Mapping Part–Whole Hierarchies into Connectionist Networks*

Geoffrey E. Hinton
Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4

ABSTRACT
Three different ways of mapping part–whole hierarchies into connectionist networks are described. The simplest scheme uses a fixed mapping and is inadequate for most tasks because it fails to share units and connections between different pieces of the part–whole hierarchy. Two alternative schemes are described, each of which involves a different method of time-sharing connections and units. The scheme we finally arrive at suggests that neural networks have two quite different methods for performing inference. Simple "intuitive" inferences can be performed by a single settling of a network without changing the way in which the world is mapped into the network. More complex "rational" inferences involve a sequence of such settlings with mapping changes after each settling.

1. Introduction

One reason why many AI researchers are sceptical about connectionist networks that use distributed representations is that it is hard to imagine how complex, articulated structures can be represented and processed in these networks. The approach would be far more convincing if it could come up with a sensible scheme for representing the meaning of a sentence such as: "She seems to be more at ease with her fellow students than with me, her adviser." (Drew McDermott, personal communication). This meaning is clearly composed of several major constituents with relationships between them, and each major constituent has its own, complex, internal structure. A representational scheme for dealing with meanings of this complexity must, at the very least, specify how it is possible to focus attention on the constituents of the whole and how it is possible, in some sense, to have the whole meaning in mind at once.

The example given above is typical of examples from many different domains. It appears that whenever people have to deal with complexity they

* This research was supported by grants from the Alfred P. Sloan Foundation and the Ontario Information Technology Research Center. The author is a fellow of the Canadian Institute for Advanced Research.

impose part–whole hierarchies in which objects at one level are composed of inter-related objects at the next level down. In representing a visual scene or an everyday plan or the structure of a sentence we use hierarchical structures of this kind. The main issue addressed in this paper is how to map complex part–whole hierarchies into the fixed hardware of a connectionist network. The main conclusion is that it is essential to use some form of timesharing so that a portion of the connectionist network is used, at different times, to represent different parts of the part–whole hierarchy.

Most existing connectionist simulations do not use timesharing of the connectionist apparatus because they focus on computations that can be performed rapidly by parallel constraint satisfaction, and they typically ignore the issue of how the real world gets mapped into the bottom level units in the network or how the results produced by the network are integrated over longer periods of time. These simulations are best viewed as investigations of the computations that can be done by one internally parallel module. At best, they give little insight into how complex part–whole hierarchies should be mapped into connectionist networks, and at worst they lend support to the naive idea that the entire part–whole hierarchy should be mapped simultaneously using a fixed, inflexible mapping (as described in Section 5 below).

Given any finite connectionist network, we can always design a task that is so difficult that it cannot all be done in parallel by a single settling of the network. The task can be designed to have subtasks that require the same knowledge to be applied to different data, and although we can replicate portions of the network so that some of these subtasks can be performed in parallel, we will eventually run out of hardware and will be driven to use time instead of space. So eventually we have to face the issue of timesharing a module of the network between different pieces of a single task. This inevitably leads to questions of how we implement a flexible mapping of pieces of the task into a module of the network, how we store the intermediate results produced by the module so that it can be liberated to solve the next subtask, and how we use intermediate results to determine which subtask is tackled next. These questions have been widely ignored within neural network research, particularly within the sub-areas that have been inspired by physics and biology.

2. Symbols and the Conventional Implementation of Hierarchical Structures

It will be helpful to begin by reviewing the standard way of implementing hierarchical data-structures in a conventional digital computer. There are obviously many minor variations, but a suitable paradigm example is the kind of record structure that is found in languages like C. Each instance of a record is composed of a pre-determined set of fields (sometimes called "slots" or "roles") each of which contains a pointer to the contents of the field which may be either another instance of a record, or a primitive object. Since the pointers can be arbitrary addresses, this is a very flexible way of implementing a hierarchical data-structure, but the flexibility is bought at the price of the von Neumann bottleneck: The addressing mechanism means that only one pointer can be followed at a time.[1]

The addresses act as symbols for expressions, and they illustrate the essence of a symbol: It is a small representation of an object that provides a "remote access" path to a fuller representation of the same object.[2] In general, this fuller representation is itself composed of small representations (e.g. the addresses of the structures that fill the fields of the record). Because a symbol is small, many symbols can be put together to create a "fully-articulated" representation of some larger structure and the size of this fully-articulated representation need not be any larger than the fully-articulated representations of its constituents.

When addresses are used as symbols, there is normally an arbitrary relationship between the internal structure of a symbol and the fully articulated representation to which it provides access. Looking at the individual bits in the symbol provides no information about what it represents. Occasionally this is not quite true. If, for example, one type of data-structure is kept in the top half of memory and another type in the bottom half, the first bit of a symbol reveals the type of the data-structure to which it provides access. So it is possible to check the type without following the pointer. This trick can obviously be extended so that many of the bits in a symbol convey useful information. A symbol can then be viewed as a "reduced description" of the object.

One conclusion of this paper is that patterns of activity in some parts of a connectionist network need to exhibit the double life that is characteristic of symbols. The patterns must allow remote access to fuller representations, but so long as the patterns are also reduced descriptions this remote access need only be used very occasionally (e.g. a few times per second in a person). Most of the processing can be done by parallel constraint satisfaction on the patterns themselves. One interesting consequence of using parallel constraint satisfaction as a powerful but somewhat inflexible inner loop in a sequential process is that it leads to two quite different ways of performing an inference.

3. Rational and Intuitive Inference

Given a parallel network, some inferences can be performed very efficiently by simply allowing the network to settle down into a stable state [28]. The states or external inputs of a subset of the units are fixed to represent the premises,

[1] Architectures such as the Connection Machine [8] use routing hardware that allows many pointers to be followed at once.

[2] There is, of course, much debate about the meaning of the word "symbol." The informal definition given here emerged from conversations with Allen Newell.

and when the network has settled down, the conclusion is represented by the states of some other subset of the units. A large amount of knowledge about the domain can influence the settling process, provided the knowledge is in the form of connection strengths. This method of performing inference by a single settling of a network will be called "intuitive inference." More complex inferences require a more serial approach in which parts of the network are used for performing several different intuitive inferences in sequence. This will be called "rational inference." The distinction between these two kinds of inference is not simply a serial versus parallel distinction. A network that is settling to a single stable state typically requires a series of iterations. Also, it may exhibit another emergent type of seriality during a single settling because easily drawn conclusions may emerge early in the settling. So even within one settling a network can exhibit something that looks like sequential inference [27]. This interesting phenomenon makes it clear that the crucial criterion for distinguishing rational from intuitive inference is not seriality. The defining characteristic of rational inference is that the way in which entities in the domain are mapped into the hardware changes during the course of the inference.

The distinction between these two types of inference applies quite well to a conventional computer. Intuitive inferences correspond, roughly, to single machine instructions and rational inferences correspond to sequences of machine instructions that typically involve changes in the way in which parts of the task are mapped into the CPU. Moreover, the very same inference can sometimes be performed in different ways. The task of multiplying two integers, for example, can be performed in a single instruction by dedicated hardware, or it can be performed by a sequential program. In the first case the inference is very fast but is limited in flexibility. It may work well for 32 bit numbers but not for 33 bit numbers.

The idea that the same inference can be performed in radically different ways is important in defending connectionist research against the claim of Fodor and Pylyshyn [6] that connectionist networks which do not implement classical symbol processing are simply a revival of discredited associationism. To characterize a multiplier chip as simply associating the input with the correct output is misleading. A chip that multiplies two N-bit numbers to produce a $2N$-bit number is able to use far less than the $O(2^{2N})$ components required by a table look-up scheme because it captures a lot of regularities of the task in the hardware. Of course, we can always render it useless by using bigger numbers, but this does not mean that it has failed to capture the structure of multiplication. A computer designer would be ill-advised to leave out hardware multipliers just because they cannot cope with all numbers. Similarly, a theoretical psychologist would be ill-advised to leave out parallel modules that perform fast intuitive inference just because such modules are not the whole story.

One big difference between computers and people is in the amount of computation that can be done in an intuitive inference. A computer typically breaks up a computation into very many, very small machine instructions that are executed in sequence. For the computations that people can do well, they typically use a few sequential steps each of which involves a computationally intensive intuitive inference. So we can think of people as "huge instruction set computers." This view is quite close to Fahlman's idea [5] that a network of simple processors could make important operations such as set intersection or transitive closure almost as fast as a single machine instruction. The enormous difference between people and conventional computers in the amount of computation that gets done by a single intuitive inference may explain why many psychologists and even some AI researchers find typical AI accounts of natural language processing so implausible and are attracted to connectionist accounts even though the performance of connectionist models is currently much worse. In a typical AI model, understanding a sentence involves an enormous amount of sequential symbol processing. To rephrase this in our new terminology, typical AI models use rational inference to do almost everything, probably because this is the convenient way to get things done on a conventional computer.

A further difference between people and computers is that a computer does not change its instruction set as it runs, whereas people seem to be capable of taking frequently repeated sequences and eliminating the sequential steps so that an inference that was once rational becomes intuitive. A sketch of how this could happen in a connectionist network is given in Section 7.4. Of course, it is also possible to model this process in software on a conventional computer. One example is the SOAR system [18]. Another example is a checkers program which may start by using a deep mini-max search in a particular situation, but after using the results of deep searches to learn a better evaluation function may be able to arrive at the same conclusions with a much shallower search [25]. Berliner [2] uses the terms "reasoning" and "judgment" to denote the two kinds of inference in the context of game playing programs.

A good example of a large computation that can be performed by a single intuitive inference in a connectionist network is the task of completing a schema when given a subset of the slot-fillers [4, 9]. In a familiar domain, there will be many constraints between the fillers of the various slots in a schema. If the appropriate representations are used, it is possible to express these constraints as connection strengths which all act in parallel to determine the most plausible fillers for unfilled slots. Section 4 gives a detailed example and shows how connectionist learning techniques can be used to discover both the constraints and the representations that allow these constraints to be expressed effectively. Sections 5, 6, and 7 then describe three alternative ways of mapping a part–whole hierarchy into a connectionist network.

4. Learning to Perform Intuitive Inference

This section illustrates the kind of inference that can be performed by a connectionist network in a single settling. The example shows that a single settling can do more than just associate an input with an output. It can perform a simple inference. The example was first described in [10] and uses the backpropagation learning procedure operating in a layered, feedforward network. The equivalent of "settling" in such a network is a single forward pass from the input to the output. To make this example more compatible with the recurrent networks that are implicitly assumed in the rest of this paper, it would be necessary to reimplement it using one of the gradient descent learning techniques for recurrent networks.[3]

4.1. The family trees task

Figure 1 shows two family trees. All the information in these trees can be represented in simple propositions of the form (person1, relationship, person2). These propositions can be stored as the stable states of activity of a neural network which contains a group of units for the role person1, a group for the role relationship and a group for the role person2. The net will also require further groups of units in order to achieve the correct interactions between the three role-specific groups. Figure 2 shows a network in which one further group

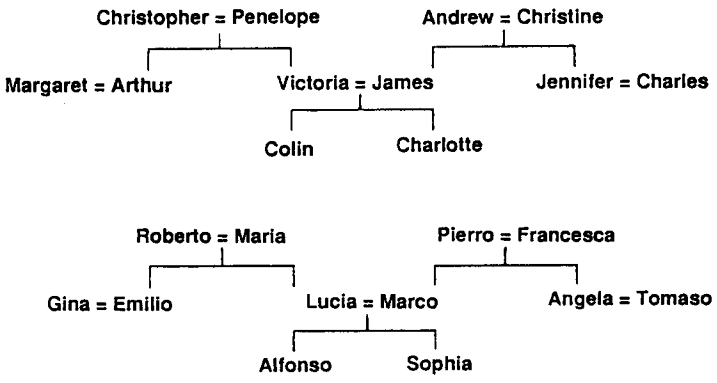

Fig. 1. Two isomorphic family trees. The symbol "=" means "married to."

[3] Rumelhart et al. [24] describe another version of the procedure which does not require a layered net. It works for arbitrary recurrent networks, but requires more complex units that remember their history of activity levels. Pineda [22] describes an alternative to backpropagation for recurrent networks that settle to stable states. Hinton [11] describes an efficient deterministic version of the Boltzmann machine learning procedure that could also be used for this task. These learning procedures for recurrent nets have not been tried on the family trees task.

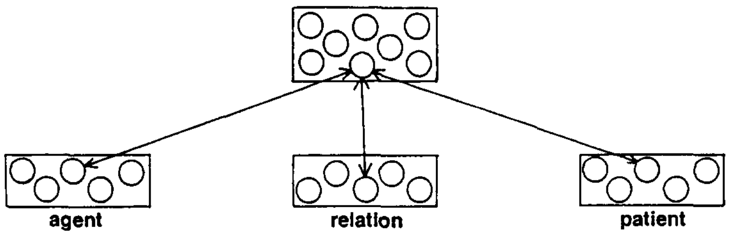

Fig. 2. An extra group of units can be used to implement higher-order constraints between the role-specific patterns.

has been introduced for this purpose. Units in this extra group detect combinations of features in the role-specific groups and can be used for causing appropriate completions of partial patterns. Suppose, for example, that one of the extra units becomes active whenever person1 is old and relationship requires that both people be the same age (e.g. the relationship has-husband in the very conventional domain we use). The extra unit can then activate the unit that represents the feature old within the person2 group. An extra unit that works in this way will be said to encode a micro-inference. It uses some of the features of some of the role-fillers to infer some of the features of other role-fillers and it is typically useful in encoding many different propositions rather than just a single one. By dedicating a unit to a micro-inference that is applicable in many different propositions, the network makes better use of the information carrying capacity of its activity levels and its weights than if it dedicated a single extra unit to each proposition. This is an example of the technique of coarse-coding described in [13]. In describing how a micro-inference could be implemented, we assumed that there was a single unit within the person1 group that was active whenever the pattern of activity in that group encoded an old person. This would not be true using random patterns, but it would be true using a componential representation.

Micro-inferences store propositions by encoding the underlying regularities of a domain. This form of storage has the advantage that it allows sensible generalization. If the network has learned the micro-inference given above it will have a natural tendency to make sensible guesses. If, for example, it is told enough about a new person, Jane, to know that Jane is old and it is then asked to complete the proposition Jane has-husband? it will expect the filler of the person2 role to be old. To achieve this kind of generalization of domain-specific regularities, it is necessary to pick a representation for Jane in the person1 role that has just the right active units so that the existing micro-inferences can cause the right effects in the other role-specific groups. A randomly chosen pattern will not do.

The real criterion for a good set of role-specific representations is that it makes it easy to express the regularities of the domain. It is sensible to dedicate a unit to a feature like old because useful micro-inferences can be expressed in terms of this feature. There is another way of stating this point which enables us to avoid awkward questions about whether the network really understands what old means. Instead of saying that activity in a unit means that the person is old, we can simply specify the set of people for which the unit is active. Each unit then corresponds to a way of partitioning all the people into two subsets, and good representations are ones for which these partitions are helpful in expressing the regularities. The search for good representations is then a search in the space of possible sets of partitions.[4]

4.2. Giving the network the freedom to choose representations

The network shown in Fig. 2 has the disadvantage that it is impossible to present a proposition to the network without already having decided on the patterns of activity that represent the people and relationships. We would like the network to use its experience of a set of propositions to construct its own internal representations of concepts, and so we must have a way of presenting the propositions that is neutral with respect to the various possible internal representations. Figure 3 shows how this can be done. The network translates a neutral input representation in which each person or relationship is represented by a single active unit into its own internal representation before making any associations. In the input representation, all pairs of concepts are equally

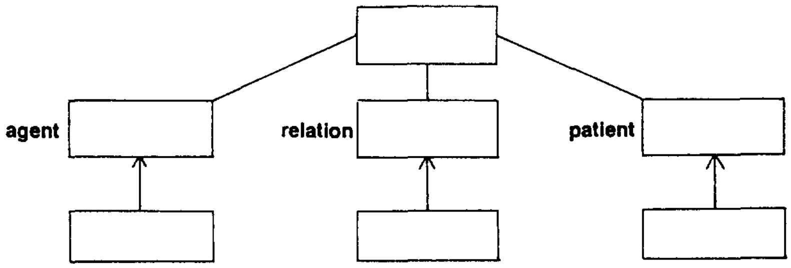

Fig. 3. The state of each role-specific group can be fixed via a special input group. By varying the weights between the special input groups and the role-specific groups the network can develop its own role-specific representations instead of being forced to use representations that are pre-determined.

[4] If the units can have intermediate activity levels or can behave stochastically, they do not correspond to clean cut partitions because there will be borderline cases. They are more like fuzzy sets, but the formal apparatus of fuzzy set theory (which is what defines the meaning of "fuzzy") is of no help here so we refrain from using the term "fuzzy." In much of what follows we talk as if units define clearcut sets with no marginal cases. This is just a useful idealisation.

similar. But we expect that the network will develop a hidden representation in which similar patterns of activity are used to represent people who have similar relationships to other people.

4.3. Distorting the task so that backpropagation can be used

To use the backpropagation learning procedure we need to express the task of learning about family relationships in a form suitable for a layered feed-forward network. There are many possible layered networks for this task and so our choice is somewhat arbitrary: We are merely trying to show that there is at least one way of doing it, and we are not claiming that this is the best or only way. The network we used is shown in Fig. 4. It has a group of input units for the filler of the person1 role, and another group for the filler of the relationship role. The output units represent the filler of the person2 role, so the network

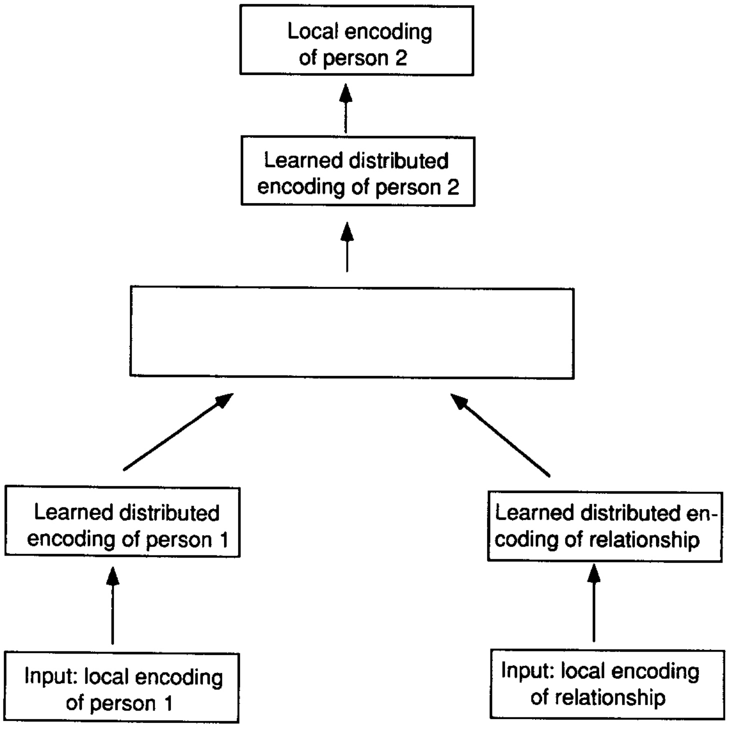
Local encoding of person 2
Learned distributed encoding of person 2
Learned distributed encoding of person 1
Learned distributed en-coding of relationship
Input: local encoding of person 1
Input: local encoding of relationship

Fig. 4. The architecture of the network used for the family trees task. It has three hidden layers in which it constructs its own representations. The input and output layers are forced to use localist encodings.

can only be used to complete propositions when given the first two terms.[5] The states of the units in the input groups are clamped from outside and the network then determines the states of the output units and thus completes the proposition.

For some relationships, like uncle, there may be several possible fillers for the person2 role that are compatible with a given filler of the person1 role. In a stochastic network it would be reasonable to allow the network to choose one of the possibilities at random. In the deterministic network we decided to insist on an output which explicitly represented the whole set of possible fillers. This is easy to do because the neutral representation that we used for the output has a single active unit for each person and so there is an obvious representation for a set of people.

Using the relationships father, mother, husband, wife, son, daughter, uncle, aunt, brother, sister, nephew, niece there are 104 instances of relationships in the two family trees shown in Fig. 1. We trained the network on 100 of these instances. The details of the training are given in [10]. The training involved weight-decay which ensures that the final magnitude of a weight is proportional to the amount of work that it does in reducing the error in the output. This means that weights which are unimportant for the performance of the network shrink to near zero, which makes it much easier to interpret the weight displays. After 1500 sweeps through all 100 training examples the weights were very stable and the network performed correctly on all the training examples: When given a person1 and a relationship as input it always produced activity levels greater than 0.8 for the output units corresponding to correct answers and activity levels of less than 0.2 for all the other output units.

The fact that the network can learn the examples it is shown is not particularly surprising. Any associative memory or table look-up scheme could do that. The interesting questions are: Does it create sensible internal representations for the various people and relationships that make it easy to express regularities of the domain that are only implicit in the examples it is given? Does it generalize correctly to the remaining examples? Does it make use of the isomorphism between the two family trees to allow it to encode them more efficiently and to generalize relationships in one family tree by analogy to relationships in the other? If it does all these things, it seems reasonable to say that it is doing inference rather than mere association.

4.4. The representations

Figure 5 shows the weights on the connections from the 24 units that are used

[5] We would have preferred it to perform completion when given any two terms. This could have been done by using a bigger network in which there were three input groups and three output groups, but learning would have been slower in the larger network and so we opted for the simpler case.

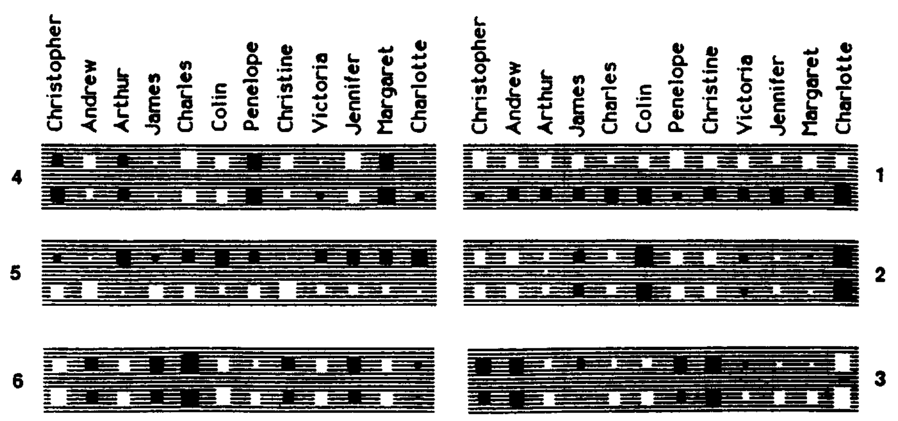

Fig. 5. The weights from the 24 input units that represent people to the 6 units in the second layer that learn distributed representations of people. White rectangles stand for excitatory weights, black for inhibitory weights, and the area of the rectangle encodes the magnitude of the weight. The weights from the 12 English people are in the top row of each unit. Beneath each of these weights is the weight from the isomorphic Italian.

to give a neutral input representation of person1 to the 6 units that are used for the network's internal, distributed representation of person1. These weights define the "receptive field" of each of the 6 units in the space of people. It is clear that at least one unit (unit number 1) is primarily concerned with the distinction between English and Italian. Moreover, most of the other units ignore this distinction which means that the representation of an English person is very similar to the representation of their Italian equivalent. The network is making use of the isomorphism between the two family trees to allow it to share structure and it will therefore tend to generalize sensibly from one tree to the other.

Unit 2 encodes which generation a person belongs to. Notice that the middle generation is encoded by an intermediate activity level. The network is never explicitly told that generation is a useful three-valued feature. It discovers this for itself by searching for features that make it easy to express the regularities of the domain. Unit 6 encodes which branch of the family a person belongs to. Again, this is useful for expressing the regularities but is not at all explicit in the examples.[6]

[6] In many tasks, features that are useful for expressing regularities between concepts are also observable properties of the individual concepts. For example, the feature male is useful for expressing regularities in the relationships between people and it is also related to sets of observable properties like hairyness and size. We carefully chose the input representation to make the problem difficult by removing all local cues that might have suggested the appropriate features.

It is initially surprising that none of the 6 units encodes sex. This is because of the particular set of relationship terms that was used. Each of the 12 relationship terms completely determines the sex of person2 so the sex of person1 is redundant. If we had included relationships like spouse there would have been more pressure to encode the sex of person1 because this would have been useful in constraining the possible fillers of the person2 role.

4.5. Generalization

The network was trained on 100 of the 104 instances of relationships in the two family trees. It was then tested on the remaining four instances. The whole training and testing procedure was repeated twice, starting from different random weights. In one case the network got all four test cases correct and in the other case it got 3 out of 4, where "correct" means that the output unit corresponding to the right answer had an activity level above 0.5, and all the other output units were below 0.5. In the test cases, the separation between the activity levels of the correct units and the activity levels of the remainder were not as sharp as in the training cases.

Any learning procedure which relied on finding direct correlations between the input and output vectors would generalize very badly on the family trees task. Consider the correlations between the filler of the person1 role and the filler of the person2 role. The filler of person1 that is used in each of the generalization tests is negatively correlated with the correct output vector because it never occurred with this output vector during training, and it did occur with other output vectors. The structure that must be discovered in order to generalize correctly is not present in the pairwise correlations between input units and output units.

4.6. Componential versus structuralist accounts of concepts

The family trees example sheds an interesting new light on a long-running controversy between rival theories of conceptual structure. There have been many different proposals for how conceptual information may be represented in neural networks. These range from extreme localist theories in which each concept is represented by a single neural unit [1] to extreme distributed theories in which a concept corresponds to a pattern of activity over a large part of the cortex [19]. These two extremes are the natural implementations of two different theories of semantics. In the structuralist approach, concepts are defined by their relationships to other concepts rather than by some internal essence. The natural expression of this approach in a neural net is to make each concept be a single unit with no internal structure and to use the connections between units to encode the relationships between concepts. In the componential approach each concept is simply a set of features and so a neural net can be made to implement a set of concepts by assigning a unit to each feature and setting the strengths of the connections between units so that each concept corresponds to a stable pattern of activity distributed over the whole network [15, 17, 33]. The network can then perform concept completion (i.e. retrieve the whole concept from a sufficient subset of its features). The problem with most componential theories is that they have little to say about how concepts are used for structured reasoning. They are primarily concerned with the similarities between concepts or with pairwise associations. They provide no obvious way of representing articulated structures composed of a number of concepts playing different roles within the structure.

The family trees example shows that componential reduced descriptions can be learned from structural information about how concepts go together within propositions. Given a sufficiently powerful learning procedure, the structuralist information can be converted into componential representations that facilitate rapid intuitive inference.

5. The Fixed Mapping

Perhaps the most obvious way to implement part–whole hierarchies in a connectionist network is to use the connections themselves as pointers. The simplest version of this uses localist representations, but once that version has been understood, it is easily converted into a version that uses distributed representations. Figure 6 shows a localist example taken from the work of McClelland and Rumelhart [21]. It is a network that recognizes a word when

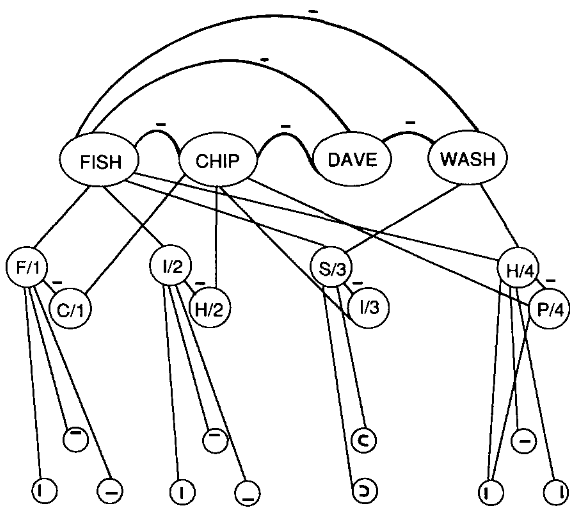

Fig. 6. Part of a network used for recognizing words. Only a few of the units and connections are shown. The connections between alternative hypotheses at the same level are inhibitory.

given partial information about the features of the letters in the word.[7] Because each relationship in the hierarchical tree-structure is implemented by its own dedicated connection, it is possible to do a lot of parallel processing during recognition. Simultaneously, many different letters can check whether the features they require are present, and many different words can check whether the letters they require are present.

One very important aspect of the McClelland and Rumelhart network is that each of the letter units has to be replicated for each possible position of a letter within the word. There are separate units for an H as first letter and an H as second letter. All the letter features and all the knowledge about which combinations of letter features make an H must also be replicated for each of the positions. This replication is a natural consequence of implementing part–whole relationships with pairwise connections. A part–whole relationship involves *three* different things: The part, the whole, and the role that the part plays within the whole. In the conventional implementation using pointers, the role is encoded by which field the pointer is in. A pairwise connection between neuron-like units does not have anything equivalent to a field, and so one of the two units is used to represent *both* the field and the contents of the field. Thus, instead of having a single role-independent representation of H which is pointed to from many different fields, we have many different "role-specific" representations. Activity in any one of these units then represents the *conjunction* of an identity and a role.

At first sight, the fixed mapping seems very wasteful because it replicates the apparatus for representing and recognizing letters across all the different roles. However, the replication has some useful consequences. It makes it possible to recognize different instances of the same letter in parallel without any of the contention that would occur if several different processes needed to access a single, central store of knowledge simultaneously. Also, when letters are used as cues for words, it is not just the letter identities that are important. It is the conjunction of the identity and the spatial role within the word that is the real cue. So it is very convenient to have units that explicitly represent such conjunctions.

5.1. The fixed mapping with distributed representations

The McClelland and Rumelhart network uses localist representations in which each entity is represented by activity in a single unit. Localist representations are efficient if a significant fraction of the possible entities are present on any one occasion or if the knowledge associated with each entity has little in common with the knowledge associated with other, alternative entities. Both

[7] We use this as the standard, concrete example of a part–whole hierarchy because it has clearly defined levels and the parts have convenient names, but this paper is not about word recognition.

these conditions hold quite well at the level of letter recognition. For the more natural part–whole hierarchies that occur in everyday scenes, neither condition holds. Only a tiny fraction of the possible objects are present on any one occasion, so if one unit is devoted to each possible object almost all the units will be inactive. This is a very inefficient way to use the representational capacity. Also, different objects, like a cup and a mug, may have similar appearances and may make similar predictions. This means that there can be a lot of useful sharing of units and connections. Most of what we know about cups and mugs could be associated with a unit that is active for either a cup or a mug. If this method of sharing is taken to its logical conclusion we arrive at distributed representations in which each object is represented by activity in many units and each unit is involved in the representation of many objects [13].

One way of viewing a distributed representation is as a *description* of an object in terms of a set of primitive descriptors. The description denotes the intersection of the sets of objects denoted by each of the individual descriptors. It is natural to assume that the individual units correspond to fixed primitive descriptors, but a more interesting possibility is that the primitive descriptors continually change their meanings. Each time a new object is encountered, many connection strengths change slightly and this causes slight changes in the circumstances in which each unit becomes active and in the effect that its activity has on the rest of the network. In the short term, the meanings of the primitive descriptors are fairly stable, but over a longer time scale they shift around and move towards a vocabulary that makes it easy to express the structure of the network's environment in the connection strengths [10].

One major advantage of using descriptions rather than single units as representations is that it is possible to create representations of novel objects (and also novel role-specific representations) by using novel combinations of the same set of primitive descriptors. This avoids the problem of having to find a suitably connected unit for each novel object.

5.2. Sharing units between similar roles

So far, we have assumed that each role within a structure has its own dedicated set of "role-specific" units. Each of these units may use "coarse-coding" in the space of possible identities for the role-filler, but it is entirely specific about the role. This way of localising the roles is reasonable if there are only a few possible roles (such as the four letter positions in the McClelland and Rumelhart model), but it has difficulty dealing with structures that have a large or indefinite number of potential roles. If, for example, we consider the semantic cases in English sentences, it is very hard to decide how many cases there really are, and it is also clear that some cases are very similar to others. In the sentences "Mary beat John at tennis" and "Mary helped John with his algebra" it is clear that tennis and algebra occupy similar but not quite identical semantic roles. A natural way to handle this phenomenon is to use conjunctive units that are coarse-coded in role-space as well as in identity space [9]. An instantiated structure then consists of a set of activations in these units, and each binding of an identity to a role is encoded by many of these coarse-coded units. Naturally, the representation becomes ambiguous if we simultaneously encode many bindings of similar identities to similar roles, but people also fail in these circumstances. Smolensky (this issue) gives a formal treatment of this type of representation using the formalism of tensor products.[8]

5.3. Disadvantages of the fixed mapping approach

The major problems of the fixed mapping approach are:

(1) Extra hardware is required to replicate the recognition apparatus across all roles.
(2) Extra learning time is required to train all the separate replicas of the same recognition apparatus.
(3) The replication raises the issue of how, if at all, different role-specific representations of the same entity are related to one another.
(4) The model presupposes some input apparatus for transforming the retinal image of a word into the primitive features that are used for recognition. As the word changes its position on the retina, the very same set of primitive feature units must remain active. It seems sensible to perform recognition by using a fixed network in which every connection implements a particular piece of knowledge about how things go together in good interpretations, but this can only work if there is a way of correctly mapping the external world into the bottom level units of the fixed network. Much of the difficulty of recognition lies in discovering this mapping.
(5) As we go down the hierarchy, there are fewer and fewer units available for representing each constituent. For relatively shallow, man-made hierarchies of the kind that are important in reading or speech recognition, it may be tolerable to always devote fewer units to representing smaller fragments of the overall structure. But for domains like normal visual scenes this strategy will not work. A room, for example, may contain a wall, and the wall may contain a picture, and the picture may depict a room. We need to be able to devote just as much apparatus to representing the depicted room as the real one. Moreover, the very same knowledge that is applied in recognizing the real room needs to be

[8] I suspect that extreme coarse-coding in role-space is a mistake. In a nonlinear system, it is probably easier to make use of the information about the fillers of roles if this information is localised (as it was in the family-trees example described in Section 4).

applied in recognizing the room in the picture. If this knowledge is in the form of connections and if the knowledge is not duplicated there must be a way of mapping the depicted room into an activity pattern on the very same set of units as are used for representing the real room.

6. Within-level Timesharing

Distributed representations provide a way of sharing units and connections between alternative objects or alternative role-specific representations. In this respect they work just like pointers in a conventional computer memory. Instead of using a separate bit for each possible object that could be pointed to, each bit is shared between many possible alternative objects. As a result, a word of memory can only point to one object at a time.[9] The following analysis of the functions performed by a role-specific representation suggests a quite different and complementary method of sharing which can be used to share connectionist apparatus between the different role-specific instances that occur within one whole.

In the McClelland and Rumelhart model each role-specific letter unit has three functions:

(1) It recognizes the occurrence of that letter in that spatial role. The recognition is accomplished by having appropriately weighted connections coming from units at the feature level.
(2) It contributes to the recognition of words. This is accomplished by its connections to units at the word level.
(3) Its activity level stores the results of letter recognition.

There is an alternative way of mapping the part–whole hierarchy into a connectionist network that uses role-specific letter units for functions (2) and (3), but not for function (1). Instead, the alternative method uses a single letter-recognition module which is applied to one position within the word at a time. Once the letter at the current position has been recognized, the combination of its identity and its position within the word activates a role-specific letter unit which acts as a temporary memory for the results of the recognition and also contributes to the recognition of the word (see Fig. 7.) This method is called "within-level" sharing because a single recognition module is shared across the entities within one level.

The letter-recognition module must be applied to one letter at a time and so there must be extra "attentional" apparatus that selects out one portion of the parallel input (which contains features of all the letters), maps this portion into

[9] Some ancient implementations of LISP actually use two separate role-specific representations within one word so that the first part of a word can point to one object and the second part can point to another.

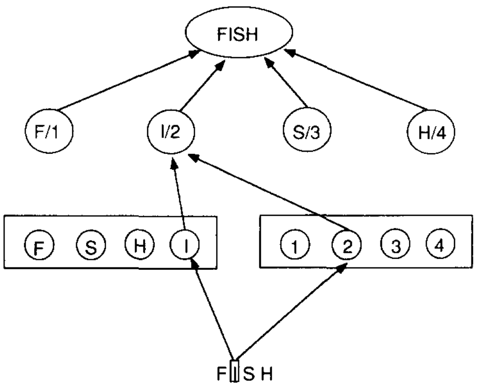

Fig. 7. Some of the apparatus required to store the sequence of outputs of a single, sequential letter-recognition module in order to recognize a word. The network is "attending" to the second letter of the word. Notice that the role-specific units do not need to be able to recognize letters. The apparatus required for mapping the appropriate part of the input into the letter recognition module is not shown.

the input of the letter-recognition module, and also creates an explicit representation of where the currently selected letter lies within the word. Actually, the McClelland and Rumelhart model implicitly presupposes that there is apparatus of a similar kind in order to pick out the features of one word within a sentence or to cope with changes in the position of a word. So the new model does not require any qualitatively new attentional apparatus, it merely requires it at the level of letters instead of at the level of words. Also, it makes explicit the requirement for attentional apparatus, storage apparatus, and control apparatus.

7. Between-level Timesharing

There is one limitation of within-level sharing that is unimportant in the domain of reading but is very important in most other domains where the same knowledge can be applied at many different levels. For reading, the knowledge is quite different at each level: Knowledge about the shape of a letter is quite different from knowledge about which sequences of letters make words, so there is little point in trying to use the same set of connections to encode both kinds of knowledge. In most natural domains, however, wholes and their parts have much in common. One example has already been given in which a room contains a picture that depicts a room. Another example is the sentence "Bill was annoyed that John disliked Mary." One of the constituents of this sentence, "John disliked Mary," has a lot in common with the whole sentence. The same kind of knowledge is needed for understanding the constituent as is needed for understanding the whole. This is also typical of natural visual scenes which generally have just as much richness at every level of detail.

To share the knowledge in the connections between different levels in the part–whole hierarchy, it is necessary to use flexible mappings between the entities in the part–whole hierarchy and the groups of units in the connectionist network. The hardware is viewed as a window that can be moved up and down (in discrete steps) over the part–whole hierarchy (see Fig. 8). One node in the hierarchy is chosen as the current whole and all of the units in the main network are then devoted to recognizing and representing this whole. Some units are used for describing the global properties of the whole, and others are used for role-specific descriptions of the major constituents of the whole. The entire pattern of activity will be called the "Gestalt" for the current whole.

The crucial property of the moveable window scheme is that the pattern of activity that represents the current whole is totally different from the pattern of activity that represents the very same object when it is viewed as being a constituent of some other whole.[10] In one case the representation of the object and its parts occupies all of the main network and in the other case it is a role-specific description that occupies only the units devoted to that role.

The idea that the very same object can be represented in different ways depending on the focus of attention is a radical proposal which violates the very natural assumption that each entity in the world has a unique corresponding representation in the network. Violating this assumption leads to so many obvious difficulties, that few researchers have considered it seriously, but these problems must be faced if we are to capture between-level regularities in a connectionist network. The problems include:

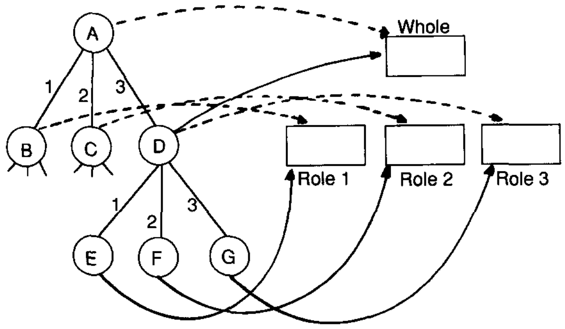

Fig. 8. The solid and dashed lines show two different ways of mapping a part–whole hierarchy (on the left) into the same connectionist hardware (on the right). Notice that node *D* in the hierarchy can be represented by two totally different activity patterns that have nothing in common.

[10] Charniak and Santos [3] describe a connectionist parser in which the hardware acts as a window that slides over the parse-tree, but they do not use the idea of reduced descriptions: To move the hardware window up the hierarchy they simply copy the representations from one hardware level to the next level down.

(1) When the mapping between the world and the network is changed in such a way that one constituent of the previous whole becomes the new focus of attention, what kind of internal operations are required to convert the previous, role-specific description of that constituent into a full description that occupies the whole of the main network?
(2) How is temporary information about recent Gestalts stored so that the network can return to them later? The information cannot be stored as the activity pattern that the network settles to when the Gestalt is created because the very same network is needed for creating the next Gestalt.
(3) How is the next mapping chosen? Is the choice made by a separate parallel computation, or can it be part of the same computation that is involved in forming the Gestalt? This is not a problem for the fixed mapping approach because all the parts of the hierarchy are represented simultaneously in a single large network.

The following subsections address these issues.

7.1. Moving up and down the part–whole hierarchy

Figure 9 shows some of the extra apparatus that might be required to allow a connectionist network to move down the part–whole hierarchy by expanding a role-specific, reduced description into a full description of the role-filler. This corresponds to following a pointer in a conventional implementation. Notice that it is a slow and cumbersome process. Moving back up the hierarchy is even more difficult. First, the full description of a part must be used to create the appropriate role-specific, reduced description of that part. This involves using the apparatus of Fig. 9 in the reverse direction. Then the role-specific, reduced description must be used to recreate the earlier full description of which it is a constituent. If the hierarchical structure is highly overlearned, it is possible to train a network to recover the full description [23], but if the hierarchy is a novel one, the only way to move back up it without additional external guidance is to use some kind of content-addressable working memory for earlier Gestalts. Traversing the part–whole hierarchy can be made simpler by using reduced descriptions whose microfeatures are systematically related to the microfeatures of the corresponding full descriptions. Even though the reduced and full descriptions correspond to quite different patterns of activity, it is much easier to generate one from the other if the patterns of activity are related in a non-arbitrary way.

The obvious way to implement the working memory required for recovering earlier Gestalts is to set aside a separate group of "working memory" units. If it is only necessary to remember one Gestalt at a time, this group can simply contain a copy of the pattern of activity in the network where Gestalts are formed. If several Gestalts need to be remembered at a time, several different

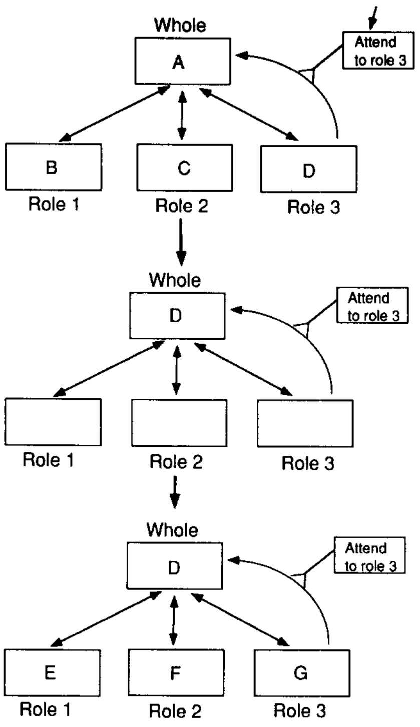

Fig. 9. One way of using some additional hardware to allow the network to access the full description of node *D* from a role-specific reduced description.

groups could be used. Alternatively, a single group could be used provided that the various patterns of activity that need to be stored are first recoded in such a way that they can be superimposed without confusing them with one another. Examples of such encodings are described in [29, 32]. Touretzky [30, 31] shows how this kind of working memory can be used to traverse and transform tree structures.

An interesting alternative implementation of working memory uses temporary modifications of the connection strengths in the network that is used for creating the Gestalt. Each internal connection in this network can be given two different weights: A long-term weight which changes relatively slowly and a short-term weight which is limited in magnitude, changes rapidly, and spontaneously decays towards zero. The effective connection strength at any time is simply the sum of the short-term and long-term weights. The long-term weights encode knowledge about which patterns of activity constitute good interpretations of the input to the network (i.e. familiar or plausible Gestalts). The short-term weights act as a contextual overlay[11] that encodes information about which patterns of activity occurred recently. If the network receives a rich external input which is incompatible with recently occurring Gestalts, it will settle to a new Gestalt and the short-term weights will act as noise (to which these networks are very resistant). If, however, parts of the external input are missing and the remainder fits some recently occurring Gestalt, the short-term weights will favor this Gestalt over other alternative Gestalts which would fit the partial input just as well if the short-term weights were not considered. So the short-term weights will implement a content-addressable memory for recent Gestalts.

Some unpublished simulations I performed in 1973 showed that short-term weights could be used to allow a network to return to a partially completed higher-level procedure after using between-level sharing to execute a recursive call of the same procedure in the same hardware. In these simulations desired states were specified for all the units at all times so there were no hidden units and a variation of the perceptron convergence procedure could be used to learn the appropriate long-term weights. The short-term weights were adapted by a simple local rule that combined a decay term with a term that increased the weight between two unit-states that occurred in close temporal proximity.

With the advent of the backpropagation learning procedure, it is now possible to use backpropagation through time to adapt the long-term weights of hidden units in such a way that a recurrent network can learn for itself how to store and retrieve temporary information in the short-term weights. This modification of the backpropagation procedure is rather complex, since it requires the backpropagated derivatives to take into account indirect effects of the following form: A change in a long-term weight at time t causes activity levels to change at time $t+1$ which causes short-term weights to change at time $t+2$ which causes activity levels to change at all subsequent times. A much simpler alternative is to ignore these complicated components of the derivatives and to rely on the fact that a simple hebbian rule for incrementing the short-term weights will typically cause these weights to store the information that is required a short time later. This simplification is similar to the simplification often used in recurrent nets in which derivatives are not back-propagated through time in a recurrent network. Instead, it is assumed that the hidden units will fortuitously encode the historical information that is subsequently required for performing the task [16].

7.2. Choosing mappings

Decisions about which parts of the world or the task should be mapped into

[11] A very different use of this contextual overlay is described in [14]. It can be used to approximately cancel out recent changes in the long-term weights, thus allowing earlier memories to be "deblurred."

which parts of the connectionist hardware are clearly of central importance and would form a major part of any account of how a particular network performed a particular rational inference. The purpose of this subsection is simply to point out that there is an interesting spectrum of possible ways in which a network might focus its attention on a particular piece of the domain. At one end of the spectrum, a separate "executive" module would select the current mapping without using any information about the representations that were then generated using that mapping. The choice of mapping would be based on previously generated representations, but not on the representation generated using the current mapping. At the other end of the spectrum, the desired mapping could be defined in terms of the representations it produced, so that the network would have to simultaneously settle on the mapping and the representation. Although this search might involve many iterations, it would not be sequential in the sense in which rational inference is sequential: It would not involve a sequence of *settlings*. So in the description of the rational inference, the network would just intuitively choose the appropriate mappings to generate the required representations at each step.

A working example of this way of choosing mappings is described in [12]. A network that is trying to locate a letter in an image that contains several letters can activate the desired shape representation and use this to select the correct mapping of the image onto the recognition apparatus.

7.3. An example of between-level sharing

So far, the discussion of between-level sharing has been rather abstract, mainly because there is no working implementation that properly illustrates the approach. Even without an implementation, however, the ideas may be clearer in the context of a specific task.

Consider a network that is trying to arrive at a plan of action that satisfies several goals such as arriving home on time with the TV guide, the wine and the pizza. Given enough units, it is possible to design a network in which all the possible choices and all the constraints between them can be simultaneously represented. Then, if the activities of some units are clamped to represent the satisfied goals, a single settling of this whole network can arrive at a set of choices that are consistent and that satisfy the goals. Given fewer units, however, this fixed mapping approach is not feasible and it is necessary to solve the overall task one piece at a time.

If there were no interactions between the subtasks, the serial computation would be relatively simple, and the connectionist implementation would be relatively uninteresting. But to arrive home on time, it may be necessary to choose a single store that sells several of the desired items, or a set of stores that are close to one another. Instead of solving the subtasks one at a time, it may be better to first settle on a rough overall plan, and to leave the details until later. The rough overall plan may contain reduced descriptions of the way each subtask will be solved. These reduced descriptions must have sufficient detail to ensure that the solutions to the subtasks fit together properly, but they do not require the full details of the solution. Of course, when the network later focusses its attention on a subtask, it may turn out to be impossible to fill in the details in a way that is consistent with the assumed solutions of the other subtasks, in which case a new overall rough plan will have to be formulated.

By sequentially re-using the same hardware for settling on the detailed solutions to the subtasks, the network can share the knowledge in the connections between these subtasks. However, the need to go back and reformulate the overall plan when its parts cannot be implemented consistently illustrates the major drawback of this kind of sharing: If the constraints are not all satisfied at the same time, it may be necessary to change the focus of attention many times before arriving at a solution which is consistent at all levels. For a network to avoid this kind of sequential "thrashing" it must use reduced descriptions of the subtasks that are compact enough to allow the global constraint satisfaction, but detailed enough to ensure that the parts of the global solution are locally feasible. For genuine puzzles, this may not be possible, but for normal common sense reasoning where there are many possible solutions it should be possible to learn appropriate reduced descriptions that allow large but friendly real-world constraint satisfaction problems to be hierarchically partitioned into manageable pieces. Whether this can actually be done remains to be seen.

7.4. Capturing regularities versus parallel recognition

The use of between-level sharing allows a connectionist network to capture certain kinds of regularity, but only by resorting to serial processing, and the loss of parallel processing seems like a very high price to pay. Connectionists would like their models to have all three of the following properties:

(1) All the long-term knowledge should be in the connection strengths.
(2) The network should capture the important regularities of the domain.
(3) Rapid recognition should be achieved by recognizing the major parts of a whole in parallel.

Unfortunately, it is very hard to achieve all three properties at once. If the knowledge is in the connections, and if the same knowledge is required to recognize several different parts of the same object (e.g. the letters in the word "banana") it seems as if we have to make an unpalatable choice. We can either capture the regularities in the appearances of different tokens of the same letter by sequentially re-using the very same connections to recognize the different instances of the letter "a," or we can achieve parallel recognition by replicating "a" recognizers across all the different positions in the word. McClelland [20] describes a clever but inefficient way out of this dilemma:

There is a single canonical representation of the knowledge whose weights are copied *during recognition* to produce the required parallel recognizers. An alternative way out of the dilemma is to use the learning process to transfer the knowledge from a canonical representation to a set of role-specific recognizers. The knowledge transfer is much slower than in McClelland's scheme, but it requires far fewer connections because weights are not transferred explicitly, and once the learning has been done, no transfer is required during recognition. This idea has strong similarities to a method that has been used by linguists to escape from the same dilemma. Some regularities of English that appear to require a context-sensitive grammar can actually be captured by generating sets of specific context-free rules from some underlying context-free meta-rules [7]. The whole system is context-free, and although there is great redundancy among sets of specific context-free rules, the regularity is nevertheless captured by the fact that the members of each set were all generated from the same underlying meta-rule.

Figure 7 suggests one way of using learning to transfer knowledge. The serial recognizers can be used to train the parallel, role-specific recognizers. A network that timeshares a serial recognizer already requires role-specific units for storing its successive outputs. If these role-specific units have some connections to the perceptual input, these connections can be trained by assuming that the outcome of the serial recognition is the desired outcome of the parallel recognition. The canonical, timeshared representation of the knowledge would then be acting as an internal local teacher for the parallel recognition hardware. This resembles the way in which the outcome of a serial mini-max tree search can be used to evaluate a position and hence to specify the desired outcome of a "parallel" board evaluation [25]. In the language of Section 3, rational inference can be used to train intuitive inference.

The usefulness of this method of transferring knowledge obviously depends on how much faster the learning is with an internal local teacher. This, in turn, depends on whether the canonical representations and the role-specific representations are distributed. If the representations are distributed, and if there is a non-arbitrary mapping between the canonical and the role-specific representations of the same entity, then the role-specific representations can be learned *much* faster because the canonical representations provide detailed information about what pattern of activity to use. This is equivalent to specifying desired values for the hidden units of a network. Connectionist learning is slow when it has to construct hidden representations and it is much faster when all units have their desired values specified.

8. Are These Just Implementation Details?

Proponents of the "classical" symbol processing approach to cognition have argued that many of the issues discussed in this paper are mere implementation details [6]. My view of this claim is best expressed by an analogy. Consider the theory that all locomotion requires wheels. A proponent of this theory could reasonably argue that people have wheels. Since people move over rough ground, they need very large diameter wheels with very springy suspension. They achieve this by implementing one spoke and one piece of rim for each wheel and then timesharing it. The only problem with this argument is that many of the "obvious" properties of wheels that we normally take for granted do not hold in this biological implementation. The timesharing removes the need for a fully revolving bearing, but it requires repeated accelerations and decelerations that make the system energetically inefficient at high speeds. Also, the timeshared spokes can be used in many other ways that do not at all resemble uses of normal wheels. If we insist on viewing connectionist networks as just a novel implementation of classical symbol processing, we should expect many of the standard intuitions we derive from classical systems to be equally misleading.

One major contribution of the connectionist approach is that it focusses attention on the distinction between rational and intuitive inference. Classical symbol processing takes rational inference as its paradigm. On a serial machine it is easy to program the kind of deliberate reasoning that occurs when a person sorts a list of numbers into ascending order. This is no coincidence: serial machines were designed to model this kind of deliberate reasoning, and so they are easy to program whenever a task can be reduced to straightforward deliberate reasoning. From the connectionist perspective, this is unsurprising and provides no justification for the idea that the kinds of inference which we make fast and intuitively should also be modeled by the same kind of symbol processing. The idea that fast intuitive inference is similar in form to conscious inference, but without the consciousness, seems to me to be a major psychological error. So long as we only consider serial machines, however, there is not much motivation for using a quite different model for fast intuitive inference, because the motivation comes from considerations about what kind of processing can be done fast in a connectionist network.[12]

A major weakness of the connectionist systems that have been sketched out in this paper is that they have great difficulty handling quantifiers and variable binding. A primitive scheme for variable binding is described in [32], but connectionist networks would be more convincing if they could rapidly construct complex, plausible scenarios in which many instantiated schemas fit together appropriately. Shastri [26] has recently described one way of doing this using localist representations. Each schema has its own dedicated hardware including dedicated hardware for representing the fillers of its slots. Shastri uses temporal phase as a temporary symbol for a slot-filler, but the same basic

[12] Tractability may also provide a motivation for some limited form of inference, but it is not clear why the same tractability argument should not also be applied to rational inference.

scheme could be implemented using reduced descriptions. This may allow fast, parallel, intuitive inference to do much more than simply completing the instantiation of a single schema.[13]

9. Summary of the Three Types of Mapping

If we consider how to map a part–whole hierarchy into a finite amount of parallel hardware there are three broad approaches: Fixed mappings, within-level timesharing, and between-level timesharing. These three approaches can be distinguished by abstract properties of the mappings involved:

(1) The fixed mapping uses a one-to-one mapping. Each object in the part–whole hierarchy is always mapped into a pattern of activity in the same set of units, and each set of units is always used to represent the same object.
(2) Within-level sharing uses a many-to-one mapping. Many different objects at the same level can be mapped into the same set of units in the serial recognition apparatus. But whenever one of these objects is recognized, it is represented in the same units.
(3) Between-level sharing uses a many-to-many mapping. It allows many different objects at the same level to be mapped into the same set of units, but it also allows the same object to be mapped into different sets of units depending on the level at which attention is focussed.

10. Conclusions

An important consequence of using role-specific reduced descriptions to implement pointers is that the relationship between a pointer and the thing it points to is not arbitrary. So computations, such as inferring the filler of one role when given the fillers of other roles, that would normally require a lot of sequential pointer-chasing, can be performed without accessing the full descriptions of the fillers. This means that the process of using a reduced description to gain access to the corresponding full description can be a relatively rare and slow event.

The combination of slow sequential access to full descriptions and fast parallel constraint-satisfaction using reduced descriptions is a style of computation that is well-suited to neural networks. The parallel inner loop allows the network to perform a great deal of computation by settling into a state that

[13] In [9] I rejected the idea of using separate hardware for each schema because I thought this would make it impossible to share common information between schemas. Now I can see a way out of the dilemma: For parallel inference, we give each schema its own hardware, but to capture the regularities among different schemas we have additional hardware for the fully-articulated representation of a schema. In this additional hardware, different schemas correspond to alternative activity patterns.

satisfies constraints that are encoded in the connections. So if the appropriate representations are known, a lot of useful computation can be done by a single intuitive inference. More elaborate computations which cannot be performed in a single settling using the current representations are performed by a sequence of settlings, and after each settling the mapping between the world and the network is changed. The ability to change the mapping is what allows the knowledge in any particular set of connections to be brought to bear on any part of the task, and thus provides the great flexibility that is characteristic of rational human thought.

ACKNOWLEDGEMENT

I thank Eugene Charniak, Jeffrey Elman, and David Plaut for helpful comments on an earlier draft. Many of the ideas described here were influenced by members of the PDP Research Group at San Diego, the Boltzmann group at Carnegie-Mellon and the Connectionist Research Group at Toronto.

REFERENCES

1. H.B. Barlow, Single units and sensation: A neuron doctrine for perceptual psychology? *Perception* **1** (1972) 371–394.
2. H.J. Berliner and D.H. Ackley, The QBKG system: Generating explanations from a non-discrete knowledge representation, in: *Proceedings AAAI-82*, Pittsburgh, PA (1982) 213–216.
3. E. Charniak and E. Santos, A connectionist context-free parser which is not context-free, but then it is not really connectionist either, in: *Proceedings Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA (1987).
4. M. Derthick, Counterfactual reasoning with direct models, in: *Proceedings AAAI-87*, Seattle, WA (1987).
5. S.E. Fahlman, *NETL: A System for Representing and Using Real-world Knowledge* (MIT Press, Cambridge, MA, 1979).
6. J.A. Fodor and Z.W. Pylyshyn, Connectionism and cognitive architecture: A critical analysis, *Cognition* **28** (1988) 3–71.
7. G. Gazdar, Phrase structure grammar, in: P. Jacobson and G.K. Pullum, eds., *The Nature of Syntactic Representation* (Reidel, Dordrecht, 1982) 131–186.
8. W.D. Hillis, *The Connection Machine* (MIT Press, Cambridge, MA, 1985).
9. G.E. Hinton, Implementing semantic networks in parallel hardware, in: G.E. Hinton and J.A. Anderson, eds., *Parallel Models of Associative Memory* (Erlbaum, Hillsdale, NJ, 1981).
10. G.E. Hinton, Learning distributed representations of concepts, in: *Proceedings Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA (1986).
11. G.E. Hinton, Deterministic Boltzmann learning performs steepest descent in weight-space, *Neural Comput.* **1** (1989) 143–150.
12. G.E. Hinton and K.J. Lang, Shape recognition and illusory conjunctions, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985).
13. G.E. Hinton, J.L. McClelland and D.E. Rumelhart, Distributed representations, in: D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **1**: *Foundations* (MIT Press, Cambridge, MA, 1986).
14. G.E. Hinton and D.C. Plaut, Using fast weights to deblur old memories, in: *Proceedings Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA (1987).

15. J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Nat. Acad. Sci. USA* **79** (1982) 2554–2558.
16. M.I. Jordan and D.A. Rosenbaum, Action, in: M.I. Posner, ed., *Foundations of Cognitive Science* (MIT Press, Cambridge, MA, 1989) Chapter 19, 727–767.
17. T. Kohonen, *Associative Memory: A System-theoretical Approach* (Springer, Berlin, 1977).
18. J.E. Laird, P.S. Rosenbloom and A. Newell, Chunking in Soar: The anatomy of a general learning mechanism, *Mach. Learn.* **1** (1986) 11–46.
19. K. Lashley, In search of the engram, in: *Symposia of the Society for Experimental Biology* **4** (Academic Press, New York, 1950).
20. J.L. McClelland, The programmable blackboard model of reading, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **2**: *Applications* (MIT Press, Cambridge, MA, 1986).
21. J.L. McClelland and D.E. Rumelhart, An interactive activation model of context effects in letter perception, Part 1: An account of basic findings, *Psychol. Rev.* **88** (1981) 375–407.
22. F.J. Pineda, Generalization of back propagation to recurrent and higher order neural networks, in: *Proceedings IEEE Conference on Neural Information Processing Systems*, Denver, CO (1987).
23. J.B. Pollack, Recursive distributed representations, *Artificial Intelligence* **46** (1990) 77–107, this issue.
24. D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by back-propagating errors, *Nature* **323** (1986) 533–536.
25. A.L. Samuel, Some studies in machine learning using the game of checkers, in: E.A. Feigenbaum and J. Feldman, eds., *Computers and Thought* (McGraw-Hill, New York, 1963) 71–105.
26. L. Shastri, A connectionist system for rule based reasoning with multi-place predicates and variables, Tech. Rept. MS-CIS-89-06, Department of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, PA (1989).
27. P. Smolensky, Schema selection and stochastic inference in modular environments, in: *Proceedings AAAI-83*, Washington, DC (1983) 109–113.
28. P. Smolensky, On the proper treatment of connectionism, *Behav. Brain Sci.* **11** (1988) 1–74.
29. P. Smolensky, Tensor product variable binding and the representation of symbolic structures in connectionist systems, *Artificial Intelligence* **46** (1990) 159–216, this issue.
30. D.S. Touretzky, Reconciling connectionism with the recursive nature of stacks and trees, in: *Proceedings Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA (1986).
31. D.S. Touretzky, BoltzCONS: Dynamic symbol structures in a connectionist network, *Artificial Intelligence* **46** (1990) 5–46, this issue.
32. D.S. Touretzky and G.E. Hinton, Symbols among the neurons: Details of a connectionist inference architecture, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985).
33. D.J. Willshaw, O.P. Buneman and H.C. Longuet-Higgins, Nonholographic associative memory, *Nature* **222** (1969) 960–962.