1 Learning and Connectionist Representations

David E. Rumelhart and Peter M. Todd

1.1 INTRODUCTION: REPRESENTATIONAL TOOLS IN CONNECTIONIST NETWORKS

Connectionist modeling is undergoing a renaissance. As the merits of brain-style computation (Rumelhart 1990) have become apparent, a bewildering variety of connectionist applications has cropped up throughout the cognitive sciences and engineering (for instance, see Lippmann, Moody, and Touretzky 1991). One of the central issues in all of these models is the representation of knowledge in the connectionist network. Getting a coherent picture of "what goes on" inside a network as it develops, manipulates, and alters the representation of the knowledge it processes is vital for our understanding of connectionist information processing, and likely for our understanding of the minds these systems model. In this chapter we explore the sorts of representations that connectionist systems employ and the crucial role learning plays in constructing them.

The representational materials that connectionist systems have to work with are remarkably simple. A connectionist network can basically be described as a collection of simple processing units, each of which has a current *state of activation*, linked together by a set of connections, each of which has a current *strength* or *weight* (see fig. 1.1). The weighted connections modify the activation values that they pass among the processing units. One set of units is typically assigned the special role of receiving inputs from the "external world"; these input units have their activation values set (at least in part) by an external stimulus. Another set of units is usually designated as the outputs of the system; the activation values of these output units is monitored as the final result of processing by the network. Units that fall into neither of these classes are often called hidden units, and they play a key role in the representation of knowledge in the network. We can further abstract the system's description to a simple vector $A[i]$ of activation values of all the units in the network, and a matrix $W[i, j]$ of the weights on the connections between units in the network. From this standpoint, there are now basically just two representational formats in which information is held in a connectionist network.

The first is the overall pattern of activation, $A[i]$, across all the units of the network. This pattern of activation corresponds to the *state of processing* of

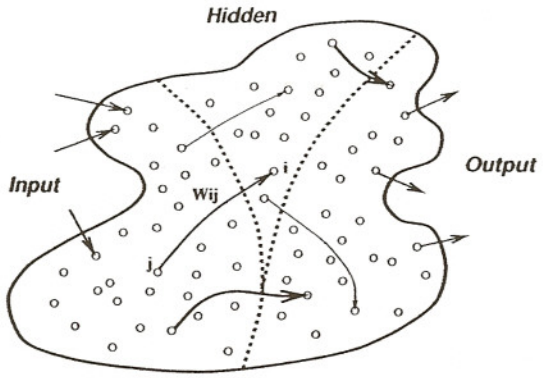

Figure 1.1 Generalized structure of a connectionist network, showing the input, hidden, and output processing units (e.g., *i* and *j*) and weighted interconnections (e.g., *Wij*).

the system at any point in time and determines what information is currently being represented and acted on by the network. It is important to note that this representational format achieves its meaning not through its internal structure, but through its relationship to the structure and meaning of the inputs and outputs of the system. While we usually impose a semantic interpretation on the network's inputs and outputs, the network is typically under no constraints to develop internal representations that we can comprehend and instead will come up with whatever best fits the task (i.e., the input-output mapping) at hand. Thus there may be no simple interpretation of the network's overall pattern of activation in terms easily understood by a human observer. The meaning of the pattern is rather determined by what causes it and what it causes (i.e., the network's inputs and outputs, respectively).

The second representational format in a connectionist system is the pattern of connectivity in the network, that is, the matrix $W[i, j]$ of connection strengths among the units of the network. This pattern is even more difficult to interpret than the pattern of activation in the network. Yet the pattern of connectivity is, in a sense, the more critical representational tool in connectionist networks, since it corresponds to the current *state of knowledge* of the system. The knowledge stored in the network's connectivity determines the effects of the inputs on the system's overall pattern of activation, and the effects of that activation on the behavior and outputs of the system as a whole. Thus, the network's pattern of activation can be thought of as simply a consequence of the pattern of connectivity of the system. As we will see, learning serves to modify the connection strengths in the network—its knowledge—so that it will produce the proper activations (particularly at the outputs) in response to various inputs—its processing.

These connectionist representational tools may seem at first unduly weak when compared to the formal logic predicates, semantic networks, frames and scripts, and other relatively sophisticated schemes for representing knowledge developed over the years in artificial intelligence (AI) and cognitive science (cf. Rumelhart and Norman 1988, Brachman and Levesque 1985). But there are

in fact great advantages inherent in these simpler representational devices. Before turning to a detailed discussion of connectionist representation methods, it is useful to gain a historical perspective on this issue, by looking briefly at the history of representational systems and the (negative) correlation between interest in learning systems and interest in complex representation schemes. Many early AI systems, such as Oliver Selfridge's Pandemonium model of pattern recognition (Selfridge 1959), Frank Rosenblatt's perceptron model (Rosenblatt 1962), and the checkers-playing system of Arthur Samuel (Samuel 1959), had a strong focus on learning but a relatively weak and simple representational format. As interest in more sophisticated representation schemes grew, interest in learning per se correspondingly diminished. The idea was roughly that if we didn't know how the information was ultimately to be represented, we couldn't know what exactly should be learned in the first place. This view appears to have dominated most AI research during the 1970s.

With the 1980s, however, came some change of heart. Many researchers have recently developed increasingly simplified representational systems for which relatively simple learning procedures can be defined. One example of such work is the Soar system developed by Allen Newell and his colleagues (Newell 1990). They dropped many of the most complicated aspects of 1970s-era representation schemes and retained only a rather streamlined representational system, further enabling them to develop appropriate learning rules for the system. The connectionist paradigm goes one step further in simplifying its representational system and focusing even more on learning as a mechanism for developing representations. Given the large numbers of units and connections in a typical network, it is virtually impossible to "hand-wire" a connectionist system capable of very sophisticated behavior. This results in an increasing dependence on learning rules and mechanisms to "wire up" networks, and thereby to determine the details of connectionist representational formats. In the remainder of this chapter, we will focus on the distinctions made among the classes of such representations, the reasons and mechanisms for learning them, and some of the examples of their usefulness in connectionist systems.

1.2 DISTRIBUTED VERSUS LOCALIST REPRESENTATIONS

Perhaps the simplest representational scheme within the connectionist framework is the *localist* representation system (cf. Feldman and Ballard 1982). In this case, each unit in a network corresponds to a single concept. This one-concept–one-unit representational system has the advantage of simplicity and clarity. First of all, it is easy to see how the units should connect to one another—"hand-wiring" of localist networks is little problem. Furthermore, when a set of concepts is to be represented, we simply turn on each of the units corresponding to those individual concepts. McClelland and Rumelhart employed such a localist scheme in a word perception model designed to account for the word-superiority effect, the phenomenon in which letters are

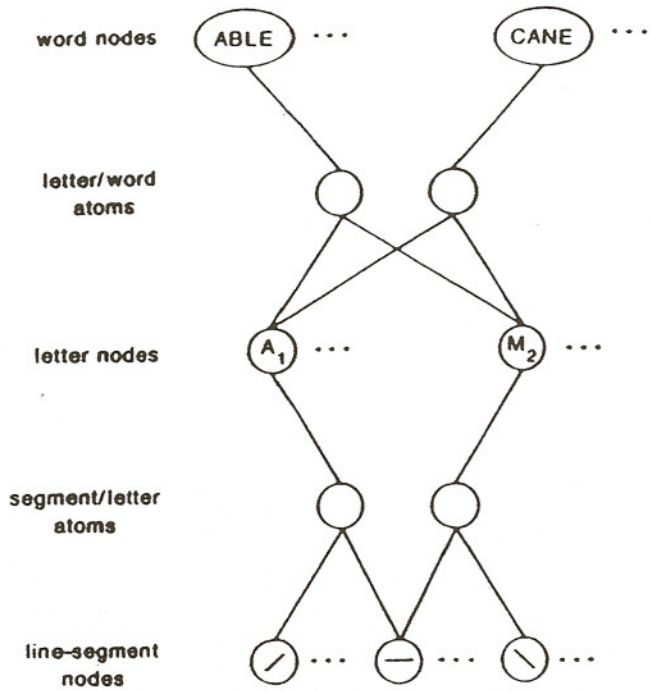

Figure 1.2 Example of a connectionist network using localist representations for word perception from letters and their subfeatures.

more easily recognized when presented in the context of normal words than in random letter strings (McClelland and Rumelhart 1981; Rumelhart and McClelland 1982). As seen in figure 1.2, this network had three sets of interconnected localist units: features, letters, and words. Connecting the units in this network was quite straightforward. Visual feature units were connected to the letters that contained them (e.g., vertical and horizontal bar detecting units were connected to the "T" letter unit), and letter units were similarly connected to the words in which they were found (e.g., "T" and "R" were connected to the word unit representing "CART"). Very useful results were obtained from this relatively simple model; but, as is typical of localist systems, the representations it used were all predetermined and of little interest compared to its processing behavior.

An alternative and rather more powerful representational system is the *distributed* representation format (described in detail in Hinton, McClelland, and Rumelhart 1986). In this case, a concept is represented not by the activation of a single unit, but rather by a pattern of activation over several units at once. Thus, each concept incoporates many units, and each unit participates in the representation of many concepts. It is useful to think of the individual units as representing what Hinton calls *microfeatures* (Hinton, McClelland, and Rumelhart 1986). A particular concept, then, is represented by a particular pattern of microfeatures. This leads to a natural measure of similarity between

concepts: two concepts are similar to the extent that they share the same set of microfeatures. The key role of similarity as a fundamental processing strategy makes distributed connectionist systems unique, as we will see in the next section.

Because learning in a distributed-representation network occurs as modification of connections among microfeatures rather than among concepts directly, generalization and transfer of learning between concepts is inescapable. Increasing or decreasing connection strengths among the microfeatures of one concept inevitably affects the representation—and thus the meaning and consequences—of other concepts. This transfer of learning or generalization process can be both good and bad: good in that similar concepts should generally be responded to similarly, but bad, however, when important distinctions must be made between concepts that are very similar. In the former case, the system must learn which common microfeatures similar concepts share, while in the latter case, it must learn distinctive microfeatures that differentiate otherwise similar concepts. Learning is thus a key means of discovering relevant representations in both cases.

Another important aspect of distributed representations is their ability to represent a very large number of potential concepts in terms of a finite number of representational elements. If we use localist representations, with one unit corresponding to one concept, then with n units we have a vocabulary of n possible concepts. This finiteness of concept vocabulary is somewhat worrying—the combinatorics of the situation are definitely not in our favor. However, in a distributed representation we can have an enormous vocabulary of possible concepts with a relatively small number of representational elements. For example, if we have n binary units (which can take on activations of 0 or 1 only) available, we have a vocabulary of 2^n concepts we can possibly represent. For even small values of n, such as 100 or 1000, the vocabulary of possible concepts is 2^{100} or 2^{1000}—enormous numbers by any measure.

Of course, we pay a price for this large vocabulary. In a distributed representation we can represent any one of a very large number of concepts at one time, but we cannot represent many concepts at the same time. In a localist representation, as we mentioned earlier, we can simultaneously represent any desired combination of the possible individual concepts. Generally, in the parallel distributed processing (PDP) connectionist framework, we believe that the advantage of a large vocabulary of possible distributed concepts outweighs the advantage of being able to simultaneously represent arbitrary combinations of localist concepts. And it is still possible to represent a number of distributed concepts simultaneously, provided that number is reasonably small.

Perhaps the most serious disadvantage of distributed representations is our inability to easily interpret the complex patterns developed over the course of learning. It is this difficulty of interpretation and understanding that makes distributed representations seem so mysterious. But as we will demonstrate in the following sections, the patterns need not always be so enigmatic.

1.3 LEARNING REPRESENTATIONS IN CONNECTIONIST NETWORKS

Figure 1.3 illustrates a very simple connectionist network. It consists of two layers of units, the input units and output units, connected by a set of weights. As a result of the particular connectivity and weights of this network, each pattern of activation presented at the input units will induce another specific pattern of activation at the output units. This simple architecture is useful in a number of ways. If the input and output patterns all use distributed representations—namely, if they can all be described as sets of microfeatures—then this network will exhibit the important property mentioned in the previous section that "similar inputs yield similar outputs," along with the accompanying generalization and transfer of learning. Such two-layer networks behave this way because the activation of a particular output unit is given by a relatively smooth function of the weighted sum of its inputs. Thus, a slight change in the value of a particular input unit will yield a similarly slight change in the values of the output units.

Although this similarity-based processing is mostly useful, it does not always yield the correct generalizations. In particular, in a simple network of the kind shown in figure 1.3, the similarity metric employed is determined by the nature of the inputs themselves. And the "physical similarity" we are likely to have at the inputs (based on the structure of stimuli from the physical world) may not be the best measure of the "functional" or "psychological" similarity we would like to employ at the output (to group together appropriate similar responses). For example, it is probably true that a lowercase *a* is physically less similar to an uppercase *A* than to a lowercase *o*, but functionally and psychologically, a lowercase *a* and an uppercase *A* are more similar to one another than are the two lowercase letters. Thus, physical relatedness is an inadequate similarity metric for modeling human responses to letter-shaped visual inputs. It is therefore necessary to transform these input patterns somehow from their initial physically derived format into another representational form in which patterns requiring similar (output) responses are indeed similar to one another. This involves learning new representations.

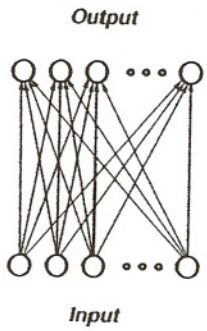

Figure 1.3 A simple two-layer connectionist network consisting solely of input units connected to output units.

Figure 1.4 illustrates a simple layered feedforward network, in which information (activation) flows up from the input units at the bottom, through successive layers of internal ("hidden") units, to create the final response at the layer of output units on top. Such a network is useful for illustrating how an appropriate psychological or functional representation can be created. If we think of each input vector as a point in some multidimensional space, we can think of the similarity between two such vectors as the distance between their two corresponding points. Furthermore, we can think of the weighted connections from one layer of units to the next as implementing a transformation that maps each original input vector into some new vector. This transformation can create a new vector space in which the relative distances among the points corresponding to the input vectors are different from those in the original vector space, essentially rearranging the points. And if we use a sequence of such transformations, each involving certain nonlinearities, by "stacking" them between successive layers in the network, we can entirely rearrange the similarity relations among the original input vectors.

Thus, a layered network can be viewed simply as a mechanism for transforming the original set of input stimuli into a new similarity space with a new set of distances among the input points. For example, it is possible to move the initially distant "physical" input representations of a lowercase *a* and an uppercase *A* so that they are very close to one another in a transformed "psychological" output representation space, and simultaneously to transform the distance between the lowercase *a* and a lowercase *o* output representations so that they are rather distant from one another. (Generally, we seek to attain a representation in the second-to-last layer that is sufficiently transformed for us to rely on the principle that similar patterns yield similar

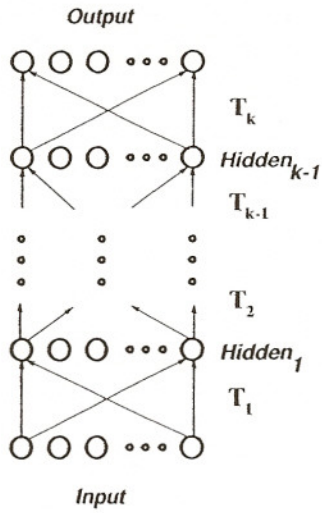

Figure 1.4 A multilayer connectionist network with several hidden layers interposed between the input and output layers; each hidden layer computes a transformation **T** of the representation structure.

outputs at the final layer.) The problem is to find an appropriate sequence of transformations that accomplish the desired input-to-output change in similarity structures.

The *backpropagation* learning algorithm is a simple procedure for discovering such a sequence of transformations. In fact, we can see the role of learning in general as a mechanism for constructing the transformations that will convert the original physically based configuration of the input vectors into an appropriate functional or psychological space, with the proper similarity relationships between concepts for making generalization and transfer of learning occur automatically and correctly. The details of the backpropagation learning procedure are described in Rumelhart, Hinton, and Williams (1986) and will not be repeated here. However, the basic idea is the following: A performance measure is defined for how well a network satisfies some set of predefined criteria, such as producing a desired output for a given input. Backpropagation works by simply computing the gradient of this performance measure with respect to each of the weights in the network and then modifying those weights to improve the performance measure on a subsequent learning trial (e.g., the next input–desired output pair). This simple gradient search method has turned out to be quite effective in developing new and appropriate representations for many rather complex problems, by constructing similarly complex sequences of transformations.

In the remainder of the chapter, we will describe three different examples of the application of such a simple learning procedure and illustrate the interesting and useful representations it can create.

1.4 AUTOENCODERS

Perhaps the simplest mechanism for creating new and useful representations is the autoencoder.[1] The basic architecture of the autoencoder is illustrated in figure 1.5. The network has what may be called an hourglass architecture. The input and output vectors are of high and equal dimensionality—that is, there are n input units and n output units, where n is relatively large. The middle layer of hidden units has much lower dimensionality, containing m units, with $m \ll n$. The target for the output units is simply the input vector. Thus, the goal of an autoencoder is to take a high-dimensional input vector, recode it in a lower-dimensional space, and then use this low-dimensional representation to reproduce the original input vector again. This process of recoding is similar in many respects to ordinary principle components analysis (PCA). However, since the hidden units and output units are usually nonlinear, the network performs a nonlinear variant of PCA.

To illustrate how an autoencoder network can find a useful re-representation of its input, consider the set of visual patterns illustrated in figure 1.6. Imagine the set of characters that can be created out of the four line segments shown in the figure. We can create the letter *O* by having all four line segments on. We can create the letter *L* by turning on left and bottom line segments. The letter *C* consists of the top, bottom, and left line segment, and so forth.

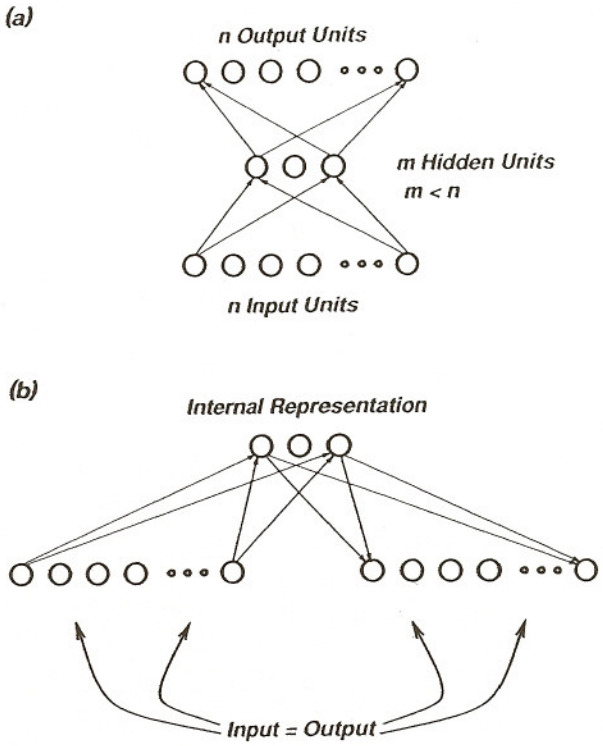

Figure 1.5 Basic structure of an autoencoder network (a) showing its hourglass architecture, and (b) emphasizing the identity of inputs and outputs.

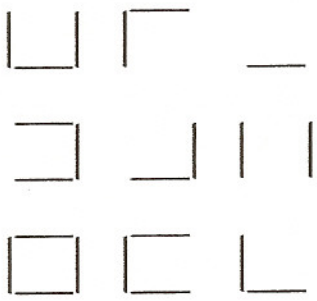

Figure 1.6 Examples of characters created from a set of four line segments, for use in an autoencoding experiment.

There are a total of sixteen possible images that can be constructed out of the four line segments. This includes the image in which no line segments are on. Now imagine that we use a 10 × 10 square array of pixels as our input and simply turn on just those pixels along each edge of the square corresponding to the line segments in the desired character. In this experiment, we know a priori that all sixteen possible input characters can be created out of four underlying features (line segments). It doesn't matter that the inputs are one hundred-dimensional vectors—there is a simple and much more compact representation of the set of input vectors that can actually occur. We want to see if a network can actually learn this simpler representation.

In a series of experiments involving a network with one hundred input units, four hidden units, and one hundred output units, trained on all sixteen characters as input and output using the backpropagation learning algorithm, we found that the four hidden units always turned out to correspond to the four underlying line segments. The autoencoder had effectively extracted the underlying structure of the population of input (and output) vectors and had learned the proper connection weights needed to implement the transformations back and forth between the full 100-dimensional vector representation and the reduced four-dimensional representation. In a variation on this experiment we used noisy input character vectors. In this case, the input essentially consisted of the desired line segments superimposed on a field of noise, so that each element of the input vector had a .25 probability of having the wrong value. (That is, units that were supposed to be on would be off 25 percent of the time, and vice versa.) This generated a very messy set of input vectors for the network to try to learn. But since the noise was random and unpredictable, it could not be extracted in the form of stable features of the input to be used in the reduced representation. Rather, we found that the four hidden units again picked up the four reliable underlying features—the line segments—of the character set.

This robust behavior under noisy data conditions is a further demonstration of the ability of PDP connectionist systems to learn useful distributed representations and to generalize and transfer that learning appropriately between inputs. In particular, with the autoencoder we have a system capable of extracting those features from the input vectors that are best able to predict the structure of the entire vector. Since the noise in the previous example was unpredictable it was ignored, and only the useful features were learned. In this way, the network builds representations that have as much information as possible about the input patterns, but that are also as concise as possible.

With slight modifications in the hourglass network architecture, we can build what are basically autoencoders that ignore other irrelevant aspects of the input. Imagine, for instance, that we have a large visual field that somewhere contains a single much smaller character of the type described earlier in this section. We may be interested in the character itself, without caring where it occurs—in essence, we want a sort of position-invariant attentional mechanism that represents the *type* of character and ignores its spatial location. A network that will do precisely this is shown in figure 1.7. The input visual field

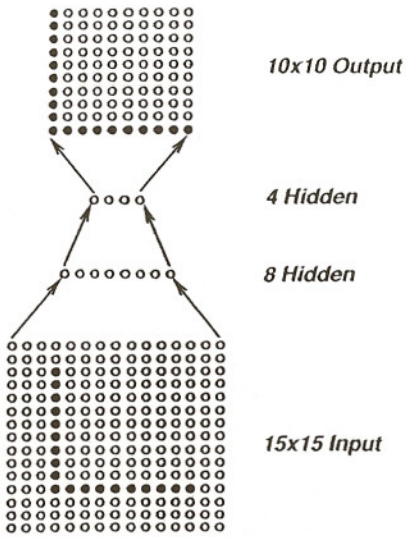

Figure 1.7 Modified autoencoder architecture for abstracting location-invariant features from a character in a larger visual field.

is a 15 × 15 array, in which our 10 × 10 character can appear at any of 36 (6 × 6) positions. The output is the same 10 × 10 character in canonical position. Feeding into the output is our layer of four hidden units, as before. But we also add one more layer to this network, consisting of eight hidden units, interposed between the input array and the four-unit hidden layer. The task of this second hidden layer is to create an intermediate representation of the input visual stimuli in which the position information has been discarded. The four-unit hidden layer can then transform that representation into one that is more useful at the output layer. This network was trained using only eleven of the sixteen possible characters, those that have two or more segments so that their position in the input array can be unambiguously determined. Each of these characters was presented at each of the thirty-six positions in the input, for a total of 396 training patterns. Once the network had learned to perform this task suitably well, we found that as before the four-unit hidden layer had adopted a one-unit-per-line-segment representation. Thus, again only the information needed for the task at hand—reconstructing the character and ignoring its position—was abstracted by the network for its auto-encoded representation.

1.5 REPRESENTING SEMANTIC NETWORKS IN CONNECTIONIST SYSTEMS

One of the classic representational systems much studied in the 1970s was the *semantic network* (Quillian 1968; Collins and Quillian 1970). Figure 1.8 shows a fairly standard example of a semantic network, represented as a tree structure that can be interpreted as a concise summary of a large number of individual

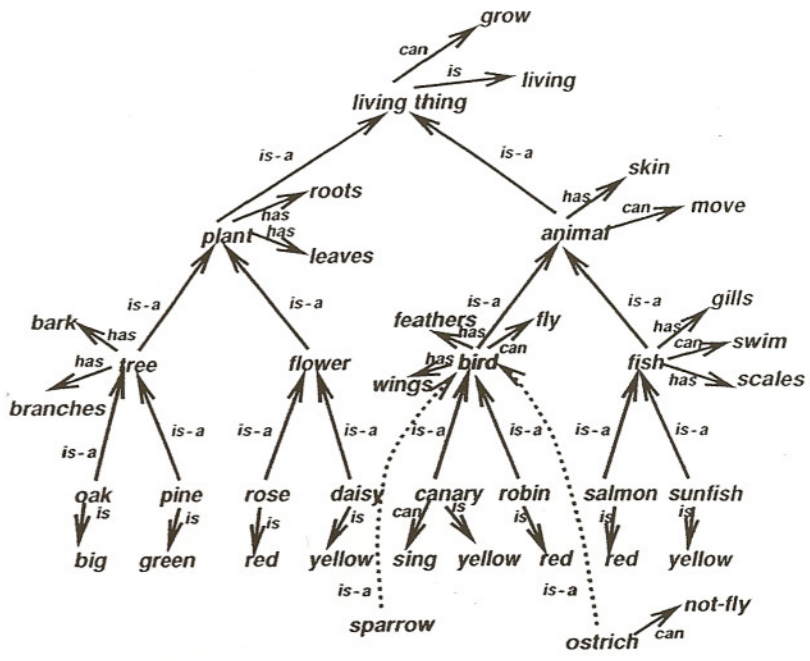

Figure 1.8 A semantic network encoding explicit and implicit ("inherited") facts about plants and animals.

facts. For example, the semantic network in figure 1.8 directly encodes the fact that a canary is a bird, a bird can fly, a bird is an animal, an animal has skin, an animal is a living thing, and a living thing can grow. It also contains the fact that a canary can fly, a canary has skin, and so forth. These secondary facts are implicit in the network through the principle of *inheritance*: if an x is a y and a y has a certain property, then generally speaking x can be said to have inherited that property. Inheritance allows a great number of implicit facts to be captured in a semantic network. How can we represent this collection of facts and inferences in a connectionist network?

In answering this question, one might first be tempted to design a localist representation in which each term or node of the semantic network such as *canary* or *bird* or *sing* is represented by a single unit in a connectionist network; then when the *canary* unit is activated, perhaps it would activate the units for *sing* and *yellow* and *bird* and *animal* and so on. This way of modeling semantic networks may be straightforward and easily followed, but it adds little to the representational characteristics of the traditional semantic network. (But see Shastri 1988a, b for an essentially localist connectionist instantiation of a semantic network that shows useful processing abilities.)

A more interesting approach is to represent the semantic network's information in a distributed fashion. With this approach, the problem becomes how to represent terms such as *canary* and *robin* and relations such as *is-a*, *has*, and *can* as distributed patterns in our connectionist network. One possible solution is shown in figure 1.9—a simple connectionist network architecture designed to

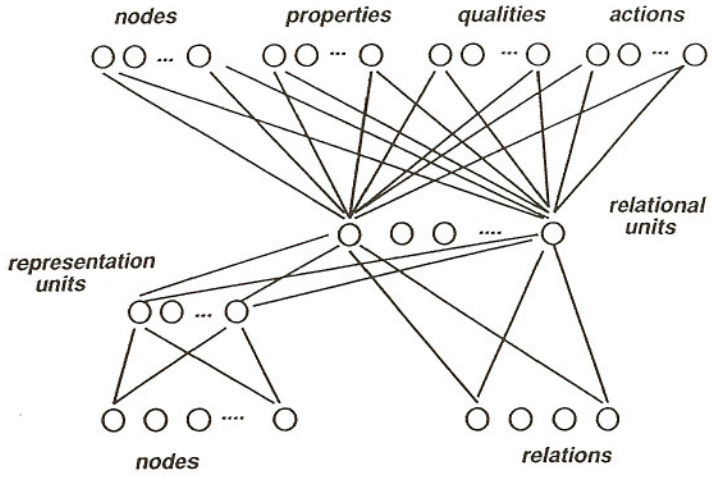

Figure 1.9 A connectionist network designed to learn distributed representations for the facts contained in a semantic network.

learn the appropriate distributed representations for the concepts in a semantic network. The idea is to train the network to map from concept-relation pairs as inputs to a listing of all terms that stand in that relation to that concept as outputs. For example, given *canary* and *can* as inputs, we want to produce *sing, fly, move,* and *grow* as outputs. Similarly, given *daisy* and *is* as inputs we want *yellow* and *living* as outputs, and *tree* and *has* as inputs should produce *bark, leaves,* and *roots.* If we represent these inputs and outputs in a localist fashion, then in order to perform this mapping the network will have to learn to reencode the localist representations of the inputs into appropriate distributed representations at the hidden layers from which the outputs can be produced. Hinton (1986) has employed a similar method to learn about family relationships.

To get the network to perform this mapping, we trained it in the following way. First, all the facts contained in the semantic network (both explicitly, and implicitly via inheritance) were converted into input/output training pairs. Each fact in the semantic network has the form *term1-relation-term2*; to convert these into training pairs for the network, we essentially collected all the facts that related to a specific *term1* and *relation* and compressed them into a single statement of the form *term1-relation-*⟨*set of terms*⟩. Thus the facts *bird has feathers* and *bird has wings* were compressed into the single statement *bird has* ⟨*feathers and wings*⟩. These compressed facts were turned into the training pairs by using the first two parts, *term1* and *relation,* as inputs, and the last part, ⟨*set of terms*⟩, as the outputs.

After constructing these input/output pairs, training the network then consisted of turning on the "node" input unit corresponding to *term1* (e.g., *bird, fish, daisy,* etc.) and the "relation" input unit corresponding to *relation* (i.e., *is-a, has, is,* or *can*) (see fig. 1.9). The output units corresponding to each of the terms in ⟨*set of terms*⟩ were also turned on. Depending on the relation

type turned on in the input, the outputs turned on would all be in one of the four clusters of output units corresponding to nodes, properties, qualities, or actions in the semantic network. For example, if the relation type was *is-a* in the input, then only node output units would be turned on (e.g., *living-thing*); if the relation type *has* was turned on, then some set of the property units would be turned on (e.g., *feathers*); the relation *is* would turn on qualities (e.g., *red*); and the relation *can* would turn on actions (e.g., *swim*). The network was trained on input-output pairs of this type by backpropagation until it was able to produce correctly the appropriate pattern over the output units for each input pattern. This required several hundred pairings of each input-output pair.

The network architecture used here is slightly more complicated than a standard three-layer form. The node input units project first onto a layer of hidden units called representation units, before being combined with the relation inputs at the second hidden layer, made up of what we call relational units. This structure is used because we want the first layer of weights to transform the localist input representations of the node terms into distributed representations of the concepts involved, at the representation units. These distributed representations interact with the relational inputs through the layer of relational hidden units and then finally project onto the output units.

The distributed representations of the node input concepts developed at the representation unit hidden layer are our main concern in this experiment. We expect that similar concepts will have similar representations at this transformed level, in spite of the fact that at the input level all concepts are equally similar to one another (since they are represented in a localist manner and thus have no inherent differential similarity). This representational structure will develop because similar responses at the output level are to be given to similar concepts at the input level, but since this similarity is not captured in the input representation, it must be introduced in the distributed hidden-layer representation. Thus, for example, on the whole *oak* and *pine* are to be responded to similarly. As a result, the distributed representation of *oak* and of *pine* should be very similar.

Figure 1.10 shows the representations developed from one experiment, in which we used eight representation hidden units. The figure is essentially the semantic network "tree" of concepts laid on its side. Thus, to the left we see the leaves of the tree—*oak, pine, rose, daisy,* and so forth; to their right we find the representations for the next higher order terms—*tree, flower, bird,* and *fish;* then further right are the still higher order terms—*plant* and *animal,* and finally at the far right we have the representation for *living-things.* The representations are shown in terms of the weights from the node input units to the representation hidden units. Positive weights are indicated by pluses, negative weights by minuses, and weights near zero by "?"s. What we find is that the major conceptual dimensions of the semantic network are represented by particular features in the connectionist network. For example, the first feature is positive for all plants, and negative (or at least *not* positive, in the case of *fish*) for all animals. Thus, feature 1 seems to represent the plant/animal distinction (though it is purely chance that the *first* unit out of all eight picked up this

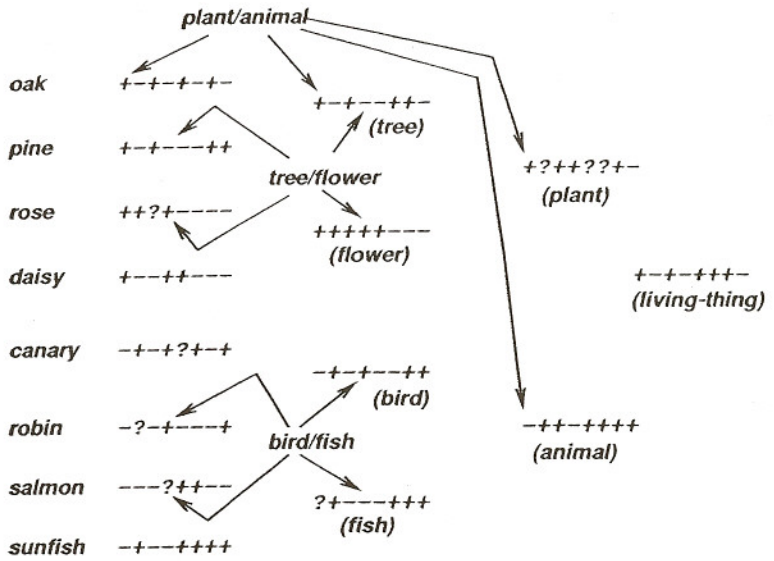

Figure 1.10 The representations for various concepts developed by a connectionist network with eight hidden units, showing interpretations of some of the microfeatures.

feature, as we will see in the next example). Similarly, if we are representing a plant, then feature 4 captures the tree/flower distinction; and interestingly this very same feature (hidden unit 4) encodes the bird/fish distinction in the case of animals. This ability for the same unit to represent different things conditional on the value of some other unit is a useful way in which connectionist representations differ from simple feature lists. By using hidden units for "double-duty" (or triple or more) conditional encoding of microfeatures, distributed representations can be much more compact than feature-list representations that have different slots allocated in the case of different concept types.

Figure 1.11 shows the results of another experiment, this time with six rather than eight representation hidden units. Here again we see that one feature, in this case the fifth, represents the plant/animal distinction, while feature 1 represents the bird/fish distinction (for animals) and feature 4 the tree/flower distinction (for plants). (The doubling up onto the same hidden unit we saw for these features in the previous example is absent here; there are enough hidden units for the representations that doubling up is not required but may happen in some cases nonetheless.) The other three features are rather more difficult to interpret and appear to represent the remaining idiosyncratic characteristics of the various node concepts.

One of the primary features of the semantic network is its inheritance property, as we mentioned earlier. New information can be added to the network and it will "inherit" many additional facts. For example, we could add the fact that a *sparrow is-a bird* to the semantic network, as indicated in figure 1.8 by the dotted line (an *is-a* link) from *sparrow* to *bird*. If we know that a *sparrow is-a bird*, we also immediately know by inheritance from *bird*, *animal*, and *living-thing* that a *sparrow can fly*, a *sparrow has feathers*, a *sparrow has skin*,

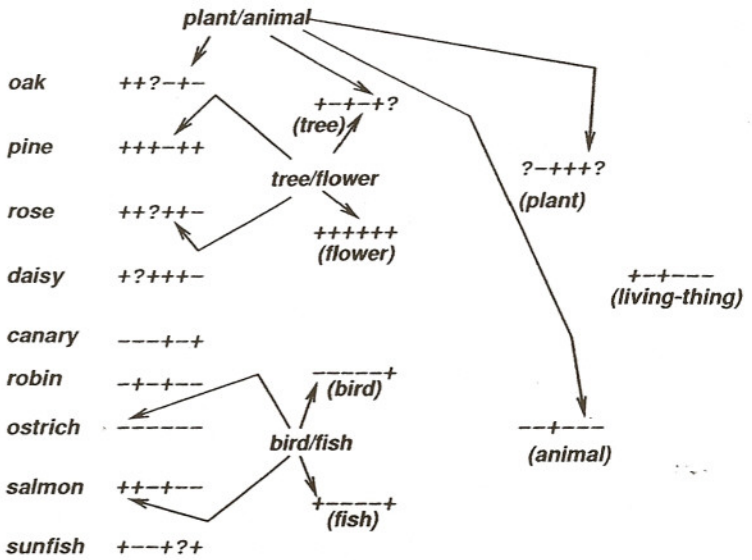

Figure 1.11 Representations developed by another connectionist network, with six hidden units.

a *sparrow can grow*, and so on—all of which is captured naturally and directly by the semantic network representation. Now switching back to our connectionist model we can ask, if we train the PDP network on the fact that a *sparrow is-a bird*, will it correctly generalize to these additional characteristics?

To answer this question, we first trained the six-representation-unit network without any initial concept of *sparrow* (but with an unused node input unit for later use) and achieved the representations illustrated in figure 1.11. After this, we further trained the network on just the proposition that a *sparrow is-a bird*. This was achieved by presenting *sparrow* and *is-a* as the two inputs (turning on the previously unused node input unit to stand for *sparrow*) and *bird* as the desired output. The difference between the actual activation on the *bird* output unit and the desired output (1.0) was backpropagated as the error signal to use in modifying the weights in the network. The error from the *bird* output unit was the only error signal used in training on this additional fact. Furthermore, only the weights from the new *sparrow* input unit to the representation hidden units were changed during learning—all other weights in the network were kept fixed, so that the other representations would not be adversely affected by this further training.

Now in order for *sparrow* to turn on *bird*, the pattern over the representation units for *sparrow* must be similar to the patterns for other node input concepts that turn on *bird*. This is in fact what happens, as we find in the representation developed for *sparrow* illustrated in figure 1.12. We see that *sparrow* has developed a pattern roughly halfway between *canary* and *robin*, perhaps a bit closer to the latter. The figure also shows the generalization results. Although the network had not been taught anything explicitly about what a *sparrow* has or can do (nothing in fact other than that it *is-a bird*), it correctly inferred that a

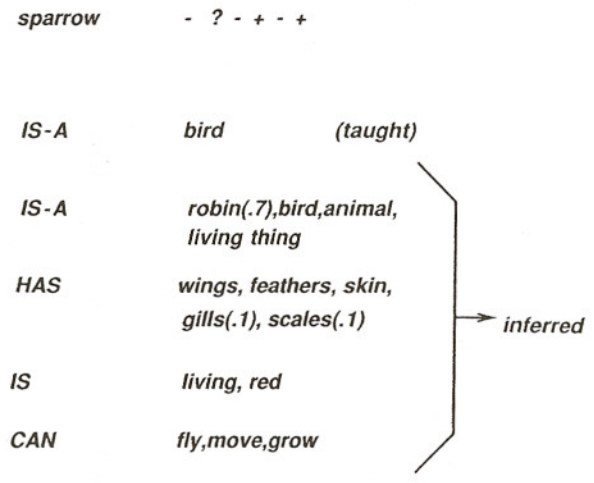

Figure 1.12 Representation developed for a new *sparrow* concept, showing its resulting inferred attributes.

sparrow can fly, can move, can grow, has wings, has feathers, has skin, is-an animal, is-a living-thing, is living, and *is red* (a good guess as to its color). The network's representation of *sparrow* also gives a strength of 0.1 for having *gills* and *scales* (erroneously, but ignorably) and a value of essentially 0.0 for every other inappropriate quality, property, and action. Thus, we see that the connectionist network provides a rather good generalization mechanism that emulates the inheritance mechanism of the semantic network, turning on the expected implications of a new concept and ignoring the low-likelihood ones. But, importantly, the connectionist network achieves all this using a very different similarity-based learning and processing mechanism.

Another feature of semantic networks is the so-called cancellation principle. This is illustrated by the example of the ostrich. Although the ostrich is a bird, the ostrich cannot fly; therefore, the normal inheritance mechanism must somehow be thwarted in this case. This can be done in traditional semantic networks by adding the direct characteristic to *ostrich* that it *can not-fly* (as shown in fig. 1.8), and by adding the general principle that no inherited property can override an explicitly stated property of a concept. (We add the *not-fly* action here rather than adding a new type of *cannot* link.) How does the connectionist network emulate this principle? Quite simply—we just train the network with the fact that an *ostrich is-a bird* and has all the usual *bird* attributes, except that we explicitly teach the network that an *ostrich can not-fly*. We do this by turning off the *fly* output unit (giving it a desired activation value of 0.0) when *ostrich* and *can* are given as inputs. Figure 1.11 illustrates the representation thereby attained for *ostrich*—in this case all minuses. This representation captures all the desired facts (and the new exception) about ostriches.

Having trained the network on *ostrich*, we then wanted to see how it generalized to other flightless birds. We further taught the network about a

new thing we called an *emu*: an *emu is-a bird*, but an *emu can not-fly*. We trained the network on these two facts about this new creature as we did for the *sparrow*, by sending error signals back only from the *bird* and *fly* output units and changing only the weights on the connections from the new *emu* input unit to the representation hidden units. We found that the network discovered a representation for *emu* that was exactly the same as that for *ostrich*. Therefore its responses to all queries were also the same as those for *ostrich*: it said that an *emu is large, has feathers, has wings, is-an animal, is-a living-thing*, and so on. Since the *ostrich* was the only example of a flightless bird that the network knew about, it simply assimilated *emu* to *ostrich* and thereby gave the right answers to essentially every query about emus.

We ran one final experiment in which we presented the network with still another flightless bird—this time the penguin. We taught the network that a *penguin is-a bird*, that a *penguin can not-fly*, but that it *can swim*. In this case we found that the network mistakenly generalized by asserting that, in addition to being a *bird*, the *penguin is-a fish*, and that it *has gills* and *has scales* as well as *has feathers* and other *bird* attributes. Not happy with this result, we tried again. This time we taught the network explicitly that the *penguin is-a bird*, the *penguin is-a not-fish*, the *penguin can not-fly*, but the *penguin can swim*. We found that the network could not both turn on the output unit for *swim* and turn off the unit for *fish*. That is, by modifying weights only from the penguin input unit to the representation hidden units, we were unable to make the system differentiate between being able to *swim* and being a *fish*. Presumably this is because the network had learned that whenever anything *is-a fish*, it *can swim*, and vice versa, and it made use of this redundancy to form appropriate generalizations. The only way the network could assimilate this new swimming penguin fact was to allow it to modify other connections—then it could eventually learn this peculiar proposition.

The case of the penguin illustrates another important feature of connectionist networks: they work by representing certain classes of concepts as similar to one another, and by exploiting the redundancies among the characteristics of the concepts within a class to make generalizations. Usually these generalizations are appropriate, as when the network responded the same way for *emu* and *ostrich*, but sometimes they are not, as when the network had a hard time learning that penguins can swim but aren't fish. The network essentially ends up reflecting the structure of the world (as we humans parse it, since we make the training sets). To the extent that there are correlations or commonalities among the (micro)features we perceive for various fishes or various birds, the network will be able to correctly generalize between members of these categories.

Finally, before leaving this semantic network example, it will be useful to look at another way of viewing the representations developed in these experiments. Figure 1.13 shows a hierarchical clustering view of the hidden unit representations for still another experiment in this series. Here we see that on the whole, animals are clustered separately from plants, trees are clustered separately from flowers, and fish are clustered separately from birds. In addi-

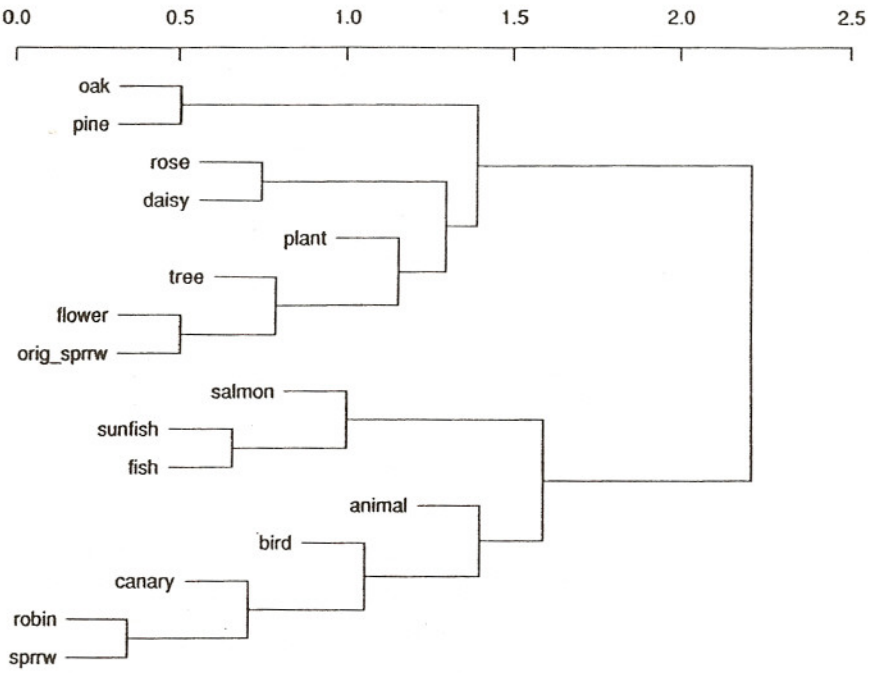

Figure 1.13 Hierarchical clustering of the representations developed by a connectionist network, showing similarity and hierarchy relations among the concepts.

tion, the most similar pairs in the figure are those of the same type at the same (low) level in the semantic network hierarchy (e.g., *oak* and *pine*). Finally, we can see how the representation of one concept, the *sparrow* described earlier, changed during the course of learning. In the already fully trained network, the original representation of *sparrow* was determined simply by the random initial setting of the weights from its corresponding input node unit. This ended up by chance to be very close to the learned representation of *flower*, as shown. However, after the previously trained network was further taught that a *sparrow is-a bird*, we see that the *sparrow* representation moved to be very close to that for *robin*, as we illustrated in figure 1.12.

We've seen in this example that connectionist learning algorithms can create new representations that abstract the important features from a set of input-output relations. A trained network can use the features in this transformed representation to provide appropriately generalized responses when it learns about new concepts. Knowing only one characteristic of a new concept—for example, that it *is-a bird*—the connectionist network can correctly infer its other major attributes. Although this behavior is compelling and suggestive, it is of course unrealistic to imagine that humans learn and develop concepts based on just this sort of mechanism. Our own concept formation processes are no doubt much more complex. In the next section we will show how connectionist networks can build representations that may come closer to the kind that people apparently employ.

1.6 CONNECTIONIST REPRESENTATIONS AND HUMAN JUDGMENTS OF SIMILARITY

In the previous sections we showed how connectionist systems can develop representations that work for performing particular tasks. For a psychological model, however, in addition to seeking representations that just *work*, we would like to constrain the network somehow to develop internal representations that resemble those employed by humans. One way to do this is to impose human similarity structures onto the network's representations, rather than giving it free rein to modify the similarity structures as needed. In particular, if we are interested in reconstructing internal human representations of input stimuli based on human judgments of similarity between those stimuli, then if we construct a network that is constrained to produce the same similarity structure between *its* internal representations of the inputs, the two representational systems—human and network model—should match. Several years ago we carried out a series of experiments designed to test this hypothesis (Todd 1987, 1988; see Todd and Rumelhart 1993 for more details.)

The basic idea is illustrated in figure 1.14. Essentially, we want a network to learn a mapping from its inputs to an internal representation, as usual, but this time we want a natural similarity measure over these representations to mimic human judgments of similarity between the inputs. The network's internal representations will then be a model of those the humans employed to make their similarity judgments. In our previous examples, the networks developed a similarity structure over their internal representations that would work to produce the desired outputs; here, since we want to constrain the network's representations to match the ones humans use in making similarity judgments, we use a little trick: we make the network's desired outputs *be* the human similarity judgments. To do this, we have to construct a network that compares two input stimuli at a time and produces an output corresponding to their similarity.

The network architecture for this purpose is shown in figure 1.15. Here we have used two instances of the encoder network from figure 1.14, whose outputs (the internal representations) are compared by a comparison subnetwork that produces the final similarity judgment as output. Humans probably

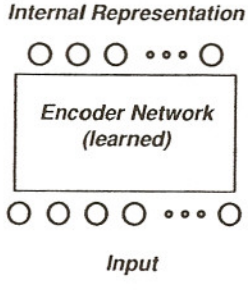

Figure 1.14 A simple encoder network that maps inputs into internal representations, for use in a psychological model of human representations.

have just one copy of something analogous to the encoder subnetwork and use it twice in succession to compare two sequentially presented stimuli; in the network we have converted this use of the encoder twice in time into using two copies of it in parallel simultaneously. The weights in both copies of the encoder network are kept equal throughout training (that is, the two encoders are always constrained to be identical and hence use the same encoding strategy on both input stimuli).

To train this network, we first collect human similarity judgments on pairs of stimuli and encode these stimuli into an appropriate form for use as network inputs with the corresponding similarity value as desired output. Then, for each judgment, one stimulus, S_1, is inserted into the left set of input units, and the other, S_2, is inserted into the right set. Each input stimulus is then mapped through its own copy of the same encoder subnetwork, to produce two corresponding internal representations, X_1 and X_2, at the two-part hidden-unit representation layer. These two representations are then compared using the fixed comparison subnetwork, which embodies some theory of how similarity judgments are made. The final output of this comparison network is the predicted similarity of stimuli S_1 and S_2. This predicted similarity is compared to the actual human similarity judgment, and the difference between them is used as an error signal to modify the weights in (both copies of) the encoder subnetwork. The weights in the comparison subnetwork are held fixed throughout training. In this way, only the mapping from input stimuli to internal representation can change during training; the similarity comparison method itself is assumed to remain constant for humans (at this time scale), and so it is kept constant in the network model as well. We train the encoder networks in this way until the similarity judgments are predicted as accurately as possible. The final results we are interested in are the internal representa-

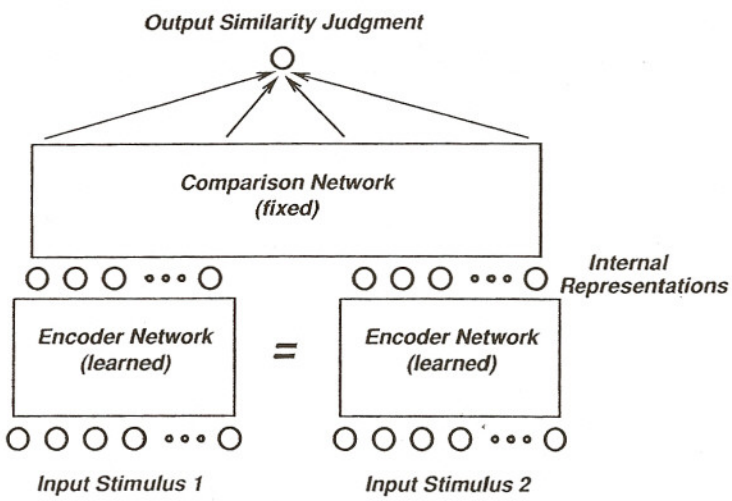

Figure 1.15 Architecture of a connectionist network for mapping from pairs of stimuli to their human-judged similarity, via a learned psychological encoding and a fixed comparison subnetwork.

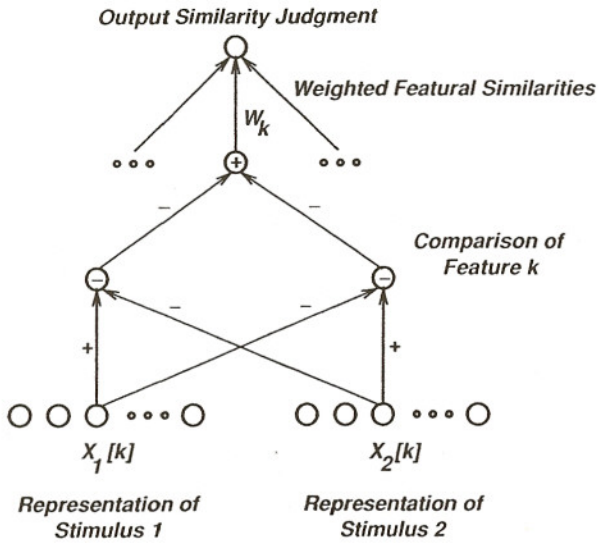

Figure 1.16 The subnetwork structure for comparing two featural dimensions using a Hamming-like similarity metric, showing the signs of the weights and biases (within unit circles) used.

tions developed by the network, which we expect will mimic those used by humans to represent these stimuli.

This procedure for modeling human representations can be carried out under a wide variety of assumptions about the nature of the similarity comparison subnetwork. For example, we can easily implement a simple Euclidian-distance measure of similarity between the representation vectors. In this case our procedure acts very much like standard multidimemional scaling (MDS; see Shepard 1980), finding representations in which individual hidden units capture real-valued featural dimensions of the stimuli. In most of our experiments we have employed something close to Hamming distance as our similarity metric, because it is a rough measure of pattern overlap, particularly suited to the sort of microfeature-vector pattern representations developed in PDP systems.

Figure 1.16 illustrates the basic structure of our comparison subnetwork for computing the Hamming-style similarity measure. The featural dimensions are compared individually; here we see the comparison of feature k alone. If feature k is about equally present in both input S_1 and input S_2, that is, if hidden units $X_1[k]$ and $X_2[k]$ have roughly equal activation values, then the comparison unit at the top of the subnetwork will also have a high activation value and will contribute strongly to the overall similarity judgment based on this feature match. If, however, feature k is present in greatly different amounts in the two input stimuli, then the output for this feature comparison will be small, indicating dissimilarity. The comparison results for each featural dimension are independently weighted and summed to form the final predicted similarity value. The entire similarity-judging feature abstraction network with this comparison subnetwork structure inserted is illustrated in figure 1.17.

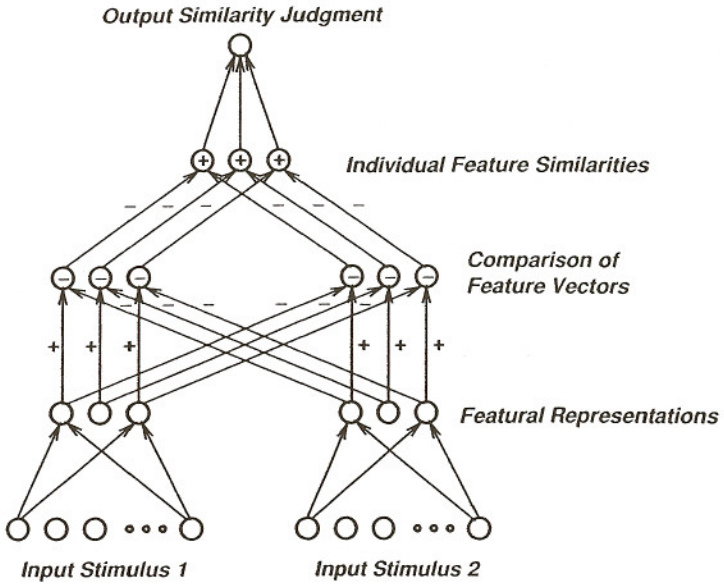

Figure 1.17 The structure of the similarity-judging network, with the Hamming-like comparison subnetwork inserted for three features.

Table 1.1 Similarity Measures between Kinship Terms

	F	B	S	GS	U	N	C
GF	.7	.1	.2	.8	.2	.1	.1
F		.4	.7	.2	.3	.1	.1
B			.7	.2	.2	.2	.2
S				.5	.1	.1	.1
GS					.1	.1	.2
U						.6	.7
N							.8

We have carried out a number of experiments with this paradigm, and in this section we provide the results of two of them to illustrate the basic characteristics of the methodology. In the first experiment we employed data provided by Roy D'Andrade (personal communication, conversation, 1986) on judgments of similarities among kinship terms. The terms employed were *grandfather, father, brother, son, grandson, uncle, cousin,* and *nephew.* The matrix of similarities between these terms is given in table 1.1. We used this data to train our network by constructing pairs of localist representations of the terms as inputs (i.e., eight input units for each stimulus, with one turned on at a time), with their corresponding similarity value as the desired output. So, for example, to train the network on the pair ⟨*grandfather, brother*⟩, we turn on the unit in S_1 corresponding to *grandfather* and the unit in S_2 corresponding to

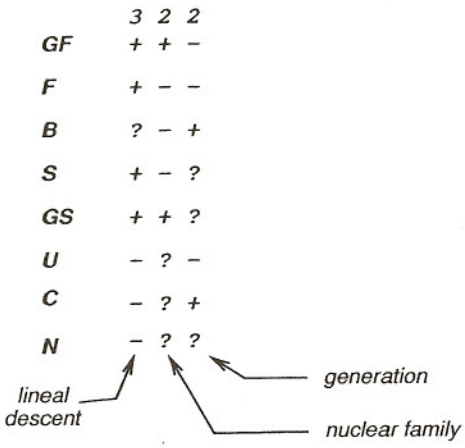

Figure 1.18 Representations developed by a similarity-judging network for kinship terms, showing the three features discovered.

brother and use 0.1 as the desired output value. We used three representation hidden units in this example.

The representations developed by the network are illustrated in figure 1.18. The three features extracted correspond to rather plausible categories and are weighted in ratios of approximately 3 to 2 to 2; that is, the first feature was the most important in terms of determining the similarity, and the second two features were about equally important. The first feature appears to correspond roughly to the dimension of lineal descent: *grandfather, father, son,* and *grandson* are all given the same featural value (+) and all belong to the same line of descent. *Uncle, cousin,* and *nephew* are given a very different value (−), and *brother* is classified somewhere between. The second dimension appears to correspond to distance from the nuclear family. Thus, *father, brother,* and *son* are all classified together and all belong to the same nuclear family. Likewise, *grandfather* and *grandson* are both equally distant from the nuclear family and so are classified together with the same feature value, as are *uncle, cousin,* and *nephew*. The third dimension appears to correspond to generation. In this case all of those older than ego (self) are classified together, namely, *grandfather, father,* and *uncle*. Similarly, those younger—*son, grandson,* and *nephew*—are classified together, as are those of the same generation—*brother* and *cousin*. These featural dimensions discovered by the network are all readily interpretable by a human observer, supporting our hypothesis that the network is picking up the same types of features that people use to represent these concepts. (See Hinton 1986 for an alternative and less psychologically motivated network representation of family relation concepts.) Finally, it is interesting that all three dimensions appear to employ trinary values. Since our comparison subnetwork simply checks to see whether the feature values are significantly different from one another, a trinary system is easily attained.

One of the particularly nice features of the architecture we have employed here is that we need not use localist representations for our input stimuli. For

instance, consider the case in which the input stimuli are geometric forms, like those shown in figure 1.6, presented in pairs with their corresponding similarity value. Now when the network learns how to encode the input stimuli, it is picking up something about the relationship between the input's geometric structure and its internal representation—in this case, that four units alone can capture the structure inherent in the collection of "line segments" presented over the input units. This suggests that if we train the network on a *subset* of the geometric forms in the pair comparisons, we could then present it with new examples and it could "predict" their featural representations and their proper similarity value—even though it has never seen these stimulus patterns before. Furthermore, this generalization to new stimuli should match how humans would generalize. In a final experiment we illustrate these properties.

For this experiment we chose a set of data from a study by Rothkopf (1957; also reported in Shepard 1963), in which subjects indicated the similarity among pairs of Morse codes for individual letters and numerals (through intercode confusions). Since the maximum length of a single Morse code for a letter or numeral is a collection of five dots and dashes, we represented each code with ten input units, one for a dot and one for a dash for each of the five possible positions. A dot in a particular position would turn on the first of the pair of input units, a dash the second, and the absence of both (indicating that we were beyond the end of the code) would mean that neither unit at that position was turned on. Thus the letter *E* (Morse code ".") was converted into the input vector $\langle 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$, and the numeral 0 (Morse code "−−−−−") was converted into $\langle 0, 1, 0, 1, 0, 1, 0, 1, 0, 1 \rangle$. In this way all the letters and numerals could be easily represented.

We ran several simulations in which we trained a network with two featural representation hidden units on a subset of these Morse codes and then tested its generalization ability on the untrained patterns. In one case, we trained the network on pairs using thirty-one out of the thirty-six total letters and numbers, leaving out *I*, *D*, *H*, *V*, and *1* completely from the training set. The resulting representations of all thirty-six codes are illustrated in figure 1.19, where we've graphed the activation values of the two feature units on the x and y axes. The interpretation of the two featural dimensions is readily seen. The x-axis dimension corresponds to the length of the code for the letter or number involved. On the far left of the figure we see the short codes, and on the far right the long codes. The y-axis feature roughly corresponds to the proportion of dots or dashes in the code. Near the top we see the codes for *E*, *S*, *H*, and *5*, which consist entirely of dots, and at the bottom we see *0*, *O*, and *M*, which consist entirely of dashes. Intermediate values along both dimensions correspond to the medium-length, mixed dot-and-dash codes.

These are the same features that Shepard found with his MDS analysis (Shepard 1963), but in this case they were more easily interpreted, as follows. Often, the spatial configurations of the stimuli that MDS discovers must be rotated before the dimensions can be interpreted properly. But because of anisotropies in our network's stimulus-to-representation mapping (from the

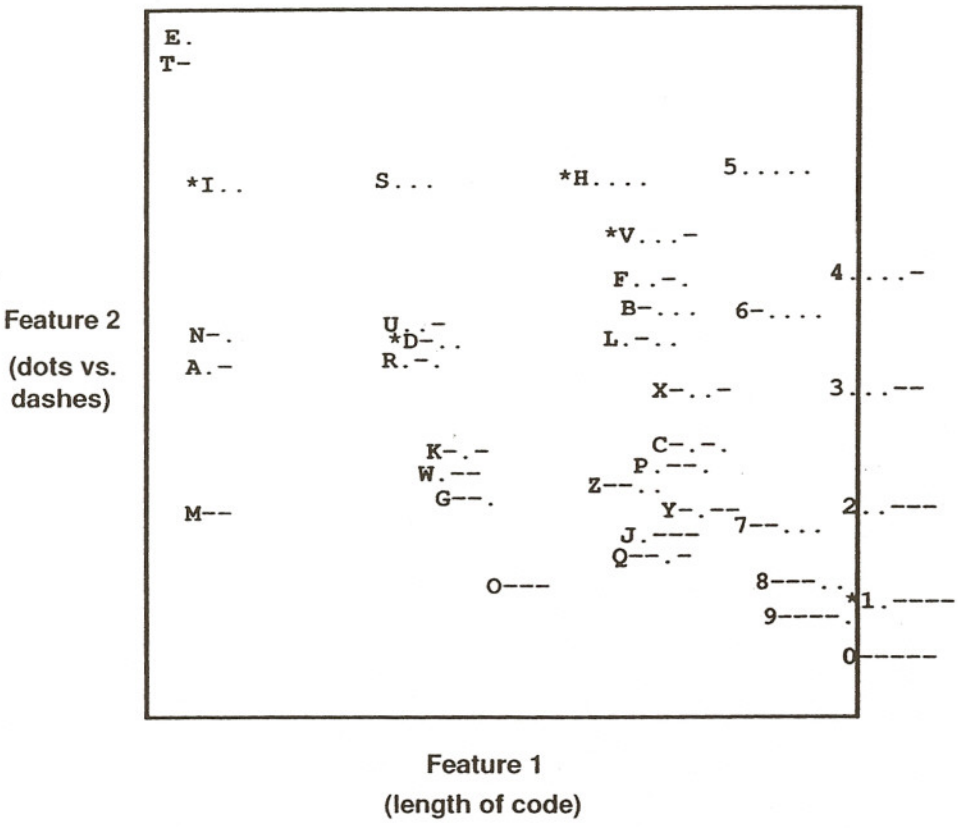

Figure 1.19 Representations developed by a similarity-judging network for Morse code signals, with the two features plotted on the two axes and showing the locations generalized to for five previously unseen codes (starred).

logistic activation functions used), the features came out directly from the two hidden units without the need for rotation. As in the previous example, the features have psychological meanings that we can ascribe to the human representations of these stimuli. Furthermore, the network in this case can generalize to new stimuli; the codes that the network never saw during training are indicated by stars in the positions they are encoded to by the learned representational mapping. Obviously these codes have taken their appropriate place in the representational format. The network has been able to abstract the basic features underlying the human similarity judgments of Morse code signals, and can generalize from those features to predict the similarities among stimuli it has never before seen—something that is quite impossible with standard MDS.

1.7 CONCLUSION

In this chapter we have shown the important role that learning and similarity play in the construction of representations in connectionist networks. Simple

learning procedures such as the backpropagation algorithm can discover sequences of transformations that create the different internal distributed representations needed to perform a given input-output mapping task. Along the way, these transformations must alter the physically based similarity relations among the input vectors into the psychological and functional similarities required at the output. The representations learned as a result at the hidden layers will embody similarity relations that allow proper generalization to and transfer of learning between new stimuli. This similarity-based ability to learn and modify internal representations is an essential feature of any complex information processing system. The examples we described of autoencoders, connectionist semantic networks, and psychological similarity judgment models serve to illustrate the power and variety of representation learning approaches all captured under the unified paradigm of brain-style computation.

NOTE

1. The idea of using the autoencoder to re-represent input patterns was first suggested to me in 1985 by Geoffrey Hinton. It has since become a popular method.

REFERENCES

Brachman, R. J., and Levesque, H. J. (Eds.). (1985). *Readings in knowledge representation.* Los Altos, CA: Morgan Kaufmann.

Collins, A. M., and Quillian, M. R. (1970). Facilitating retrieval from semantic memory: The effect of repeating part of an inference. In A. F. Sanders (Ed.), *Attention and performance III.* Amsterdam: North Holland.

Feldman, J. A., and Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science, 6,* 205–254.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* 1–12. Hillsdale, NJ: Lawrence Erlbaum Associates.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press/Bradford Books.

Lippmann, R. P., Moody, J. E., and Touretzky, D. S. (Eds.). (1991). *Advances in neural information processing 3.* San Mateo, CA: Morgan Kaufmann.

McClelland, J. L., and Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: Part 1. An account of basic findings. *Psychological Review, 88,* 375–407.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Quillian, M. R. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic information processing.* Cambridge, MA: MIT Press.

Rosenblatt, F. (1962). *Principles of neurodynamics.* New York: Spartan.

Rothkopf, E. Z. (1957). A measure of stimulus similarity and errors in some paired-associate learning tasks. *Journal of Experimental Psychology, 53,* 94–101.

Rumelhart, D. E. (1990). Brain style computation: Learning and generalization. In S. F. Zornetzer, J. L. Davis, and C. Lau (Eds.), *An introduction to neural and electronic networks.* San Diego: Academic Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press/ Bradford Books.

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

Rumelhart, D. E., and Norman, D. A. (1988). Representation in memory. In R. C. Atkinson, R. J. Herrnstein, G. Lindzey, and R. D. Luce (Eds.), *Stevens' handbook of experimental psychology.* New York: Wiley. Extracts of chapter reprinted as "Representation of Knowledge" in A. M. Aitkenhead and J. M. Slack (Eds.), (1985), *Issues in cognitive modeling.* London: Lawrence Erlbaum Associates.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3,* 210–229.

Selfridge, O. (1959). Pandemonium: A paradigm for learning. In *Symposium on the mechanization of thought processes.* London: HM Stationery Office.

Shastri, L. (1988a). *Semantic networks: An evidential formalization and its connectionist realization.* Los Altos, CA: Morgan Kaufmann/London: Pitman.

Shastri, L. (1988b). A connectionist approach to knowledge representation and limited inference. *Cognitive Science, 12,* 331–392.

Shepard, R. N. (1963). Analysis of proximities as a technique for the study of information processing in man. *Human Factors, 5,* 33–48.

Shepard, R. N. (1980). Multidimensional scaling, tree-fitting, and clustering. *Science, 210,* 390–398.

Todd, P. M. (1987). *Abstracting musical features using a parallel distributed processing approach.* Master's thesis, UC San Diego Institute for Cognitive Science, La Jolla, CA.

Todd, P. M. (1988). *A connectionist multidimensional scaling method allowing semantic generalization.* Unpublished manuscript, Stanford University Psychology Department, Stanford, CA.

Todd, P. M., and Rumelhart, D. E. (1993). Feature abstraction from similarity ratings: A connectionist approach. In Y. Chauvin and D. E. Rumelhart (Eds.), *Backpropagation: Theory, architectures, and applications.* Hillsdale, NJ: Lawrence Erlbaum Associates.