structures

called frames, composed of slots and fillers. This idea begat a body of research on what became known as case-based reasoning, as individual cases could be stored and generalized into frames (Kolodner, 1993; Riesbeck & Schank, 1989). So far, case-based reasoning has not engendered many applications to developmental psychology.

Computational psychology became more brainlike with the introduction of neural-network models, the basis for which can be traced to the McCulloch and Pitts (1943) proposal for a model of a neuron as a binary threshold machine. A variety of proposals for learning in networks of such devices eventually followed (Rosenblatt, 1962). Learning algorithms that overcame some of the limitations of these early models were introduced into psychology in the late 1980s (Rumelhart, Hinton & Williams, 1986). Application of neural computation to developmental psychology was firmly established by Elman, Bates, Johnson, Karmiloff-Smith, Parisi, and Plunkett (1996).

Models versus reality

It is interesting that physical-science models are rarely confused for reality—they are clearly just models, not the real phenomena that they model. But in the late twentieth century, when cognitive scientists began modeling psychological phenomena and researchers in artificial intelligence (AI) began to create devices to undertake cognitive tasks, there was a curious tendency for models to become confused with reality. Perhaps caught up in the hype for a new and energetic field or overly influenced by popular science fiction, enthusiastic practitioners and commentators alike were interpreted to claim that these computer programs were actually thinking. Critics attacked such claims by pointing out that because these early models of intelligence lacked semantics and intentionality, it was ridiculous to claim that they were actually thinking (e.g., Searle, 1980, 1984). The idea that such models actually think became known as the claim of *strong AI*. The ensuing theoretical disputes seemed to generate much more heat than light. All but lost in the commotion was the more sensible notion of *weak AI*, that these models of thinking were simply *models* of thinking. As models, they could be expected, at best, to provide the same sort of benefits as models in the physical sciences: explanation, linkage, prediction, and improvement, all in a concrete,

malleable format. If the model was a success, such benefits could be expected to be considerable, even if the models did not actually think.

Although empirical research, theory, and modeling are not nearly as advanced in cognitive science as in physical science, it is a goal of this book to show that model building is worth doing in the domain of psychological development. It is worth doing for exactly the same reasons as in other sciences—modeling is an enormous aid to the conduct of empirical research and to theorizing; namely, modeling makes theorizing more precise and easier to do, and it organizes, explains, and predicts empirical findings.

Why *Computational* Modeling?

Even people who accept the notion that modeling has been useful in the physical sciences may balk at the idea of using *computer* models of psychological processes, particularly as applied to developing children. The argument is sometimes made that computers cannot model development because children develop, but computers do not (Beilin, 1983; Neisser, 1976).

There is nothing magical about using computers to do psychological modeling. Any of the existing modeling algorithms can be implemented by human calculations aided perhaps by paper and pencil. Indeed, it has been argued that anything complex enough to have states could be a computer, even a roll of toilet paper and some small stones (Searle, 1980)! Such methods, however, would be so time consuming and difficult that no one would ever do simulations in that way. Modelers use computers for simulations mainly because of the convenience, accuracy, and power that computers supply.

Underlying this argument for computers is the notion that cognition, development, and other psychological processes are too complex for merely verbal theories. There are needs for precision, complexity, and theoretical unification that cannot easily be met without using computers. This quickly becomes apparent in using symbolic rules for modeling when a rule-interpreter program is required to keep track of large numbers of rules, active-memory elements, and variable bindings over the course of a problem-solving episode. Symbolic, rule-based systems

are described in chapter 3. Similarly, connectionist models may employ a large, complex network of many neuronal units, whose activations are a nonlinear function of their inputs. Connectionist models are more fully described in chapter 2. In both cases, it is much too tedious and difficult to run simulations by hand, particularly when there are multiple experiments to simulate, each with several different conditions, and variations in parameter settings to explore. High-speed computers and powerful programming languages are extremely helpful in such cases.

Indeed, there is a substantial empirical component to computer simulations, even to the extent that simulation results may need to be subjected to statistical procedures such as analysis of variance to determine the nature and significance of condition effects. Multiple networks, each starting with unique connection weights and learning in a unique environment, might correspond to multiple human participants, and simulations might mimic the complexity of psychological experiments or longitudinal studies.[1] If this is true, does computational modeling fall under the heading of theoretical work or empirical work? In a way, modeling is both; modeling is theoretical work with an empirical component. A model may implement a theory by specifying various environmental and innate constraints, but its output may need to be replicated with multiple runs and statistically analyzed to determine exactly what happened and whether the results are statistically reliable.

Before leaving this section on benefits of computational modeling, it is appropriate to take note of Newell's (1990) *challenge of computational sufficiency*. His challenge goes like this: if you believe that you have a coherent and correct psychological theory of some set of phenomena, then you should be able to implement that theory on a modern digital computer to effectively simulate those phenomena. Such an exercise would amount to considerably more than mere flamboyance. For in implementing a theory computationally, a researcher typically discovers all sorts of contradictions, poor specifications, and other formidable challenges. Solving these various problems to complete the simulation invariably improves the original theory in many ways, principally by identifying weaknesses, suggesting computationally feasible fixes, and being more specific about almost everything in the theory—how knowledge is represented, how it is processed, etc. Newell's challenge is a significant

one that every serious psychological researcher should consider. If the challenge is successfully met, then one has produced a model that is computationally sufficient to actually produce a simulation of the phenomenon—not merely a somewhat convincing verbal story about what might be going on, but a clear proposal that is good enough to actually work.

It should not be forgotten that psychologists are very skilled at verbal explanation, particularly after the results are in. I have never seen a psychological result that went unexplained for any significant period of time. Witnessing enough of these "theoretical" explanations reminds one of the quotation attributed to the German poet Goethe: "When ideas fail, words come in very handy." Newell's computational challenge is about firming up theoretical ideas so we can more easily determine whether they fail or not.

Notice that Newell's challenge is one of *sufficiency*, rather than *necessity*. He is claiming not that a successful simulation makes a convincing case for a particular model's being necessary to capture a phenomenon—only that a successful model is sufficient. The reason for demurring from computational necessity is simple. Tomorrow or the next day, someone may come up with a better model. This not only happens frequently in science; experienced modelers fully expect it to happen. Recall that one of the benefits of modeling specified earlier was that a good model can be improved after observations or experiments reveal new facts. The very specificity and sufficiency of a good model facilitates such improvements. It is relatively easy to see how and where a well-specified model is failing, which in turn helps one to fix it. So it is unreasonable to expect that a given model will ever be the final word, that it is in fact *necessary* as an explanation of some phenomenon. Sufficiency is all that one may hope for in a model, but sufficiency is still often a significant challenge, particularly in models of psychological processes.

Notice finally that this argument about sufficiency and necessity applies equally well to traditional verbal theories. The best any theory can do is to provide a *sufficient* explanation of something. A theory cannot be expected to provide a *necessary* explanation, because a better theory may come along at any future time. As with most issues, the difference is that the argument can be a bit clearer with computational models than without because sufficiency can be easier to assess with a running computational model. To see if a model is computationally sufficient, just run the model with the same inputs as given to human participants and see if it produces the same responses that they do.

What Is Neural Computation?[2]

As will be seen, there is considerable choice in techniques for computational modeling of psychological development. Consequently, it is important to justify the use of any particular technique, such as neural networks. Even before providing that justification, it is important to know what neural networks are and how they function.

This area of study is known by various names, including *neural networks*, *artificial neural networks*, *neural computation*, *neural modeling*, *connectionism*, and *parallel distributed processing*. In this book, these names are used interchangeably without any distinction in meaning.

Neural networks were inspired by two different sources: neuroscience and the mathematics of statistical mechanics. The neuroscience inspiration is based on highly simplified abstractions of how information is processed in biological neurons (brain cells). Simplified abstraction is, of course, an essential feature of scientific modeling.

Biological neurons fire by sending electrical impulses from their cell body down the axonal branches. At the ends of axonal branches are tiny gaps (synapses), whose chemical activity allows the transmission of these impulses to the nearby dendrites of other neurons. A neuron's average rate of firing can be taken as an index of its general activity level. Rates of neural firing range from 0 up to about 300 Hz. Brains have large numbers of neurons with extensive synaptic connections to neighboring neurons. For example, the human brain is estimated to contain 10^{11} neurons, each neuron having several thousand synaptic input connections. Brain neurons are organized in up to six layers of connectivity, with some connections bypassing intermediate layers. The signals coming across synaptic gaps either tend to raise or lower the electrical potential of the receiving neuron, but in any case are integrated in some way by the cell body of the receiving neuron. When the integrated potential exceeds a threshold, the receiving neuron itself fires. Although

this account ignores many relevant details in the active area of neuroscience, it is imagined that much of this detail is unimportant for guiding a general study of the properties of neural networks.

The amount of neurological detail that is used to constrain neural modeling varies considerably. There are numerous attempts to model actual neurological circuits for low-level processes and many more attempts to use general neuroscience principles in the modeling of both low- and high-level processes. Most of the material in this book falls into the latter category. Too little is known about the actual underlying neural circuits involved in psychological development for very detailed neural modeling.

Inspiration for neural modeling can also be found in the mathematics of a branch of physics called statistical mechanics. Like some physical systems, neural networks are composed of potentially large numbers of elements whose interactions can have emergent properties. Thus, the mathematics developed for such complex systems in physics can be usefully applied to neural networks. Indeed, many neural-network researchers are physicists by training, e.g., Hertz, Krogh, and Palmer (1991). Some of the relevant mathematics essential to understanding neural networks as applied to psychological development are presented in chapter 2.

Activity and connectivity

For the most part, artificial neural networks can be understood in terms of the twin notions of *activity* and *connectivity* (Mareschal & Thomas, 2001). As noted in the first row of table 1.1, neural networks are composed of two types of elements: units and connection weights. Units in these artificial neural networks correspond roughly to neurons in biological networks (brains); connection weights correspond roughly to biological synapses. Activity in units corresponds roughly to firing rate in neurons, whereas connection weights correspond roughly to the ability of neurons to excite or inhibit the activity of other neurons. Both units and connection weights are implemented in neural networks as real numbers. At the psychological level of memory, unit activity corresponds to active memory, known in the past as working memory or short-term

Table 1.1
Activity and connectivity in artificial neural networks

	Activity	Connectivity
Elements	Units	Connection weights
Brain analogy	Neurons	Synapses
Behavior	Firing rate	Excitation, inhibition
Implementation	Real numbers	Real numbers
Memory type	Active	Long-term
Time scale of change	Seconds	Seconds to years
Diagram representation	Circles	Lines

memory, while connection weights encode the system's long-term memory. Whereas unit activations change over seconds, changes in connection weights can occur rather quickly or over a period of years when representing very long-term learning over the course of development. Finally, in conventional diagrams of neural networks, units are represented as circles and connection weights are represented as lines.

Network topology

An artificial neural network is a set of units and connection weights organized in a particular topology. A wide variety of network topologies are possible. The most general network topology is one in which each unit is connected to each other unit, implementing a so-called auto-associator network (McClelland & Rumelhart, 1985). Each unit in an auto-associator network gets input from the environment and sends output to the environment as well.

All other network topologies can be understood in terms of restricting connections within an auto-associator network. Some units might receive no input from the environment, or some units might not send any output to the environment. Some connections in the auto-associator topology could be deleted, or constraints might be placed on the values of some connection weights. For example, certain groups of units might have mutually inhibitory connections, implementing a winner-take-all network level. In such a winner-take-all level, only one unit tends to be active. This could implement so-called *grandmother* cells, in which, for

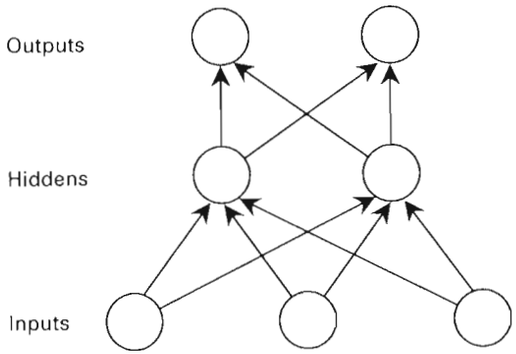

Figure 1.5
A sample multilayer, feed-forward, back-propagation network.

instance, there is one neuron for recognizing a person's grandmother (Page, 2000).

Easily the most popular network topology in connectionist modeling is the so-called multilayer, feed-forward, back-propagation network, shown in figure 1.5. This network topology has a layer of input units, a layer of output units, and one or more layers of hidden units. Hidden units are so-called because they are hidden from the environment—they neither receive input from the environment nor send output to the environment. If input units can be seen as analogous to sensory neurons and output units as analogous to motor neurons, then hidden units can be viewed as analogous to interneurons.[3] As such, hidden units are involved in important computations performed in between the input and output layers. As explained in chapter 2, the nature of the activation functions for hidden units is important in enabling these critical, intermediate computations.

Further deletion of connection weights from the fully connected auto-associator connection scheme ensures that typical feed-forward networks have no backward connections, lateral connections within layers, or cross-layer forward connections.

Most neural networks are hand-designed and static, in the sense that their topology remains unchanged over the course of learning. Later I discuss neural learning algorithms in which network topology is constructed automatically during the course of learning. This innovation of

allowing network growth proves to be particularly important in simulating many aspects of psychological development.

Integration of inputs

Each unit in a neural network is running a simple program in which it computes a weighted sum of inputs coming from other units and outputs a number, which is a nonlinear function of the weighted sum of inputs. This output is then sent on to other units running this same simple program.

In a feed-forward network, of the sort commonly used in simulations of psychological development, activations are passed forward from the input units to the hidden units, and then on to the output units.

Representation

Representation of knowledge in neural networks is often described as being either local or distributed. The aforementioned grandmother (or grandfather) cells constitute a so-called *local* representation, in which instantiation of a concept is represented by activation of a single unit. In contrast, so-called *distributed* representations use a number of units to represent a given concept or idea. Each such distributed unit is typically involved in representing many different concepts. There is some controversy about whether it is preferable to use local or distributed knowledge representations in neural-network simulations. Whereas distributed representations may offer some advantages in terms of robustness and generalization, local representations may be easier to port to other cognitive functions because a local representation is compactly represented on a single unit.

Learning

In what is called supervised learning, the discrepancy between actual network outputs and target outputs that the network is supposed to produce is computed as network error. Networks are often trained by presenting many examples of input-output pairs. Each such pair specifies a particular pattern of input activations that should produce a particular pattern of output activations. Network weights are adjusted to reduce the discrepancy between actual and target output activations, using

techniques presented in more detail in chapter 2. In the case of many multilayer networks, error is propagated backward to earlier layers of weights; hence the term *back-propagation* learning. Unsupervised learning can also occur, without output targets, as networks learn to group together similar input patterns. Output unit activations in these cases can implement a topographic feature map of the inputs.

Summary

Many of these features of neural networks are elaborated considerably in chapter 2. For now, it is only important to convey these characteristics of neural networks:

- They embody many of the basic features of brains.
- They utilize some of the mathematics from statistical mechanics for understanding complex system dynamics and emergent properties.
- They simulate both active and long-term memory processes.
- They can learn by modifying connection weights, in either supervised or unsupervised paradigms.

Why Use Neural Networks for Modeling?

Even if one agrees that modeling can be a useful tool in studying psychological development, it is not a forgone conclusion that neural networks ought to be employed for the modeling. In fact, there are a number of promising modeling methods that could be selected for developmental research. In rather sharp contrast to neural-network techniques, many of them involve symbolic computation and represent knowledge in terms of rules or frames. Rules are if-then statements that specify what actions to take when certain conditions are satisfied. Frames are slot and filler structures that organize knowledge about objects or events. Later in the book, symbolic techniques, particularly rules, are examined and compared to neural-network methods.

Justifications for the use of neural modeling in the developmental arena range from demonstrated success of these models to their having such features as neural plausibility, graded representations, self-organization, and principled naturalness (Elman et al., 1996; Shultz, 2001).

Demonstrated success

One of the most compelling reasons for using neural networks to study psychological development is their demonstrated success. But how can relative modeling success be measured in any objective fashion? Surely any such measure would result in endless controversy.

A crude indication of modeling success would be a simple count of published papers in each relevant category of model, the assumption being that researchers prefer models that actually work. An extremely cheap way to do such a count is to consult some recent, balanced survey of modeling in psychological development. Fortunately for this argument, there is such a survey in our field, a chapter in the latest edition of the venerable *Handbook of Child Psychology*, coauthored by a leading practitioner from each of the two main categories of computational models—rule-based and connectionist (Klahr & MacWhinney, 1998).

My classification of the 37 published computational models of psychological development in that survey yields 5 rule-based models, 28 connectionist models, and 4 ad hoc models that were neither rule-based nor connectionist. This outcome is especially interesting in view of the fact that the first connectionist model was predated by several models in the other two categories. Despite the relatively short time that connectionist models of development have been around, by 1998 they already accounted for the vast majority of publications on computational modeling of psychological development. Admittedly, not all computational developmental models are reviewed in that chapter, but it is likely that a more inclusive review would find a preponderance of connectionist simulations at least as great. A more inclusive count of computational models can be undertaken by using material throughout this book.

This preliminary count is also interesting because it *may* not generalize to computational modeling of psychology in general. For example, a recent textbook on cognitive science claimed, without presenting the results of a formal count, that rule-based models were numerically predominant. "Of all the computational-representational approaches described in this book, which has had the most psychological applications? The answer is clear: rule-based systems" (Thagard, 1996, p. 51). Perhaps connectionism is particularly well suited to developmental problems, as some have argued (Plunkett & Sinha, 1992). It is doubtful that the

relative numbers of publications of different categories of models would have changed drastically in just two years, from 1996 to 1998.

In later chapters, the degree to which connectionist and other models really do succeed is given a closer examination. For now, it is perhaps sufficient to make the point that connectionist models have already demonstrated considerable comparative success in a wide variety of developmental domains, from visual perception to language acquisition, concept formation, and problem solving and reasoning.

Neural plausibility

Another reason to favor neural networks over the more traditional symbolic modeling methods is that neural networks are compatible with what is known about the brain. The principles by which artificial neural networks function were, after all, patterned on knowledge of brains and neurons. And even though many neural-network models are not accurate at the level of simulating particular brain circuits, at least they conform to brain-style computation, as it is currently known. This provides a kind of neurological plausibility that is simply absent from many other modeling methods. Indeed, the classical symbol-system view of cognition prided itself on its independence of neural phenomena. It was often called *functionalism* because it was concerned strictly with how cognition functioned rather than how it was implemented in the brain. The danger of this kind of strong functionalism is, of course, that it ignores a host of possible constraints on theories that could be supplied by knowledge of the nervous system. One of the great strengths of cognitive science has been to freely borrow constraints from neighboring disciplines. Because cognition is so difficult to study, the more constraints that can be placed on studies, the more accurate will be the models. The hope is that converging constraints from various cognitive disciplines will help to unlock the fundamental mysteries of cognition. Why arbitrarily close off access to the probably relevant constraints of neuroscience, particularly in our present era of rapid progress in neuroscientific research? For the study of psychological development in particular, there is the burgeoning neighboring discipline of *developmental cognitive neuroscience* (Johnson, 1997). Modelers who choose to ignore this work do so at their own peril.

Graded representations

Human knowledge is very often approximate and graded, rather than categorically precise. For example, we may be unsure whether something is true and thus only partly believe it, hold ambivalent attitudes about something, or only roughly estimate a fact. Such graded knowledge can be naturally implemented in neural networks whether the representation technique is local or distributed. In the case of local representation schemes, a unit's graded response is guaranteed by the fact that the unit is not simply off or on, but can assume any of a range of continuous values. In distributed representation schemes, this graded response is further enhanced by the fact that differing numbers of participating units may be relatively active. Graded responsiveness has been well exploited in the connectionist simulation of a variety of developmental phenomena. For example, precursors of a mature object concept may reflect partial knowledge representations (Munakata, McClelland, Johnson & Siegler, 1997), and small-number conservation problems may be solved correctly before large-number conservation problems (Shultz, 1998).[4]

Self-organization

In biological systems, pattern and order can emerge without the need for explicit internal or external instructions (Oyama, 1985). Many aspects of psychological development are also suspected to be self-organized rather than determined by either genetics or environment (Karmiloff-Smith, 1992; Mareschal & Thomas, 2001). This means that significant development can occur without the system being poked or prodded to develop. Instead, development arises from processes that are endogenous to the developing system. To get an explanatory handle on such processes would presumably require some kind of modeling, using techniques that themselves are capable of self-organization.

Some artificial neural networks are able to discover for themselves the important features, representations, and correlations that may be present in the environment. Such networks, in other words, display a certain amount of self-organization that could perhaps be exploited in the service of modeling self-organizing psychological development. This quality of self-organization in formal modeling is apparently somewhat rare and thus precious.

Principled naturalness

In successful modeling, it is essential, of course, to cover the data and phenomena being modeled. To cover a phenomenon means that the model reproduces the behavior being modeled. However, mere coverage is often not sufficient for a model to be considered a leading contender for scientific study. It is also essential that a model cover phenomena for the right reasons. The *right* reasons are parameters that operate according to established scientific principles that are consistent from one domain of application to the next. In contrast, even an inappropriate model could be made to seem to cover some phenomenon by clever manipulation of key model parameters that have no scientific basis.

One of the main attractions of neural-network techniques is that they work according to well-established principles of neuroscience and mathematics. They often don't need a lot of parameter tweaking or hand designing to achieve adequate data coverage.

If I may be permitted a bit of autobiographical leeway, I can describe the origin of my own fascination with neural-network techniques. I had been searching for appropriate modeling techniques for a few years because I felt that I needed to gain a deeper theoretical understanding of the phenomena that I was studying with the conventional psychological techniques of verbal theorizing and human experimentation. In such an exploratory mode, I built a number of computational models with symbolic techniques such as rules. Although I invariably felt that I learned something and gained theoretical insight by building such models, I also became uneasy because almost nothing that I learned seemed to generalize from one model to the next. This was because each domain seemed to require a distinct scheme for knowledge representation and a unique rule base. There was data coverage, for sure, but the coverage had more to do with how I designed the models rather than with more abstract principles.

In contrast, when I started modeling with neural networks, I was impressed with the fact that almost everything that I learned from one model would generalize effectively to the next domain. The apparent reason for this was that the neural-network models were following basic principles that applied without substantial change to several specific domains. This was particularly true of generative techniques such as

cascade-correlation, which not only learned the connection weights but also designed the topology of the network automatically.

Why Use *Generative* Neural Networks?

Most researchers who apply neural networks to psychological development use some variant of static, multilayered, feed-forward networks. In this technique, the network topology is designed by hand, as are the training patterns; the network weights are learned automatically. As noted in some detail in the next chapter, there are a number of problems with this kind of scheme. For example, the programmer needs to design the network topology, in most cases without knowledge of the underlying circuitry of real neurons. How many hidden units should be employed? In how many layers should they be arranged? Should some network sections be segregated so that not every unit in one layer connects to every unit in the next layer? At present, there is more art than science to settling such issues. There are several *rules of thumb*, but no comprehensive scientific principles (Reed & Marks, 1995). If there were comprehensive scientific principles governing network design, then such networks could be designed automatically.

In contrast, generative networks incorporate topology design into the learning process. Generative networks search not only in weight space for the right combinations of weight values but also in topology space to find the right arrangement of network units for the particular problem being learned. In the next chapter, we examine in detail how these two simultaneous searches are possible. The principles of network design in generative networks may turn out to be psychologically and neurologically incorrect, but at least they are principled and well specified in a mathematical and computational sense.

Other advantages of generative networks will be identified as we progress. And there will be a few cases of head-to-head competition of static and generative network models. For now, it is only important to note that coherent cases can be made for modeling, for computational modeling, for neural-network modeling, and for generative-neural-network modeling of psychological development. Objections to all of these ideas are addressed in chapter 6.

Conclusions

This chapter began by reviewing some of the important, enduring issues in developmental psychology. Among the issues identified as important are structure and transition, representation and processing, innate and experiential determinants of development, stages of development, the purpose and end of development, and the relation between knowledge and learning. Because this book tries to gain some leverage on these issues through computational modeling, using generative connectionist models, it seemed important to justify a number of strategic research decisions. Arguments were made for the importance of modeling in scientific work, reasons for computer modeling, the advantages of a connectionist approach to modeling, and a generative approach in particular. Along the way, the basics of neural networks were introduced. In chapter 2, the neural-network machinery most commonly applied to psychological development is presented in a more detailed and substantial way. Readers who already know this material or want to continue to avoid it may decide to move directly on to chapter 3. Chapters 3 to 7 can be read and appreciated without the rather technical material in chapter 2, but understanding those chapters would be enhanced by this material.

2

A Neural-Network Primer

In chapter 1, I argued that it can be fruitful to apply neural-network models to the study of basic issues in psychological development. I discussed some of the basic features of such networks and promised to supply important details in this chapter. Because the field of neural networks is vast and not all of its techniques have been used in developmental research, my plan is to cover only the most frequently used techniques. Although all of these methods are presented in detail somewhere else in the literature, I present all the essentials here (one-stop shopping) with sufficient background to enable understanding of key ideas by basically all readers, not just those with an extensive background in neural networks and mathematics. If you know algebra and some calculus, you should be able to follow the discussion quite easily. Four different appendixes present additional background on the key notion of slopes and their computation in neural learning.

Because the notion of linear separability is of recurring importance in neural-network research, I start with a brief discussion of the distinction between linearly separable and linearly nonseparable tasks.

Because of its prominence in developmental simulations, I give an extensive presentation of back-propagation, a technique for supervised learning from examples in multilayered feed-forward networks. This is preceded by a discussion of how network units integrate their inputs and determine how active they should be as a result of that integration (activation functions), as well as the basics of weight adjustment. I then present three important variations on the back-propagation algorithm, again because of their prominence in simulations of development. One of

these variations is cascade-correlation, a generative algorithm for building network topology during learning. Such increases in the computational and representational power of networks allow for simulation of underlying qualitative changes in development, long a basic assumption of Piaget and other developmental theorists. Another significant variation on back-propagation is that of simple recurrent networks, a technique that allows processing sequential inputs, such as sentences, by implementing a kind of working memory for what has just been processed in the previous step. The third variation on back-propagation is that of encoder networks. These are networks that learn to recognize stimulus patterns by encoding them onto hidden units and then decoding them onto output units. Application of encoder networks has enabled the simulation of various recognition-memory phenomena such as seen in the literature on habituation in infants.

Then I present auto-associator networks. As noted in chapter 1, these are fully connected networks in which each unit plays the roles of both input and output units. Although there are no hidden units, which would be required for learning nonlinear functions,[1] there is recurrence, with cycling of activation updates. Such networks have the potential to learn linear relations among stimulus patterns and to engage in pattern completion when presented with partial or degraded stimulus inputs. Auto-associators have been used to implement recognition memories in habituation studies and learning of category names.

An unsupervised-learning technique called feature mapping has been used to identify the main features of stimuli in concept-acquisition and object-permanence simulations. This allows a network to learn to group together stimuli that are similar in their descriptions, through a process of self-organization, without any correction from environmental feedback.

With so many types of neural networks in current use for developmental studies, it is not always apparent which technique should be selected for any particular application. To provide some advice on this, a rule-learning program processes examples of current neural models to determine a small and coherent set of rules for algorithm selection. This exercise has the added advantage of presenting the leading symbolic rule-learning program for developmental research.

Finally, interested readers are pointed towards neural software that facilitates the use of these various types of networks. Such software makes it much easier to begin neural-network simulations, often without a background in programming languages.

Linear Separability

Many artificial neural networks can learn functions that map inputs onto outputs. Of considerable theoretical and historical importance is whether the functions being learned, and the tasks that they represent, are linearly separable or not. A linearly separable function is one whose outputs are some linear combination of its inputs. This notion of linearity is usually clearer when illustrated graphically.

If the training patterns representing a linearly separable function are plotted in a multidimensional input space, where each dimension refers to variation in one of the inputs, then it is possible to separate the values on one output unit from each other by a plane. In contrast, no such planes can be found for linearly nonseparable tasks.

Examples of two linearly separable and two linearly nonseparable datasets are shown in figure 2.1. For visual simplicity, each of these pattern sets has just two continuous input units and a single binary output unit. In such two-dimensional problems, the "plane" separating input patterns with different output values is just a straight line. The planes used to separate patterns are of higher dimensionality for problems with more than two inputs. Planes of varying dimensionality are more generally referred to as *hyperplanes*. In each plot in figure 2.1, there are 60 randomly selected x, y pairs of input values. Some of them have one output value, represented in the figure by an open square and others have a different output value, represented by a filled diamond shape.

The datasets depicted in figures 2.1a and 2.1b are linearly separable, as revealed by an overlaid straight line that separates patterns of one output value from those with the other output value. The dataset plotted in figure 2.1a can be viewed as distinguishing large sums of the x and y inputs (indicated by open squares) from smaller sums (indicated by filled diamonds). Hence, the boundary line has a negative slope. The dataset plotted in figure 2.1b has a boundary line with a positive slope. This

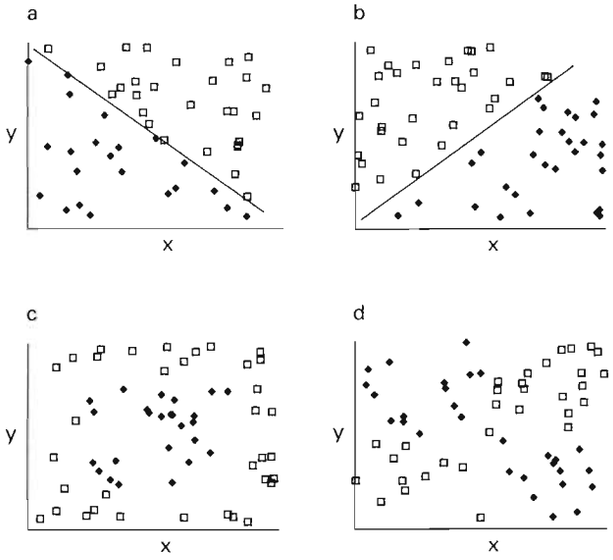

Figure 2.1
The two-dimensional input spaces for four functions, two of which are linearly separable (a and b) and two of which are not (c and d).

function can be viewed as distinguishing cases where $x > y$ (diamonds) from cases in which $y > x$ (squares).

In the case of the two datasets that are not linearly separable (figures 2.1c and 2.1d), there is no way to draw a straight line to separate patterns with the two different output values. Figure 2.1c shows a center-surround function, with one class occupying the inside of the input space (diamonds), and the other occupying the periphery (squares). The dataset in figure 2.1d is a continuous version of the exclusive-or function in which the quadrants of the input space with low values on both x and y or high values on both x and y (squares) differ from the other two quadrants (diamonds).

Artificial neural networks typically find linearly separable problems easier to learn than those that are not linearly separable. Ordinarily, the more nonlinearity in the problem, the harder it is to learn. In particular, as we will soon see, successful learning of a nonlinear dataset requires the use of so-called hidden units with nonlinear activation functions.

Because all neural-network learning algorithms need to specify how network units integrate their inputs, how this input affects their activity, and how to adjust weights, I turn next to these topics.

Integration of Inputs

In chapter 1, I noted that network units integrate their inputs by summing the weighted activations of sending units. Somewhat more formally, the weighted input x_j to unit j is computed as follows:

$$x_j = \sum_i w_{ij} y_i \tag{2.1}$$

Here w_{ij} is the connection weight between sending unit i and receiving unit j, and y_i is the activation of sending unit i. Equation 2.1 says to multiply the activation of each sending unit i by the connection weight to receiving unit j and to sum these weighted activations over all of the units indexed by i that send inputs to receiving unit j. Negatively weighted activations (which occur when either the sending-unit activation or the connection weight is negative) tend to inhibit the activation of the receiving unit, while positively weighted activations (which occur when both the sending-unit activation and the connection weight are positive or both are negative) tend to excite the activation of the receiving unit. Of course, summing the weighted activations determines the overall or net effect on the receiving unit. For this reason, the sum of weighted inputs defined in equation 2.1 is often referred to as the net input to a receiving unit.

As an example of these computations, consider the simple four-unit network shown in figure 2.2. The three sending units at the bottom of figure 2.2 have activations of 0, 1.0, and −1.0. The corresponding connection weights to the receiving unit are 0.5, 1.0, and −0.8, respectively. In this case, net input to the receiving unit is computed as $(0 \times 0.5) + (1.0 \times 1.0) + (-1.0 \times -0.8) = 0 + 1.0 + 0.8 = 1.8$.

Activation Functions

How net input to a unit is translated into activity for the unit depends on the activation function of the unit. With a linear activation function,

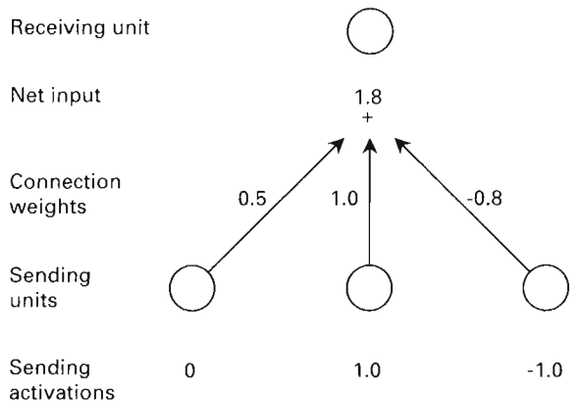

Figure 2.2
An example of computing net input in a part of a simple network.

the activity of the receiving unit could equal its net input. However, the nature of network computations that can be accomplished with linear activation functions is extremely limited. Also, linear activation functions do not conform to the characteristics of biological neurons, which are known to have a floor and a ceiling of activity. That is, real neurons have a minimum level of activity, such as none, and a maximal level of activity, typically about 300 Hz (cycles per second).

To implement these more realistic characteristics, one typically uses a nonlinear activation function, such as a sigmoid function or a hyperbolic-tangent function. A sigmoid activation function is shown in figure 2.3 as an example. This is the function:

$$y_j = \frac{1}{1 + e^{-x_j}} - 0.5 \tag{2.2}$$

Here x is the net input to unit j, and e is the exponential function. This activation function specifies that the negative exponential of the net input is added to 1 and divided into 1 before subtracting 0.5 to yield the activation of the receiving unit. As figure 2.3 reveals, for net inputs ranging from -10 to 10, this function has a floor at -0.5 and a ceiling at 0.5, with a threshold at 0.[2]

To fully understand this function, we can deconstruct it and then rebuild it step by step. We can start with the simple exponential function

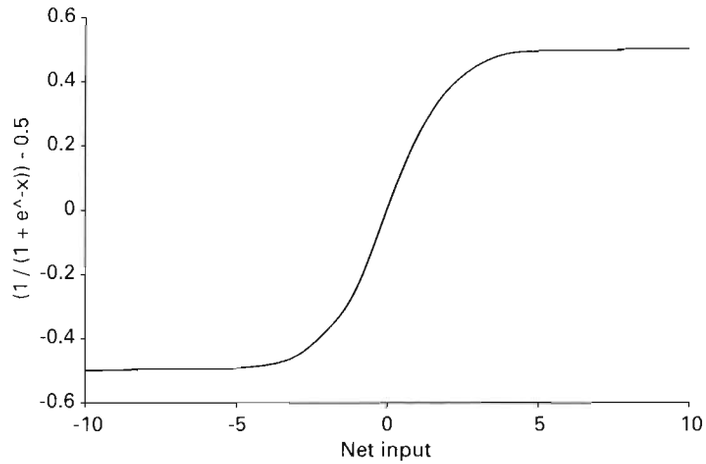

Figure 2.3
Sigmoid activation function.

$y = e^x$, again with net inputs ranging from -10 to 10, as shown in figure 2.4. With negative x, this function starts very close to 0, but it increases more and more rapidly as x increases. At $x = 0$, the exponential function yields a value of 1. As x increases above 0, the exponential function grows very fast indeed.

The next step in rebuilding the sigmoid function is to understand the negative exponential function $y = e^{-x}$, shown in figure 2.5. The negative exponential does just the reverse of the exponential: it starts very high with a negative x and then decreases less and less rapidly as x increases. Like the exponential function, at $x = 0$ the negative exponential function yields a value of 1. The main point for understanding the sigmoid function is that as x increases, the negative exponential function approaches a value of 0.

This fact is helpful in understanding how dividing 1 by the sum of 1 and the negative exponential produces a smooth function with a floor and ceiling and a steep threshold between them. This is illustrated by the asigmoid function plotted in figure 2.6[3]

$$y_j = \frac{1}{1 + e^{-x_j}} \tag{2.3}$$

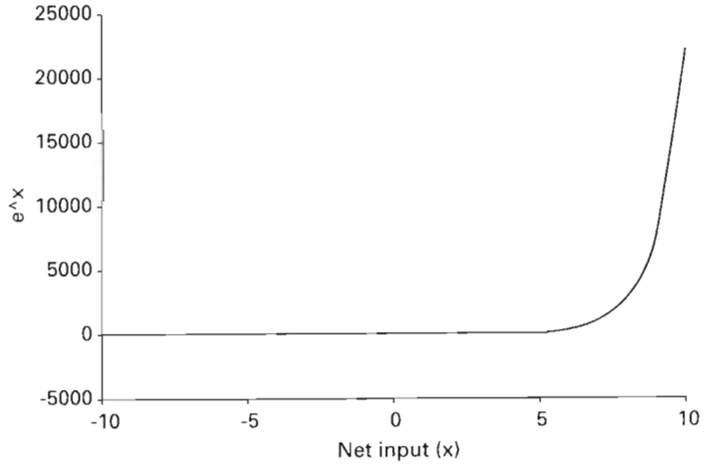

Figure 2.4
Exponential function.

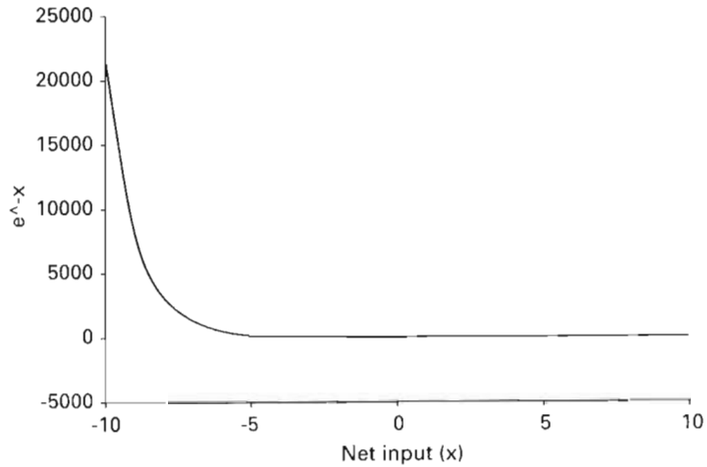

Figure 2.5
Negative exponential function.

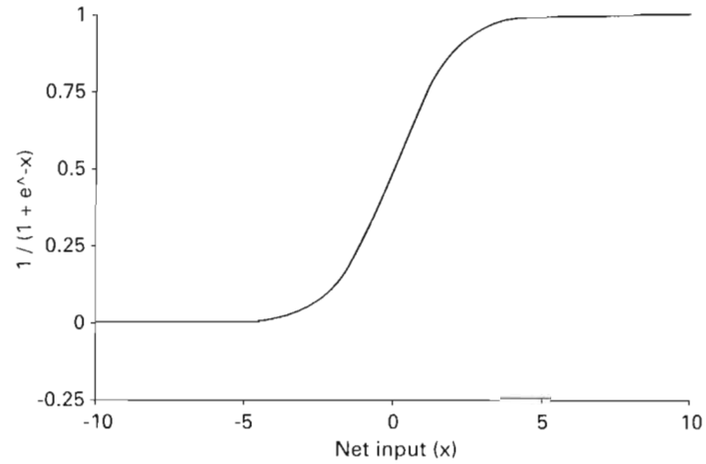

Figure 2.6
Asigmoid function.

In the asigmoid function, when x is highly negative, e^{-x} is extremely large, creating a large denominator in equation 2.3 and a value of y very close to 0, the floor of the asigmoid function. As x increases, e^{-x} approaches 0, and the denominator in equation 2.3 approaches 1. This yields a ceiling for the asigmoid function of 1. The threshold of the asigmoid function is at 0.5. This activation function, which is quite commonly used in neural network research, is called *asigmoid* because it is asymmetric around 0, always yielding positive values.

Subtracting 0.5 from the asigmoid function, as in equation 2.2, is the final step in producing the sigmoid function (figure 2.3), which is symmetrical around 0, with a floor at -0.5, a ceiling at 0.5, and a threshold at 0. The sigmoid function is the default activation function in the cascade-correlation algorithm, which is featured in this book and in many developmental simulations.

The importance of having a nonlinear activation function, such as the sigmoid function, in neural networks is to enable the learning of nonlinear target functions. Simpler, linear functions, like those in figures 2.1a and 2.1b, have outputs that are some linear combination of the inputs. However, because many aspects of the world exhibit nonlinear

functions, such as those in figures 2.1c and 2.1d, it is important for cognitive models to be able to capture this kind of more complex learning.

There are now proofs that a network with a single layer of hidden units can learn any *continuous* function to any degree of accuracy if this layer contains a sufficient number of hidden units (Hertz et al., 1991). And there are proofs that *any* function can be learned by a network with two hidden layers, again provided that there are enough hidden units in each of the two layers.

Weight Adjustment

Neural networks learn chiefly by modifying their connection weights to reduce error. Connection weights are typically trained by presenting example pairs of input and output values. Because there are often multiple inputs and multiple outputs, these example values are presented in the form of vector pairs—in each pair, one vector holds input values and the other holds output values. In general, each vector contains real numbers. Gradually, by processing such examples, a network learns to produce the correct output vector in response to each input vector. Vector pairs used in such training are typically known as the training set.

During learning, error at the output units, for a single pattern, is computed as the sum of squared discrepancies between outputs and targets:

$$E = \frac{1}{2} \sum_j (y_j - t_j)^2 \qquad (2.4)$$

Here y_j is the network's actual output at unit j, and t_j is the target output for unit j specified in the training set. The goal of learning is to minimize error as measured by equation 2.4. Because error is some function of the network's connection weights, the goal of minimizing error is accomplished by adjusting the network's connection weights. It is perhaps somewhat surprising that such connection-weight adjustments can be done with only local computations, i.e., without worrying what other weights are doing.

To understand how this kind of learning works, suppose that the error contributed by a single connection weight is some unknown parabolic function of the value of that connection weight. As shown in figure 2.7,

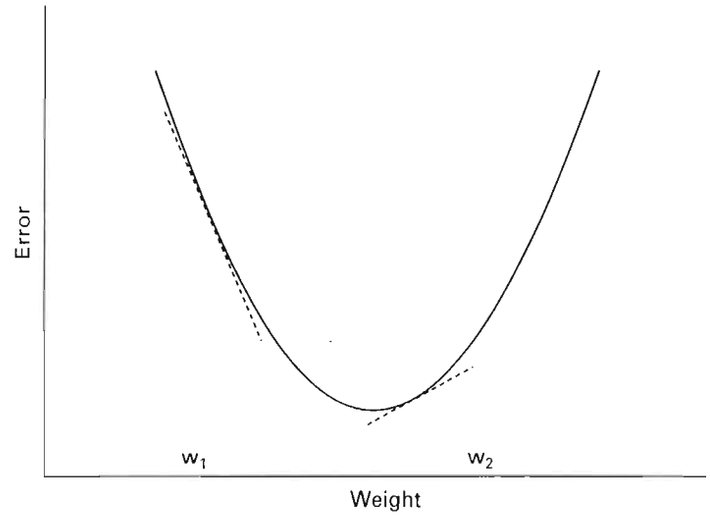

Figure 2.7
A hypothetical function relating the value of a single network weight to error. Slopes of the error function at two weight values (w_1 and w_2) are shown with dashed lines.

the assumption is that the error contributed by variation in a single weight takes the form of a parabola with the arms opening upwards. This assumption makes sense on the view that there is some optimal value for each connection weight where either increasing or decreasing the value of the weight from this optimum serves to increase error. If we knew the exact shape of this error function, learning would be easy—just make each connection equal to the value that minimizes error. The problem is, of course, that life is not so simple. These error functions are unknown to the learner, and are unlikely to be a simple, smooth parabola, as pictured in figure 2.7. The learner, as modeled by a neural network, knows about the size of the discrepancy between actual responses and target responses in the training set, but not about the shape of the error function.

Even if the exact shape of the error function is unknown, the scant information available can be used to compute the slope of this function at each connection-weight value that has been experienced. The slope (or gradient) is the first derivative of a function, evaluated at a particular

point. Slope quantifies how rapidly the y value changes as the x variable changes—in this case, how rapidly error changes as a function of changes in a connection weight. Two such hypothetical slopes are shown in figure 2.7, for two different weight values (w_1 and w_2). If the slope for a given weight value and error can be computed, then the direction in which the connection weight should be changed is obvious. If the slope is currently negative (as at w_1), then the weight should be increased to move error toward the minimum. Conversely, if the slope is currently positive (as at w_2), then the weight should be decreased to move error toward the minimum.

The greater dilemma is figuring out how much to change a connection weight. If the weight is changed too much, the deepest part of the error valley could be missed, creating an oscillation in weight adjustments that never settles into the minimum error. The general solution is to take very tiny steps of weight change so as not to miss the minimum error, but this makes learning quite slow. Knowing the slope can also be helpful in deciding how much to change a weight. The idea is to make the amount of weight change proportional to slope. With a currently steep slope, as at w_1, a rather large change in weight is called for. With a currently shallow slope, as at w_2, weight change should be rather small, as the error minimum may be nearby. This technique of using information on the slope of the error function is often known as gradient descent because it involves trying to slide downhill on the error surface to reach the point of minimum error in weight space.

Thus, being able to compute the first derivative of error with respect to weight offers a tremendous advantage in getting feed-forward neural networks to learn from examples in the training set. Generally, the amount of weight change is considered to be a negative proportion of the partial derivative of error with respect to weight:

$$\Delta w_{ij} = -r\frac{\partial E}{\partial w_{ij}} \tag{2.5}$$

The parameter r in this equation represents the learning rate and is generally set to a moderate proportion, such as 0.5, to keep the weight adjustments from oscillating wildly across the minimum error (McClelland & Rumelhart, 1988).