CHAPTER 3

# Distributed Representations

G. E. HINTON, J. L. McCLELLAND, and D. E. RUMELHART

Given a network of simple computing elements and some entities to be represented, the most straightforward scheme is to use one computing element for each entity. This is called a *local* representation. It is easy to understand and easy to implement because the structure of the physical network mirrors the structure of the knowledge it contains. The naturalness and simplicity of this relationship between the knowledge and the hardware that implements it have led many people to simply assume that local representations are the best way to use parallel hardware. There are, of course, a wide variety of more complicated implementations in which there is no one-to-one correspondence between concepts and hardware units, but these implementations are only worth considering if they lead to increased efficiency or to interesting emergent properties that cannot be conveniently achieved using local representations.

This chapter describes one type of representation that is less familiar and harder to think about than local representations. Each entity is represented by a pattern of activity distributed over many computing elements, and each computing element is involved in representing many different entities. The strength of this more complicated kind of representation does not lie in its notational convenience or its ease of implementation in a conventional computer, but rather in the efficiency with which it makes use of the processing abilities of networks of simple, neuron-like computing elements.

Every representational scheme has its good and bad points. Distributed representations are no exception. Some desirable properties arise very naturally from the use of patterns of activity as representations. Other properties, like the ability to temporarily store a large set of arbitrary associations, are much harder to achieve. As we shall see, the best psychological evidence for distributed representations is the degree to which their strengths and weaknesses match those of the human mind.

The first section of this chapter stresses some of the virtues of distributed representations. The second section considers the efficiency of distributed representations, and shows clearly why distributed representations can be better than local ones for certain classes of problems. A final section discusses some difficult issues which are often avoided by advocates of distributed representations, such as the representation of constituent structure and the sequential focusing of processing effort on different aspects of a structured object.

*Disclaimers.* Before examining the detailed arguments in favor of distributed representations, it is important to be clear about their status within an overall theory of human information processing. It would be wrong to view distributed representations as an *alternative* to representational schemes like semantic networks or production systems that have been found useful in cognitive psychology and artificial intelligence. It is more fruitful to view them as one way of implementing these more abstract schemes in parallel networks, but with one proviso: Distributed representations give rise to some powerful and unexpected emergent properties. These properties can therefore be taken as primitives when working in a more abstract formalism. For example, distributed representations are good for content-addressable memory, automatic generalization, and the selection of the rule that best fits the current situation. So if one assumes that more abstract models are implemented in the brain using distributed representations, it is not unreasonable to treat abilities like content-addressable memory, automatic generalization, or the selection of an appropriate rule as primitive operations, even though there is no easy way to implement these operations in conventional computers. Some of the emergent properties of distributed representations are not easily captured in higher-level formalisms. For example, distributed representations are consistent with the simultaneous application of a large number of partially fitting rules to the current situation, each rule being applied to the degree that it is relevant. We shall examine these properties of distributed representations in the chapter on schemata (Chapter 14). There we will see clearly that schemata and other higher-level constructs provide only approximate characterizations of mechanisms which rely on distributed

representations. Thus, the contribution that an analysis of distributed representations can make to these higher-level formalisms is to legitimize certain powerful, primitive operations which would otherwise appear to be an appeal to magic; to enrich our repertoire of primitive operations beyond those which can conveniently be captured in many higher-level formalisms; and to suggest that these higher-level formalisms may only capture the coarse features of the computational capabilities of the underlying processing mechanisms.

Another common source of confusion is the idea that distributed representations are somehow in conflict with the extensive evidence for localization of function in the brain (Luria, 1973). A system that uses distributed representations still requires many different modules for representing completely different kinds of thing at the same time. The distributed representations occur *within* these localized modules. For example, different modules would be devoted to things as different as mental images and sentence structures, but two different mental images would correspond to *alternative* patterns of activity in the same module. The representations advocated here are local at a global scale but global at a local scale.

## VIRTUES OF DISTRIBUTED REPRESENTATIONS

This section considers three important features of distributed representations: (a) their essentially constructive character; (b) their ability to generalize automatically to novel situations; and (c) their tunability to changing environments. Several of these virtues are shared by certain local models, such as the interactive activation model of word perception, or McClelland's (1981) model of generalization and retrieval described in Chapter 1.

### Memory as Inference

People have a very flexible way of accessing their memories: They can recall items from partial descriptions of their contents (Norman & Bobrow, 1979). Moreover, they can do this even if some parts of the partial description are wrong. Many people, for example, can rapidly retrieve the item that satisfies the following partial description: It is an actor, it is intelligent, it is a politician. This kind of *content-addressable* memory is very useful and it is very hard to implement on a conventional computer because computers store each item at a particular

address, and to retrieve an item they must know its address. If all the combinations of descriptors that will be used for access are free of errors and are known in advance, it is possible to use a method called *hash coding* that quickly yields the address of an item when given part of its content. In general, however, content-addressable memory requires a massive search for the item that best fits the partial description. The central computational problem in memory is how to make this search efficient. When the cues can contain errors, this is very difficult because the failure to fit one of the cues cannot be used as a filter for quickly eliminating inappropriate answers.

Distributed representations provide an efficient way of using parallel hardware to implement best-fit searches. The basic idea is fairly simple, though it is quite unlike a conventional computer memory. Different items correspond to different patterns of activity over the very same group of hardware units. A partial description is presented in the form of a partial activity pattern, activating some of the hardware units.[1] Interactions between the units then allow the set of active units to influence others of the units, thereby completing the pattern, and generating the item that best fits the description. A new item is "stored" by modifying the interactions between the hardware units so as to create a new stable pattern of activity. The main difference from a conventional computer memory is that patterns which are not active do not exist anywhere. They can be re-created because the connection strengths between units have been changed appropriately, but each connection strength is involved in storing many patterns, so it is impossible to point to a particular place where the memory for a particular item is stored.

Many people are surprised when they understand that the connections between a set of simple processing units are capable of supporting a large number of different patterns. Illustrations of this aspect of distributed models are provided in a number of papers in the literature (e.g., Anderson, 1977; Hinton, 1981a); this property is illustrated in the model of memory and amnesia described in Chapters 17 and 25.

One way of thinking about distributed memories is in terms of a very large set of plausible inference rules. Each active unit represents a "microfeature" of an item, and the connection strengths stand for plausible "microinferences" between microfeatures. Any particular pattern

[1] This is easy if the partial description is simply a set of features, but it is much more difficult if the partial description mentions relationships to other objects. If, for example, the system is asked to retrieve John's father, it must represent John, but if John and his father are represented by mutually exclusive patterns of activity in the very same group of units, it is hard to see how this can be done without preventing the representation of John's father. A distributed solution to this problem is described in the text.

of activity of the units will satisfy some of the microinferences and violate others. A stable pattern of activity is one that violates the plausible microinferences less than any of the neighboring patterns. A new stable pattern can be created by changing the inference rules so that the new pattern violates them less than its neighbors. This view of memory makes it clear that there is no sharp distinction between genuine memory and plausible reconstruction. A genuine memory is a pattern that is stable because the inference rules were modified when it occurred before. A "confabulation" is a pattern that is stable because of the way the inference rules have been modified to store several different previous patterns. So far as the subject is concerned, this may be indistinguishable from the real thing.

The blurring of the distinction between veridical recall and confabulation or plausible reconstruction seems to be characteristic of human memory (Bartlett, 1932; Neisser, 1981). The reconstructive nature of human memory is surprising only because it conflicts with the standard metaphors we use. We tend to think that a memory system should work by storing literal copies of items and then retrieving the stored copy, as in a filing cabinet or a typical computer database. Such systems are not naturally reconstructive.

If we view memory as a process that constructs a pattern of activity which represents the most plausible item that is consistent with the given cues, we need some guarantee that it will converge on the representation of the item that best fits the description, though it might be tolerable to sometimes get a good but not optimal fit. It is easy to *imagine* this happening, but it is harder to make it actually work. One recent approach to this problem is to use statistical mechanics to analyze the behavior of groups of interacting *stochastic* units. The analysis guarantees that the better an item fits the description, the more likely it is to be produced as the solution. This approach is described in Chapter 7, and a related approach is described in Chapter 6. An alternative approach, using units with continuous activations (Hopfield, 1984) is described in Chapter 14.

## Similarity and Generalization

When a new item is stored, the modifications in the connection strengths must not wipe out existing items. This can be achieved by modifying a very large number of weights very slightly. If the modifications are all in the direction that helps the pattern that is being stored, there will be a conspiracy effect: The total help for the intended pattern will be the sum of all the small separate modifications.

For unrelated patterns, however, there will be very little transfer of effect because some of the modifications will help and some will hinder. Instead of all the small modifications conspiring together, they will mainly cancel out. This kind of statistical reasoning underpins most distributed memory models, but there are many variations of the basic idea (See Hinton & Anderson, 1981, for several examples).

It is possible to prevent interference altogether by using orthogonal patterns of activity for the various items to be stored (a rudimentary example of such a case is given in Chapter 1). However, this eliminates one of the most interesting properties of distributed representations: They automatically give rise to generalizations. If the task is simply to remember accurately a set of unrelated items, the generalization effects are harmful and are called interference. But generalization is normally a helpful phenomenon. It allows us to deal effectively with situations that are similar but not identical to previously experienced situations.

People are good at generalizing newly acquired knowledge. If you learn a new fact about an object, your expectations about other similar objects tend to change. If, for example, you learn that chimpanzees like onions you will probably raise your estimate of the probability that gorillas like onions. In a network that uses distributed representations, this kind of generalization is automatic. The new knowledge about chimpanzees is incorporated by modifying some of the connection strengths so as to alter the causal effects of the distributed pattern of activity that represents chimpanzees.[2] The modifications automatically change the causal effects of all similar activity patterns. So if the representation of gorillas is a similar activity pattern over the same set of units, its causal effects will be changed in a similar way.

The very simplest distributed scheme would represent the concept of onion and the concept of chimpanzee by *alternative* activity patterns over the very same set of units. It would then be hard to represent chimps and onions at the same time. This problem can be solved by using separate modules for each possible role of an item within a larger structure. Chimps, for example, are the "agent" of the liking and so a pattern representing chimps occupies the "agent" module and the pattern representing onions occupies the "patient" module (see Figure 1).

---

[2] The internal structure of this pattern may also change. There is always a choice between changing the weights on the outgoing connections and changing the pattern itself so that different outgoing connections become relevant. Changes in the pattern itself alter its similarity to other patterns and thereby alter how generalization will occur in the future. It is generally much harder to figure out how to change the pattern that represents an item than it is to figure out how to change the outgoing connections so that a particular pattern will have the desired effects on another part of the network.

Each module can have alternative patterns for all the various items, so this scheme does not involve local representations of items. What is localized is the role.

If you subsequently learn that gibbons and orangutans do not like onions your estimate of the probability that gorillas like onions will fall, though it may still remain higher than it was initially. Obviously, the combination of facts suggests that liking onions is a peculiar quirk of chimpanzees. A system that uses distributed representations will automatically arrive at this conclusion, provided that the alternative patterns that represent the various apes are related to one another in a particular way that is somewhat more specific than just being similar to one another: There needs to be a part of each complete pattern that is identical for all the various apes. In other words, the group of units used for the distributed representations must be divided into two

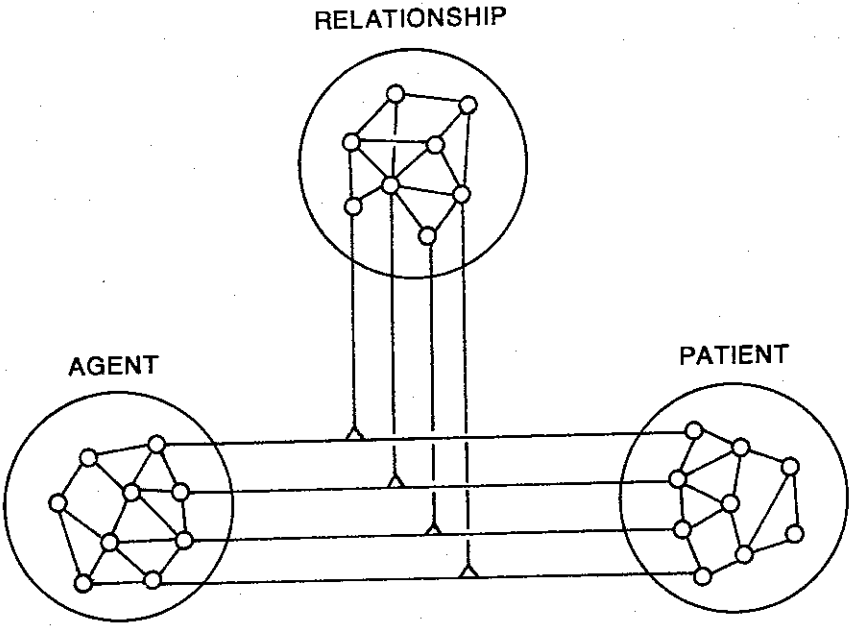


FIGURE 1. In this simplified scheme there are two different modules, one of which represents the agent and the other the patient. To incorporate the fact that chimpanzees like onions, the pattern for chimpanzees in one module must be associated with the pattern for onions in the other module. Relationships other than "liking" can be implemented by having a third group of units whose pattern of activity represents the relationship. This pattern must then "gate" the interactions between the agent and patient groups. Hinton (1981a) describes one way of doing this gating by using a fourth group of units.

subgroups, and all the various apes must be represented by the same pattern in the first subgroup, but by different patterns in the second subgroup. The pattern of activity over the first subgroup represents the *type* of the item, and the pattern over the second subgroup represents additional microfeatures that discriminate each instance of the type from the other instances. Note that any subset of the microfeatures can be considered to define a type. One subset might be common to all apes, and a different (but overlapping) subset might be common to all pets. This allows an item to be an instance of many different types simultaneously.

When the system learns a new fact about chimpanzees, it usually has no way of knowing whether the fact is true of all apes or is just a property of chimpanzees. The obvious strategy is therefore to modify the strengths of the connections emanating from *all* the active units, so that the new knowledge will be partly a property of apes in general and partly a property of whatever features distinguish chimps from other apes. If it is subsequently learned that other apes do not like onions, correcting modifications will be made so that the information about onions is no longer associated with the subpattern that is common to all apes. The knowledge about onions will then be restricted to the subpattern that distinguishes chimps from other apes. If it had turned out that gibbons and orangutans also liked onions, the modifications in the weights emanating from the subpattern representing apes would have reinforced one another, and the knowledge would have become associated with the subpattern shared by all apes rather than with the patterns that distinguish one ape from another.

A very simple version of this theory of generalization has been implemented in a computer simulation (Hinton, 1981a). Several applications that make use of this property can be found in Part IV of this book.

There is an obvious generalization of the idea that the representation of an item is composed of two parts, one that represents the type and another that represents the way in which this particular instance differs from others of the same type. Almost all types are themselves instances of more general types, and this can be implemented by dividing the pattern that represents the type into two subpatterns, one for the more general type of which this type is an instance, and the other for the features that discriminate this particular type from others instances of the same general type. Thus the relation between a type and an instance can be implemented by the relationship between a set of units and a larger set that includes it. Notice that the more general the type, the *smaller* the set of units used to encode it. As the number of terms in an *intensional* description gets smaller, the corresponding *extensional* set gets larger.

In traditional semantic networks that use local representations, generalization is not a direct consequence of the representation. Given that chimpanzees like onions, the obvious way of incorporating the new knowledge is by changing the strengths of connections belonging to the chimpanzee unit. But this does not automatically change connections that belong to the gorilla unit. So extra processes must be invoked to implement generalization in a localist scheme. One commonly used method is to allow activation to spread from a local unit to other units that represent similar concepts (Collins & Loftus, 1975; Quillian, 1968). Then when one concept unit is activated, it will partially activate its neighbors and so any knowledge stored in the connections emanating from these neighbors will be partially effective. There are many variations of this basic idea (Fahlman, 1979; Levin, 1976; McClelland, 1981).

It is hard to make a clean distinction between systems that use local representations plus spreading activation and systems that use distributed representations. In both cases the result of activating a concept is that many different hardware units are active. The distinction almost completely disappears in some models such as McClelland's (1981) generalization model, where the properties of a concept are represented by a pattern of activation over feature units and where this pattern of activation is determined by the interactions of a potentially very large number of units for instances of the concept. The main difference is that in one case there is a particular individual hardware unit that acts as a "handle" which makes it easy to attach purely conventional properties like the name of the concept and easier for the theorist who constructed the network to know what the individual parts of the network stand for.

If we construct our networks by hand-specifying the connections between the units in the network, a local representation scheme has some apparent advantages. First, it is easier to think one understands the behavior of a network if one has put in all the "knowledge"—all the connections—oneself. But if it is the entire, distributed pattern of interacting influences among the units in the network that is doing the work, this understanding can often be illusory. Second, it seems intuitively obvious that it is harder to attach an arbitrary name to a distributed pattern than it is to attach it to a single unit. What is intuitively harder, however, may not be more efficient. We will see that one can actually implement aribitrary associations with fewer units using distributed representations. Before we turn to such considerations, however, we examine a different advantage of distributed representations: They make it possible to create new concepts without allocating new hardware.

## Creating New Concepts

Any plausible scheme for representing knowledge must be capable of learning novel concepts that could not be anticipated at the time the network was initially wired up. A scheme that uses local representations must first make a discrete decision about *when* to form a new concept, and then it must find a spare hardware unit that has suitable connections for implementing the concept involved. Finding such a unit may be difficult if we assume that, after a period of early development, new knowledge is incorporated by changing the strengths of the existing connections rather than by growing new ones. If each unit only has connections to a small fraction of the others, there will probably not be any units that are connected to just the right other ones to implement a new concept. For example, in a collection of a million units each connected at random to ten thousand others, the chance of there being *any* unit that is connected to a particular set of 6 others is only one in a million.

In an attempt to rescue local representations from this problem, several clever schemes have been proposed that use two classes of units. The units that correspond to concepts are not directly connected to one another. Instead, the connections are implemented by indirect pathways through several layers of intermediate units (Fahlman, 1980; Feldman, 1982). This scheme works because the number of *potential* pathways through the intermediate layers far exceeds the total number of physical connections. If there are $k$ layers of units, each of which has a fan-out of $n$ connections to randomly selected units in the following layer, there are $n^k$ potential pathways. There is almost certain to be a pathway connecting any two concept-units, and so the intermediate units along this pathway can be dedicated to connecting those two concept-units. However, these schemes end up having to dedicate several intermediate units to each effective connection, and once the dedication has occurred, all but one of the actual connections emanating from each intermediate unit are wasted. The use of several intermediate units to create a single effective connection may be appropriate in switching networks containing elements that have units with relatively small fan-out, but it seems to be an inefficient way of using the hardware of the brain.

The problems of finding a unit to stand for a new concept and wiring it up appropriately do not arise if we use distributed representations. All we need to do is modify the interactions between units so as to create a new stable pattern of activity. If this is done by modifying a large number of connections very slightly, the creation of a new pattern need not disrupt the existing representations. The difficult problem is

to choose an appropriate pattern for the new concept. The effects of the new representation on representations in other parts of the system will be determined by the units that are active, and so it is important to use a collection of active units that have roughly the correct effects. Fine-tuning of the effects of the new pattern can be achieved by slightly altering the effects of the active units it contains, but it would be unwise to choose a *random* pattern for a new concept because major changes would then be needed in the weights, and this would disrupt other knowledge. Ideally, the distributed representation that is chosen for a new concept should be the one that requires the least modification of weights to make the new pattern stable and to make it have the required effects on other representations.

Naturally, it is not necessary to create a new stable pattern all in one step. It is possible for the pattern to emerge as a result of modifications on many separate occasions. This alleviates an awkward problem that arises with local representations: The system must make a discrete all-or-none decision about when to create a new concept. If we view concepts as stable patterns, they are much less discrete in character. It is possible, for example, to differentiate one stable pattern into two closely related but different variants by modifying some of the weights slightly. Unless we are allowed to clone the hardware units (and all their connections), this kind of gradual, conceptual differentiation is much harder to achieve with local representations.

One of the central problems in the development of the theory of distributed representation is the problem of specifying the exact procedures by which distributed representations are to be learned. All such procedures involve connection strength modulation, following "learning rules" of the type outlined in Chapter 2. Not all the problems have been solved, but significant progress is being made on these problems. (See the chapters in Part II.)

## DISTRIBUTED REPRESENTATIONS THAT WORK EFFICIENTLY

In this section, we consider some of the technical details about the implementation of distributed representations. First, we point out that certain distributed representation schemes can fail to provide a sufficient basis for differentiating different concepts, and we point out what is required to avoid this limitation. Then, we describe a way of using distributed representations to get the most information possible out of a simple network of connected units. The central result is a surprising one: If you want to encode features accurately using as few units as

possible, it pays to use units that are very coarsely tuned, so that each feature activates many different units and each unit is activated by many different features. A specific feature is then encoded by a pattern of activity in many units rather than by a single active unit, so coarse coding is a form of distributed representation.

To keep the analysis simple, we shall assume that the units have only two values, on and off.[3] We shall also ignore the dynamics of the system because the question of interest, for the time being, is how many units it takes to encode features with a given accuracy. We start by considering the kind of feature that can be completely specified by giving a type (e.g., line-segment, corner, dot) and the values of some continuous parameters that distinguish it from other features of the same type (e.g., position, orientation, size.) For each type of feature there is a space of possible instances. Each continuous parameter defines a dimension of the feature space, and each particular feature corresponds to a point in the space. For features like dots in a plane, the space of possible features is two-dimensional. For features like stopped, oriented edge-segments in three-dimensional space, the feature space is six-dimensional. We shall start by considering two-dimensional feature spaces and then generalize to higher dimensionalities.

Suppose that we wish to represent the position of a single dot in a plane, and we wish to achieve high accuracy without using too many units. We define the accuracy of an encoding scheme to be the number of different encodings that are generated as the dot is moved a standard distance through the space. One encoding scheme would be to divide the units into an X group and a Y group, and dedicate each unit to encoding a particular $X$ or $Y$ interval as shown in Figure 2. A given dot would then be encoded by activity in two units, one from each group, and the accuracy would be proportional to the number of units used. Unfortunately, there are two problems with this. First, if two dots have to be encoded at the same time, the method breaks down. The two dots will activate two units in each group, and there will be no way of telling, from the active units, whether the dots were at $(x1, y1)$ and $(x2, y2)$ or at $(x1, y2)$ and $(x2, y1)$. This is called the *binding problem.* It arises because the representation does not specify what goes with what.

[3] Similar arguments apply with multivalued activity levels, but it is important not to allow activity levels to have arbitrary precision because this makes it possible to represent an infinite amount of information in a single activity level. Units that transmit a discrete impulse with a probability that varies as a function of their activation seem to approximate the kind of precision that is possible in neural circuitry (see Chapters 20 and 21).

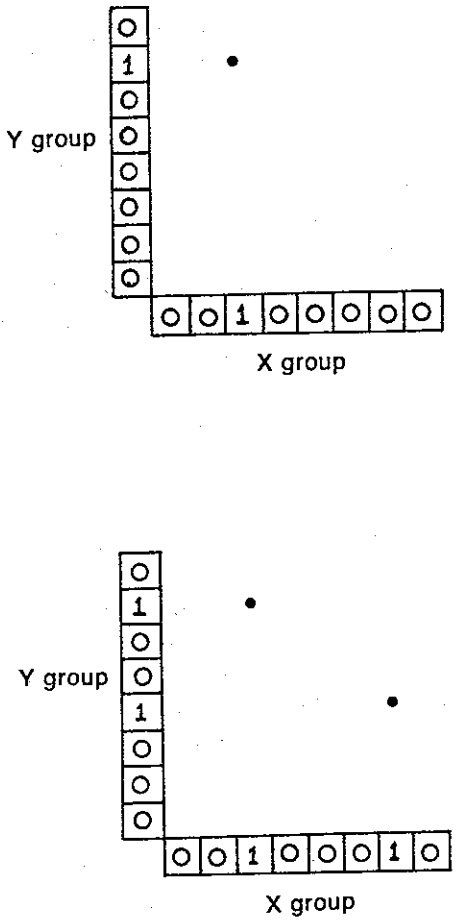


FIGURE 2. *A*: A simple way of using two groups of binary units to encode the position of a point in a two-dimensional space. The active units in the X and Y groups represent the x- and y-coordinates. *B*: When two points must be encoded at the same time, it is impossible to tell which x-coordinate goes with which y-coordinate.

The second problem arises even if we allow only one point to be represented at a time. Suppose we want certain representations to be associated with an overt response, but not others: We want $(x1, y1)$ and $(x2, y2)$ to be associated with a response, but not $(x1, y2)$ or $(x2, y1)$. We cannot implement this association using standard weighted connections to response units from units standing for the values on the two dimensions separately. For the unit for $x1$ and the unit for $x2$ would both have to activate the response, and the unit for

$y1$ and the unit for $y2$ would both have to activate the response. There would be no way of preventing the response from being activated when the unit for $x1$ and the unit for $y2$ were both activated. This is another aspect of the binding problem since, again, the representation fails to specify what must go with what.

In a conventional computer it is easy to solve the binding problem. We simply create two records in the computer memory. Each record contains a pair of coordinates that go together as coordinates of one dot, and the binding information is encoded by the fact that the two coordinate values are sitting in the same record (which usually means they are sitting in neighboring memory locations). In parallel networks it is much harder to solve the binding problem.

## Conjunctive Encoding

One approach is to set aside, in advance, one unit for each possible *combination* of $X$ and $Y$ values. This amounts to covering the plane with a large number of small, nonoverlapping zones and dedicating a unit to each zone. A dot is then represented by activity in a single unit so this is a *local* representation. The use of one unit for each discriminable feature solves the binding problem by having units which stand for the conjunction of values on each of two dimensions. In general, to permit an arbitrary association between particular combinations of features and some output or other pattern of activation, some conjunctive representation may be required.

However, this kind of local encoding is very expensive. It is much less efficient than the previous scheme because the accuracy of pinpointing a point in the plane is only proportional to the square root of the number of units. In general, for a $k$-dimensional feature space, the local encoding yields an accuracy proportional to the $k^{\text{th}}$ root of the number of units. Achieving high accuracy without running into the binding problem is thus very expensive.

The use of one unit for each discriminable feature may be a reasonable encoding if a very large number of features are presented on each occasion, so that a large fraction of the units are active. However, it is a very inefficient encoding if only a very small fraction of the possible features are presented at once. The average amount of information conveyed by the state of a binary unit is 1 bit if the unit is active half the time, and it is much less if the unit is only rarely active.[4] It would

[4] The amount of information conveyed by a unit that has a probability of p of being on is $-p\log p - (1-p)\log(1-p)$.

therefore be more efficient to use an encoding in which a larger fraction of the units were active at any moment. This can be done if we abandon the idea that each discriminable feature is represented by activity in a single unit.

## Coarse Coding

Suppose we divide the space into larger, overlapping zones and assign a unit to each zone. For simplicity, we will assume that the zones are circular, that their centers have a uniform random distribution throughout the space, and that all the zones used by a given encoding scheme have the same radius. The question of interest is how accurately a feature is encoded as a function of the radius of the zones. If we have a given number of units at our disposal is it better to use large zones so that each feature point falls in many zones, or is it better to use small zones so that each feature is represented by activity in fewer but more finely tuned units?

The accuracy is proportional to the number of different encodings that are generated as we move a feature point along a straight line from one side of the space to the other. Every time the line crosses the boundary of a zone, the encoding of the feature point changes because the activity of the unit corresponding to that zone changes. So the number of discriminable features along the line is just twice the number of zones that the line penetrates.[5] The line penetrates every zone whose center lies within one radius of the line (see Figure 3). This number is proportional to the radius of the zones, $r$, and it is also proportional to their number, $n$. Hence the accuracy, $a$, is related to the number of zones and to their radius as follows:

$$a \propto nr.$$

In general, for a $k$-dimensional space, the number of zones whose centers lie within one radius of a line through the space is proportional to the volume of a $k$-dimensional hypercylinder of radius $r$. This volume is equal to the length of the cylinder (which is fixed) times its $(k-1)$-dimensional cross-sectional area which is proportional to $r^{k-1}$.

---

5 Problems arise if you enter and leave a zone without crossing other zone borders in between because you revert to the same encoding as before, but this effect is negligible if the zones are dense enough for there to be many zones containing each point in the space.

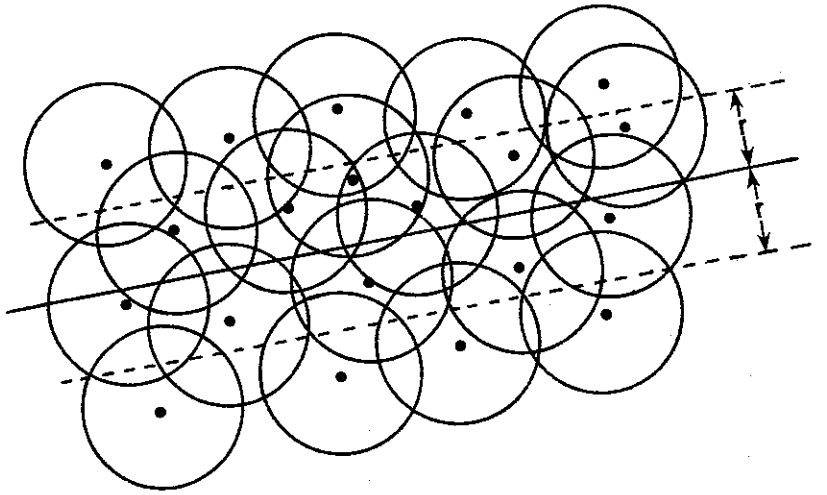

FIGURE 3. The number of zone boundaries that are cut by the line is proportional to the number of zone centers within one-zone radius of the line.

Hence, the accuracy is given by

$$a \propto nr^{k-1}.$$

So, for example, doubling the radius of the zones increases by a factor of 32, the *linear* accuracy with which a six-dimensional feature like a stopped oriented three-dimensional edge is represented. The intuitive idea that larger zones lead to sloppier representations is entirely wrong because distributed representations hold information much more efficiently than local ones. Even though each active unit is less specific in its meaning, the combination of active units is far more specific. Notice also that with coarse coding the accuracy is proportional to the number of units, which is much better than being proportional to the $k$th root of the number.

Units that respond to complex features in retinotopic maps in visual cortex often have fairly large receptive fields. This is often interpreted as the first step on the way to a translation invariant representation. However, it may be that the function of the large fields is not to achieve translation invariance but to pinpoint accurately where the feature is!

*Limitations on coarse coding.* So far, only the advantages of coarse coding have been mentioned, and its problematic aspects have been ignored. There are a number of limitations that cause the coarse coding strategy to break down when the "receptive fields" become too

large. One obvious limitation occurs when the fields become comparable in size to the whole space. This limitation is generally of little interest because other, more severe, problems arise before the receptive fields become this large.

Coarse coding is only effective when the features that must be represented are relatively sparse. If many feature points are crowded together, each receptive field will contain many features and the activity pattern in the coarse-coded units will not discriminate between many alternative combinations of feature points. (If the units are allowed to have integer activity levels that reflect the number of feature points falling within their fields, a few nearby points can be tolerated, but not many.) Thus there is a resolution/accuracy trade-off. Coarse coding can give high accuracy for the parameters of features provided that features are widely spaced so that high resolution is not also required. As a rough rule of thumb, the diameter of the receptive fields should be of the same order as the spacing between simultaneously present feature points.[6]

The fact that coarse coding only works if the features are sparse should be unsurprising given that its advantage over a local encoding is that it uses the information capacity of the units more efficiently by making each unit active more often. If the features are so dense that the units would be active for about half the time using a local encoding, coarse coding can only make things worse.

A second major limitation on the use of coarse coding stems from the fact that the representation of a feature must be used to affect other representations. There is no point using coarse coding if the features have to be recoded as activity in finely tuned units before they can have the appropriate effects on other representations. If we assume that the effect of a distributed representation is the *sum* of the effects of the individual active units that constitute the representation, there is a strong limitation on the circumstances under which coarse coding can be used effectively. Nearby features will be encoded by similar sets of active units, and so they will inevitably tend to have similar effects. Broadly speaking, coarse coding is only useful if the required effect of a feature is the average of the required effects of its neighbors. At a fine enough scale this is nearly always true for spatial tasks. The scale at which it breaks down determines an upper limit on the size of the receptive fields.

6 It is interesting that many of the geometric visual illusions illustrate interactions between features at a distance much greater than the uncertainty in the subjects' knowledge of the position of a feature. This is just what would be expected if coarse coding is being used to represent complex features accurately.

Another limitation is that whenever coarse-coded representations interact, there is a tendency for the coarseness to increase. To counteract this tendency, it is probably necessary to have lateral inhibition operating within each representation. This issue requires further research.

*Extension to noncontinuous spaces.* The principle underlying coarse coding can be generalized to noncontinuous spaces by thinking of a set of items as the equivalent of a receptive field. A local representation uses one unit for each possible item. A distributed representation uses a unit for a set of items, and it implicitly encodes a particular item as the intersection of the sets that correspond to the active units.

In the domain of spatial features there is generally a very strong regularity: Sets of features with similar parameter values need to have similar effects on other representations. Coarse coding is efficient because it allows this regularity to be expressed in the connection strengths. In other domains, the regularities are different, but the efficiency arguments are the same: It is better to devote a unit to a set of items than to a single item, provided that the set is chosen in such a way that membership in the set implies something about membership in other sets. This implication can then be captured as a connection strength. Ideally, a set should be chosen so that membership of this set has strong implications for memberships of other sets that are also encoded by individual units.

We illustrate these points with a very simple example. Consider a microlanguage consisting of the three-letter words of English made up of *w* or *l*, followed by *i* or *e*, followed by *g* or *r*. The strings *wig* and *leg* are words, but *weg*, *lig*, and all strings ending in *r* are not. Suppose we wanted to use a distributed representation scheme as a basis for representing the words, and we wanted to be able to use the distributed pattern as a basis for deciding whether the string is a word or a nonword. For simplicity we will have a single "decision" unit. The problem is to find connections from the units representing the word to the decision unit such that it fires whenever a word is present but does not fire when no word is present.[7]

[7] Note that the problem remains the same if the decision unit is replaced by a set of units and the task of the network is to produce a different pattern for the word and nonword decisions. For when we examine each unit, it either takes the same or a different value in the two patterns; in the cases where the value is the same, there is no problem, but neither do such units differentiate the two patterns. When the values are different, the unit behaves just like the single decision unit discussed in the text.

Figure 4 shows three representation schemes: a distributed scheme that does not work, a distributed scheme that does work, and a local scheme. In the first scheme, each letter/position combination is represented by a different unit. Since there are only five letter/position possibilities, only five units have connections to the output unit. Each word and nonword produces a different and unique pattern over these five units, but the connections from the five units to the decision unit cannot be set in such a way as to make the decision unit fire whenever one of the words is present and fail to fire whenever one of the nonwords is present.

The reason for the problem is simply that the connections between the letter/position units and the decision units can only capture the degree to which each letter indicates whether the string is a word or not. The *g* tends to indicate that a word is present, whereas the *r* indicates that the item is not a word; but each of the other letters, taken individually, has absolutely no predictive ability in this case.

Whether a letter string is a word or not cannot be determined conclusively from the individual letters it contains; it is necessary to consider also what *combinations* of letters it contains. Thus, we need a representation that captures what combinations of letters are present in a way that is sufficient for the purposes of the network. One could capture this by using local representations and assigning one node to each word, as in the third panel of Figure 4. However, it is important to see that one need not go all the way to local representations to solve the

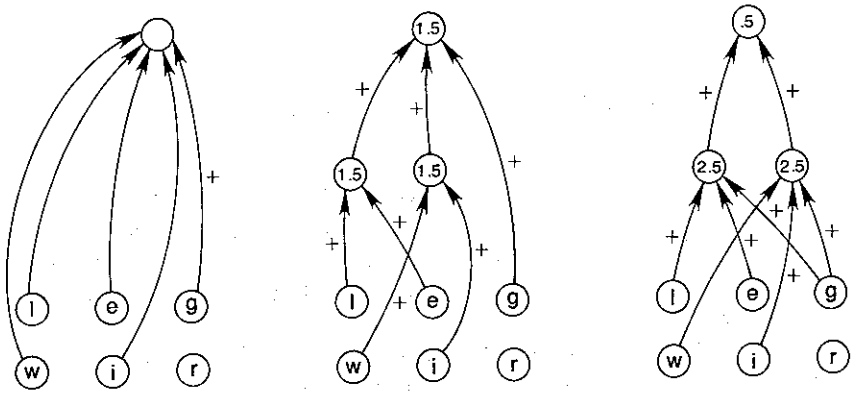


FIGURE 4. Three networks applied to the problem of determining which of the strings that can be made from *w* or *l*, followed by *i* or *e*, followed by *g* or *r* form words. Numbers on the connections represent connection strengths; numbers on the units represent the units' thresholds. A unit will take on an activation equal to 1 if its input exceeds it threshold; otherwise, its activation is 0.

problem facing our network. Conjunctive distributed representations will suffice.

The scheme illustrated in the second panel of the figure provides a conjunctive distributed representation. In this scheme, there are units for pairs of letters which, in this limited vocabulary, happen to capture the combinations that are essential for determining whether a string of letters is a word or not. These are, of course, the pairs *wi* and *le*. These conjunctive units, together with direct input to the decision unit from the *g* unit, are sufficient to construct a network which correctly classifies all strings consisting of a *w* or an *l*, followed by an *i* or an *e*, followed by a *g* or *r*.

This example illustrates that conjunctive coding is often necessary if distributed representations are to be used to solve problems that might easily be posed to networks. This same point could be illustrated with many other examples—the *exclusive or* problem is the classic example (Minsky & Papert, 1969). Other examples of problems requiring some sort of conjunctive encoding can be found in Hinton (1981a) and in Chapters 7 and 8. An application of conjunctive coding to a psychological model is found in Chapter 18.

Some problems (mostly very simple ones) can be solved without any conjunctive encoding at all, and others will require conjuncts of more than two units at a time. In general, it is hard to specify in advance just what "order" of conjunctions will be required. Instead, it is better to search for a learning scheme that can find representations that are adequate. The mechanisms proposed in Chapters 7 and 8 represent two steps toward this goal.

## Implementing an Arbitrary Mapping Between Two Domains

The attentive reader will have noticed that a *local* representation can always be made to work in the example we have just considered. However, we have already discussed several reasons why distributed representations are preferable. One reason is that they can make more efficient use of parallel hardware than local representations.

This section shows how a distributed representation in one group of units can cause an appropriate distributed representation in another group of units. We consider the problem of implementing an *arbitrary* pairing between representations in the two groups, and we take as an example an extension of the previous one: the association between the visual form of a word and its meaning. The reason for considering an arbitrary mapping is that this is the case in which local representations seem most helpful. If distributed representations are better in this

case, then they are certainly better in cases where there are underlying regularities that can be captured by regularities in the patterns of activation on the units in one group and the units in another. A discussion of the benefit distributed representations can provide in such cases can be found in Chapter 18.

If we restrict ourselves to monomorphemic words, the mapping from strings of graphemes onto meanings appears to be arbitrary in the sense that knowing what some strings of graphemes mean does not help one predict what a new string means.[8] This arbitrariness in the mapping from graphemes to meanings is what gives plausibility to models that have explicit word units. It is obvious that arbitrary mappings can be implemented if there are such units. A grapheme string activates exactly one word unit, and this activates whatever meaning we wish to associate with it (see Figure 5A). The semantics of similar grapheme strings can then be completely independent because they are mediated by separate word units. There is none of the automatic generalization that is characteristic of distributed representations.

Intuitively, it is not at all obvious that arbitrary mappings can be implemented in a system where the intermediate layer of units encodes the word as a distributed pattern of activity instead of as activity in a single local unit. The distributed alternative appears to have a serious drawback. The effect of a pattern of activity on other representations is the combined result of the individual effects of the active units in the pattern. So similar patterns tend to have similar effects. It appears that we are not free to make a given pattern have whatever effect we wish on the meaning representations without thereby altering the effects that other patterns have. This kind of interaction appears to make it difficult to implement arbitrary mappings from distributed representations of words onto meaning representations. We shall now show that these intuitions are wrong and that distributed representations of words can work perfectly well and may even be more efficient than single word units.

Figure 5B shows a three-layered system in which grapheme/position units feed into *word-set* units which, in turn, feed into *semantic* or *sememe* units. Models of this type, and closely related variants, have been analyzed by Willshaw (1981), V. Dobson (personal communication, 1984), and by David Zipser (personal communication, 1981); some further relevant analyses are discussed in Chapter 12. For simpli-

[8] Even for monomorphemic words there may be particular fragments that have associated meaning. For example, words starting with *sn* usually mean something unpleasant to do with the lips or nose (*sneer, snarl, snigger*), and words with long vowels are more likely to stand for large, slow things than words with short vowels (George Lakoff, personal communication). Much of Lewis Carroll's poetry relies on such effects.

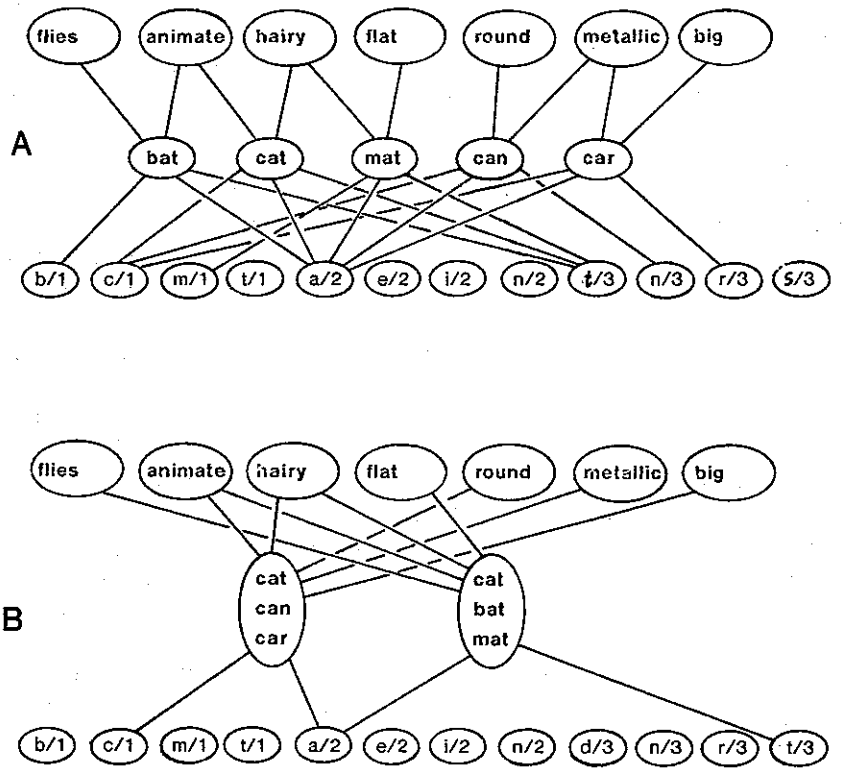


FIGURE 5. *A*: A three-layer network. The bottom layer contains units that represent particular graphemes in particular positions within the word. The middle layer contains units that recognize complete words, and the top layer contains units that represent semantic features of the meaning of the word. This network uses local representations of words in the middle layer. *B*: The top and bottom layers are the same as in (*A*), but the middle layer uses a more distributed representation. Each unit in this layer can be activated by the graphemic representation of any one of a whole set of words. The unit then provides input to every semantic feature that occurs in the meaning of *any* of the words that activate it. Only those word sets containing the word *cat* are shown in this example. Notice that the only semantic features which receive input from *all* these word sets are the semantic features of cat.

city, we shall assume that each unit is either active or inactive and that there is no feedback or cross-connections. These assumptions can be relaxed without substantially affecting the argument. A word-set unit is activated whenever the pattern of the grapheme/position units codes a word in a particular set. The set could be all the four-letter words starting with *HE*, for example, or all the words containing at least two *T*'s. All that is required is that it is possible to decide whether a word is in

the set by applying a simple test to the activated grapheme/position units. So, for example, the set of all words meaning "nice" is *not* allowed as a word set. There is an implicit assumption that word meanings can be represented as sets of sememes. This is a contentious issue. There appears to be a gulf between the componential view in which a meaning is a set of features and the structuralist view in which the meaning of a word can only be defined in terms of its *relationships* to other meanings. Later in this chapter we consider one way of integrating these two views by allowing articulated representations to be built out of a number of different sets of active features.

Returning to Figure 5B, the question is whether it is possible to implement an arbitrary set of associations between grapheme/position vectors and sememe vectors when the word-set units are each activated by more than one word. It will be sufficient to consider just one of the many possible specific models. Let us assume that an active word-set unit provides positive input to all the sememe units that occur in the meaning of any word in the word set. Let us also assume that each sememe unit has a variable threshold that is dynamically adjusted to be just slightly less than the number of active word-set units. Only sememe units that are receiving input from every active word-set unit will then become active.

All the sememes of the correct word will be activated because each of these sememes will occur in the meaning of one of the words in the active word sets. However, additional sememes may also be activated because, just by chance, they may receive input from every active word-set unit. For a sememe to receive less input than its threshold, there must be at least one active word set that does not contain any word which has the sememe as part of its meaning. For each active word set the probability, $i$, of this happening is

$$i = (1 - p)^{(w-1)}$$

where $p$ is the proportion of words that contain the sememe and $w$ is the number of words in the word set of the word-set unit. The reason for the term $w - 1$ is that the sememe is already assumed not to be part of the meaning of the correct word, so there are only $w - 1$ remaining words that could have it in their meaning.

Assume that when a word is coded at the graphemic level it activates $u$ units at the word-set level. Each sememe that is not part of the word's meaning has a probability $i$ of failing to receive input from each word-set unit. The probability, $f$, that all of these word-set units will provide input to it is therefore

$$f = (1 - i)^u$$
$$= [1 - (1 - p)^{(w-1)}]^u.$$

By inspection, this probability of a "false-positive" sememe reduces to zero when $w$ is 1. Table 1 shows the value of $f$ for various combinations of values of $p$, $u$, and $w$. Notice that if $p$ is very small, $f$ can remain negligible even if $w$ is quite large. This means that distributed representations in which each word-set unit participates in the representation of many words do not lead to errors if the semantic features are relatively sparse in the sense that each word meaning contains only a small fraction of the total set of sememes. So the word-set units can be fairly nonspecific provided the sememe units are fairly specific (not shared by too many different word meanings). Some of the entries in the table make it clear that for some values of $p$, there can be a negligible chance of error even though the number of word-set units is considerably less than the number of words (the ratio of words to word-set units is $w/u$).

The example described above makes many simplifying assumptions. For example, each word-set unit is assumed to be connected to *every* relevant sememe unit. If any of these connections were missing, we could not afford to give the sememe units a threshold equal to the number of active word-set units. To allow for missing connections we could lower the threshold. This would increase the false-positive error rate, but the effect may be quite small and can be compensated by adding word-set units to increase the specificity of the word-level representations (Willshaw, 1981). Alternatively, we could make each word-set unit veto the sememes that do not occur in any of its words. This scheme is robust against missing connections because the absence of one veto can be tolerated if there are other vetos (V. Dobson, personal communication, 1984).

There are two more simplifying assumptions both of which lead to an underestimate of the effectiveness of distributed representations for the arbitrary mapping task. First, the calculations assume that there is no fine-tuning procedure for incrementing some weights and decrementing others to improve performance in the cases where the most frequent errors occur. Second, the calculations ignore cross-connections among the sememes. If each word meaning is a familiar stable pattern of sememes, there will be a strong "clean-up" effect which tends to suppress erroneous sememes as soon as the pattern of activation at the sememe level is sufficiently close to the familiar pattern for a particular word meaning. Interactions among the sememes also provide an explanation for the ability of a single grapheme string (e.g., *bank*) to elicit two quite different meanings. The *bottom-up* effect of the activated

TABLE 1

| u | w | p | f | u | w | p | f | u | w | p | f |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 5 | .2 | 0.071 | 5 | 5 | .1 | 0.0048 | 5 | 5 | .01 | $9.5\times10^{-8}$ |
| 5 | 10 | .2 | 0.49 | 5 | 10 | .1 | 0.086 | 5 | 10 | .01 | $4.8\times10^{-6}$ |
| 5 | 20 | .2 | 0.93 | 5 | 20 | .1 | 0.48 | 5 | 20 | .01 | 0.00016 |
| 5 | 40 | .2 | 1.0 | 5 | 40 | .1 | 0.92 | 5 | 40 | .01 | 0.0036 |
| 5 | 80 | .2 | 1.0 | 5 | 80 | .1 | 1.0 | 5 | 80 | .01 | 0.049 |
| 10 | 10 | .2 | 0.24 | 10 | 10 | .1 | 0.0074 | 10 | 10 | .01 | $2.3\times10^{-11}$ |
| 10 | 20 | .2 | 0.86 | 10 | 20 | .1 | 0.23 | 10 | 20 | .01 | $2.5\times10^{-8}$ |
| 10 | 40 | .2 | 1.0 | 10 | 40 | .1 | 0.85 | 10 | 40 | .01 | $1.3\times10^{-5}$ |
| 10 | 80 | .2 | 1.0 | 10 | 80 | .1 | 1.0 | 10 | 80 | .01 | 0.0024 |
| 10 | 160 | .2 | 1.0 | 10 | 160 | .1 | 1.0 | 10 | 160 | .01 | 0.10 |
| 40 | 40 | .2 | 0.99 | 40 | 40 | .1 | 0.52 | 40 | 40 | .01 | $2.7\times10^{-20}$ |
| 40 | 80 | .2 | 1.0 | 40 | 80 | .1 | 0.99 | 40 | 80 | .01 | $3.5\times10^{-11}$ |
| 40 | 160 | .2 | 1.0 | 40 | 160 | .1 | 1.0 | 40 | 160 | .01 | 0.00012 |
| 40 | 320 | .2 | 1.0 | 40 | 320 | .1 | 1.0 | 40 | 320 | .01 | 0.19 |
| 40 | 640 | .2 | 1.0 | 40 | 640 | .1 | 1.0 | 40 | 640 | .01 | 0.94 |
| 100 | 100 | .2 | 1.0 | 10 | 100 | .1 | 0.99 | 100 | 100 | .01 | $9.0\times10^{-21}$ |
| 100 | 200 | .2 | 1.0 | 10 | 200 | .1 | 1.0 | 100 | 200 | .01 | $4.8\times10^{-7}$ |
| 100 | 400 | .2 | 1.0 | 100 | 400 | .1 | 1.0 | 100 | 400 | .01 | 0.16 |
| 100 | 800 | .2 | 1.0 | 100 | 800 | .1 | 1.0 | 100 | 800 | .01 | 0.97 |

The probability, $f$, of a false-positive sememe as a function of the number of active word-set units per word, $u$, the number of words in each word-set, $w$, and the probability, $p$, of a sememe being part of a word meaning.

word-set units helps both sets of sememes, but as soon as *top-down* factors give an advantage to one meaning, the sememes in the other meaning will be suppressed by competitive interactions at the sememe level (Kawamoto & Anderson, 1984).

*A simulation.* As soon as there are cross-connections among the sememe units and fine-tuning of individual weights to avoid frequent errors, the relatively straightforward probabilistic analysis given above breaks down. To give the cross-connections time to clean up the output, it is necessary to use an iterative procedure instead of the simple "straight-through" processing in which each layer completely determines the states of all the units in the subsequent layer in a single, synchronous step. Systems containing cross-connections, feedback, and asynchronous processing elements are probably more realistic, but they are generally very hard to analyze. However, we are now beginning to discover that there are subclasses of these more complex systems that behave in tractable ways. One example of this subclass is described in

more detail in Chapter 7. It uses processing elements that are inherently stochastic. Surprisingly, the use of stochastic elements makes these networks *better* at performing searches, *better* at learning, and *easier* to analyze.

A simple network of this kind can be used to illustrate some of the claims about the ability to "clean up" the output by using interactions among sememe units and the ability to avoid errors by fine-tuning the appropriate weights. The network contains 30 grapheme units, 20 word-set units, and 30 sememe units. There are no direct connections between grapheme and sememe units, but each word-set unit is connected to all the grapheme and sememe units. The grapheme units are divided into three sets of ten, and each three-letter word has one active unit in each group of ten (units can only have activity levels of 1 or 0). The "meaning" of a word is chosen at random by selecting each sememe unit to be active with a probability of 0.2. The network shown in Figure 6 has learned to associated 20 different grapheme strings with their chosen meanings. Each word-set unit is involved in the representation of many words, and each word involves many word-set units.

The details of the learning procedure used to create this network and the search procedure which is used to settle on a set of active sememes when given the graphemic input are described in Chapter 7. Here we simply summarize the main results of the simulation.

After a long period of learning, the network was able to produce the correct pattern of sememes 99.9% of the time when given a graphemic input. Removal of any one of the word-set units after the learning typically caused a slight rise in the error rate for several different words rather than the complete loss of one word. Similar effects have been observed in other distributed models (Wood, 1978). In our simulations, some of the erroneous responses were quite interesting. In 10,000 tests with a missing word-set unit there were 140 cases in which the model failed to recover the right sememe pattern. Some of these consisted of one or two missing or extra sememes, but 83 of the errors were exactly the pattern of sememes of some other word. This is a result of the cooperative interactions among the sememe units. If the input coming from the word-set units is noisy or underspecified as it may be when units are knocked out, the clean-up effect may settle on a similar but incorrect meaning.

This effect is reminiscent of a phenomenon called *deep dyslexia* which occurs with certain kinds of brain damage in adults. When shown a word and asked to read it, the subject will sometimes say a different word with a very similar meaning. The incorrect word sometimes has a very different sound and spelling. For example, when shown the word *PEACH*, the subject might say *APRICOT*. (See Coltheart, Patterson, & Marshall, 1980, for more information about acquired dyslexia.)

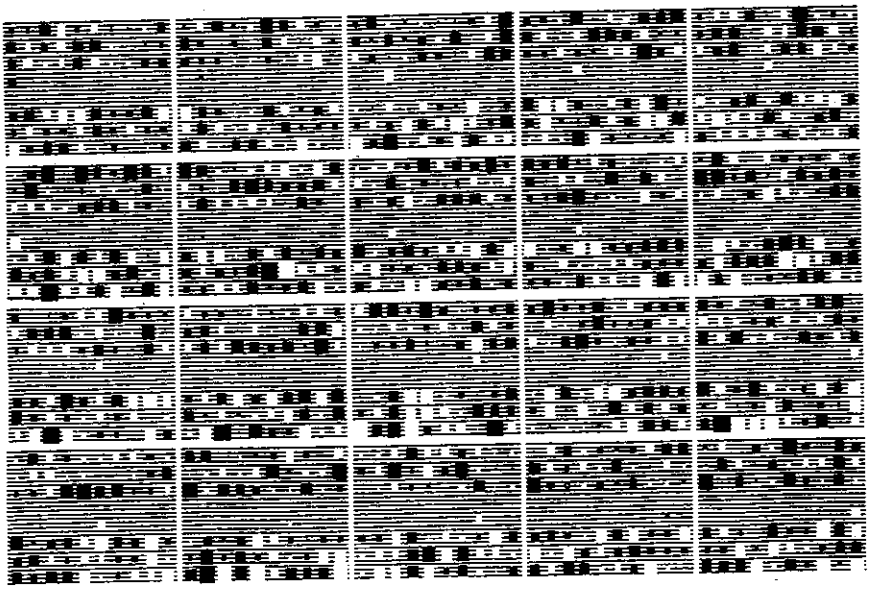

FIGURE 6. A compact display that shows all the connection strengths of the 20 units in the middle layer of a three-layer network. The network can map from a pattern of activity over the 30 units in the bottom layer (representing graphemes) to an associated pattern of activity over the 30 units of the top layer (representing sememes). Within each of the large rectangles that are used to depict middle-layer units, the 30 black and white rectangles at the top depict the weights of the connections to the top layer, and the 30 rectangles at the bottom depict the weights from the bottom layer. White rectangles are positive weights, black are negative, and the area of a rectangle depicts the magnitude of the weight. The single weight that occurs somewhere in the middle of a unit is its threshold (black means a positive threshold). The weights between the 30 units in the top layer are not shown in this display.

Semantic errors of this kind seem bizarre because it seems as if the subject must have accessed the lexical item *PEACH* in order to make the semantically related error, and if he can get to the lexical item why can't he say it? (These subjects may know and be able to say the words that they misread.) Distributed representations allow us to dispense with the rigid distinction between accessing a word and not accessing it. In a network that has learned the word *PEACH*, the graphemic representation of *PEACH* will cause approximately the right input to the sememe units, and interactions at the sememe level can then cause exactly the pattern of sememes for *APRICOT*. Another psychologically interesting effect occurs when the network relearns after

it has been damaged. The network was damaged by adding noise to every connection that involved a word-set unit. This reduced the performance from 99.3% correct to 64.3%.[9] The network was then retrained and it exhibited very rapid relearning, much faster than its original rate of learning when its performance was 64.3% correct. This rapid recovery was predicted by a geometrical argument which shows that there is something special about a set of connection strengths that is generated by adding noise to a near-perfect set. The resulting set is very different from other sets of connection strengths that exhibit the same performance. (See Chapter 7 for further discussion.)

An even more surprising effect occurs if a few of the words are omitted from the retraining. The error rate for these words is substantially reduced as the retraining proceeds, even though the other grapheme-sememe pairings have no intrinsic relation to them because all the pairings were selected randomly. The "spontaneous" recovery of words that the network is not shown again is a result of the use of distributed representations. *All* the weights are involved in encoding the subset of the words that are shown during retraining, and so the added noise tends to be removed from *every* weight. A scheme that used a separate unit for each word would not behave in this way, so one can view spontaneous recovery of unrehearsed items as a qualitative signature of distributed representations.

## STRUCTURED REPRESENTATIONS AND PROCESSES

In this section we consider two extensions of distributed representations. These extensions illustrate that the idea of distributed representations is consistent with some of the major insights from the field of artificial intelligence concerning the importance of structure in representations and processes. Perhaps because some proponents of distributed representations have not been particularly attuned to these issues, it is often unclear how structure is to be captured in a distributed representational scheme. The two parts of this section give some indication of the directions that can be taken in extending distributed representations to deal with these important considerations.

[9] The error rate was 99.3% rather than 99.9% in this example because the network was forced to respond faster, so the cooperative effects had less time to settle on the optimal output.

## Representing Constituent Structure

Any system that attempts to implement the kinds of conceptual structures that people use has to be capable of representing two rather different kinds of hierarchy. The first is the "IS-A" hierarchy that relates types to instances of those types. The second is the part/whole hierarchy that relates items to the constituent items that they are composed of. The most important characteristics of the IS-A hierarchy are that known properties of the types must be "inherited" by the instances, and properties that are found to apply to all instances of a type must normally be attributed to the type. Earlier in this chapter we saw how the IS-A hierarchy can be implemented by making the distributed representation of an instance *include*, as a subpart, the distributed representation for the type. This representational trick automatically yields the most important characteristics of the IS-A hierarchy, but the trick can only be used for one kind of hierarchy. If we use the part/whole relationship between patterns of activity to represent the type/instance relationship between items, it appears that we cannot also use it to represent the part/whole relationship between items. We cannot make the representation of the whole be the sum of the representations of its parts.

The question of how to represent the relationship between an item and the constituent items of which it is composed has been a major stumbling block for theories that postulate distributed representations. In the rival, localist scheme, a whole is a node that is linked by labeled arcs to the nodes for its parts. But the central tenet of the distributed scheme is that different items correspond to *alternative* patterns of activity in the same set of units, so it seems as if a whole and its parts cannot both be represented at the same time.

Hinton (1981a) described one way out of this dilemma. It relies on the fact that wholes are not simply the sums of their parts. They are composed of parts that play particular roles within the whole structure. A shape, for example, is composed of smaller shapes that have a particular size, orientation, and position relative to the whole. Each constituent shape has its own spatial role, and the whole shape is composed of a set of shape/role pairs.[10] Similarly, a proposition is composed of objects that occupy particular semantic roles in the whole propositional

[10] Relationships between parts are important as well. One advantage of explicitly representing shape/role pairs is that it allows different pairs to support each other. One can view the various different locations within an object as slots and the shapes of parts of an object as the fillers of these slots. Knowledge of a whole shape can then be implemented by positive interactions between the various slot-fillers.

structure. This suggests a way of implementing the relationship between wholes and parts: The identity of each part should first be combined with its role to produce a single pattern that represents the *combination* of the identity and the role, and then the distributed representation for the whole should consist of the sum of the distributed representations for these identity/role combinations (plus some additional "emergent" features). This proposal differs from the simple idea that the representation of the whole is the sum of the representations of its parts because the subpatterns used to represent identity/role combinations are quite different from the patterns used to represent the identities alone. They do not, for example, contain these patterns as parts.

Naturally, there must be an access path between the representation of an item as a whole in its own right and the representation of that same item playing a particular role within a larger structure. It must be possible, for example, to generate the identity/role representation from two separate, explicit, distributed patterns one of which represents the identity and the other of which represents the role. It must also be possible to go the other way and generate the explicit representations of the identity and role from the single combined representation of the identity/role combination (see Figure 7).

The use of patterns that represent identity/role combinations allows the part/whole hierarchy to be represented in the same way as the type/instance hierarchy. We may view the whole as simply a particular instance of a number of more general types, each of which can be defined as the type that has a particular kind of part playing a particular role (e.g., men with wooden legs).

## Sequential Symbol Processing

If constituent structure is implemented in the way described above, there is a serious issue about how many structures can be active at any one time. The obvious way to allocate the hardware is to use a group of units for each possible role within a structure and to make the pattern of activity in this group represent the identity of the constituent that is currently playing that role. This implies that only one structure can be represented at a time, unless we are willing to postulate multiple copies of the entire arrangement. One way of doing this, using units with programmable rather than fixed connections, is described in Chapter 16. However, even this technique runs into difficulties if more than a few modules must be "programmed" at once. However, people do seem to suffer from strong constraints on the number of structures of the same general type that they can process at once. The

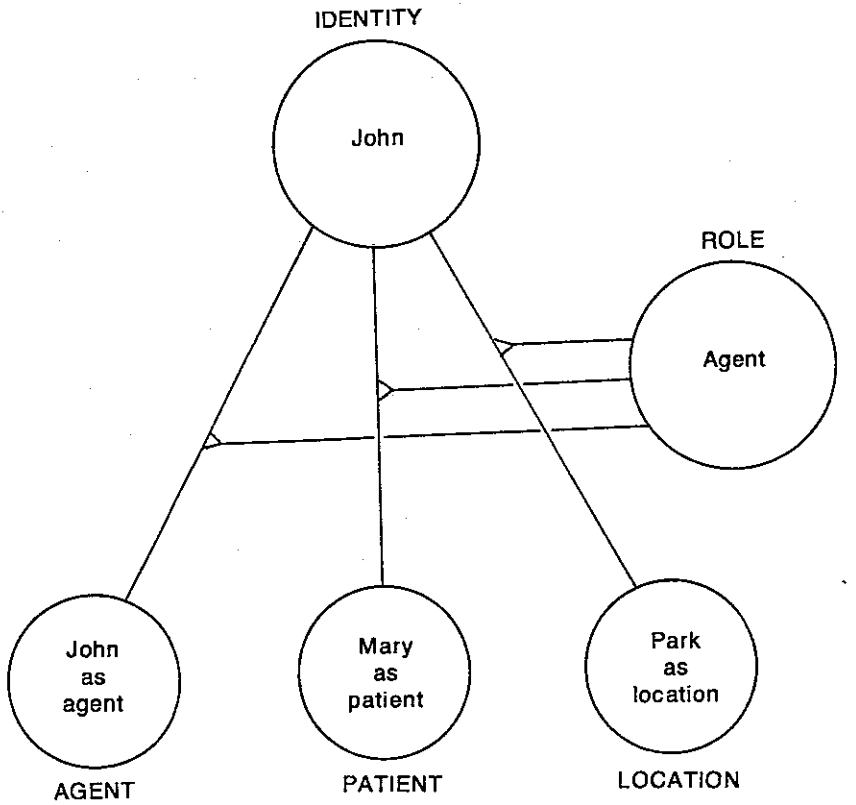


FIGURE 7. A sketch of the apparatus that might be necessary for combining separate representations of an identity and a role into a single pattern. Only one identity and only one role can be explicitly represented at a time because the identity and role groups can each have only one pattern of activity at a time. However, the various role groups allow many identity/role combinations to be encoded simultaneously. The small triangular symbols represent the ability of the pattern of activity in the group that explictly represents a role to determine which one of the many role groups is currently interacting with the identity group. This allows the identity occupying a particular role to be "read out" as well as allowing the reverse operation of combining an identity and a role.

sequentiality that they exhibit at this high level of description is initially surprising given the massively parallel architecture of the brain, but it becomes much easier to understand if we abandon our localist predelictions in favor of the distributed alternative which uses the parallelism to give each active representation a very rich internal structure that allows the right kinds of generalization and content-addressability. There may be some truth to the notion that people are sequential *symbol* processors if each "symbolic representation" is identified with a

successive state of a large interactive network. See Chapter 14 for further discussion of these issues.

One central tenet of the sequential symbol processing approach (Newell, 1980) is the ability to focus on any part of a structure and to expand that into a whole that is just as rich in content as the original whole of which it was a part. The recursive ability to expand parts of a structure for indefinitely many levels and the inverse ability to package up whole structures into a reduced form that allows them to be used as constituents of larger structures is the essence of symbol processing. It allows a system to build structures out of things that refer to other whole structures without requiring that these other structures be represented in all their cumbersome detail.

In conventional computer implementations, this ability is achieved by using pointers. These are very convenient, but they depend on the use of addresses. In a parallel network, we need something that is functionally equivalent to arbitrary pointers in order to implement symbol processing. This is exactly what is provided by subpatterns that stand for identity/role combinations. They allow the full identity of the part to be accessed from a representation of the whole and a representation of the role that the system wishes to focus on, and they also allow explicit representations of an identity and a role to be combined into a less cumbersome representation, so that several identity/role combinations can be represented simultaneously in order to form the representation of a larger structure.

## SUMMARY

Given a parallel network, items can be represented by activity in a single, local unit or by a pattern of activity in a large set of units with each unit encoding a microfeature of the item. Distributed representations are efficient whenever there are underlying regularities which can be captured by interactions among microfeatures. By encoding each piece of knowledge as a large set of interactions, it is possible to achieve useful properties like content-addressable memory and automatic generalization, and new items can be created without having to create new connections at the hardware level. In the domain of continuously varying spatial features it is relatively easy to provide a mathematical analysis of the advantages and drawbacks of using distributed represention.

Distributed representations *seem* to be unsuitable for implementing purely arbitrary mappings because there is no underlying structure and so generalization only causes unwanted interference. However, even

for this task, distributed representations can be made fairly efficient and they exhibit some psychologically interesting effects when damaged.

There are several difficult problems that must be solved before distributed representations can be used effectively. One is to decide on the pattern of activity that is to be used for representing an item. The similarities between the chosen pattern and other existing patterns will determine the kinds of generalization and interference that occur. The search for good patterns to use is equivalent to the search for the underlying regularites of the domain. This learning problem is addressed in the chapters of Part II.

Another hard problem is to clarify the relationship between distributed representations and techniques used in artificial intelligence like schemas, or hierarchical structural descriptions. Existing artificial intelligence programs have great difficulty in rapidly finding the schema that best fits the current situation. Parallel networks offer the potential of rapidly applying a lot of knowledge to this best-fit search, but this potential will only be realized when there is a good way of implementing schemas in parallel networks. A discussion of how this might be done can be found in Chapter 14.

## ACKNOWLEDGMENTS

This chapter is based on a technical report by the first author, whose work is supported by a grant from the System Development Foundation. We thank Jim Anderson, Dave Ackley, Dana Ballard, Francis Crick, Scott Fahlman, Jerry Feldman, Christopher Longuet-Higgins, Don Norman, Terry Sejnowski, and Tim Shallice for helpful discussions.