3

Learning distributed representations of concepts

GEOFFREY E. HINTON

Two simple theories of neural representation

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 (Barlow 1972) to extreme distributed theories in which a concept corresponds to a pattern of activity over a large part of the cortex. 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 (Hopfield 1982; Kohonen 1977; Willshaw, Buneman, and Longuet-Higgins 1969). The network can then perform concept completion (i.e. retrieve the whole concept from a sufficient subset of its features). The problem with 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.

Role-specific units

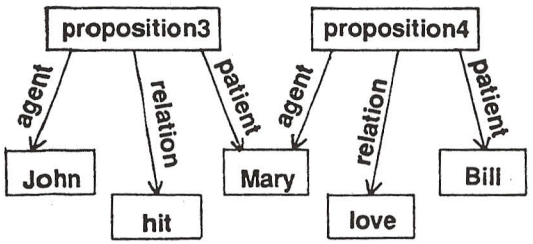
One way of using neural nets to implement articulated structures of the kind shown in the semantic net formalism in Fig. 3.1(a) is to assign a group of neural units to each possible role and to make the pattern of activity of the units in

that group represent the concept that fills the role (Hinton 1981). Each unit then represents the conjunction of a role with a feature of the concept filling that role (e.g. a unit might be active *iff* the agent is male). A proposition can then be represented by a stable combination of role fillers as shown in Fig. 3.1(b). This is a fundamentally different method of representation than either of the two more obvious methods described above. It has the interesting property that the very same concept will have quite different representations when it is playing different roles. The use of multiple different representations of the same concept appears to be a serious flaw for two reasons. First, it appears to be uneconomical. Second, it is not clear how we know that John in the representation of 'John hit Mary' has anything to do with the John in the representation of 'Mary divorced John'.

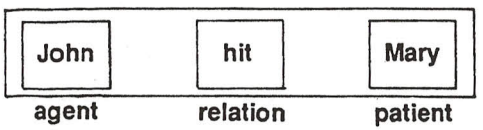
The economic considerations are complex. The 'obvious' way to represent that John is the agent of a certain proposition is to combine a canonical representation of John with a canonical representation of agent. In Lisp, for example, a symbolic expression like (agent john) would be an obvious representation and a whole proposition might be represented by the expression ((agent john) (relation hit) (patient mary)). Alternatively, the roles might be implicitly represented by the position of an element in a list and so the whole proposition could be represented by (hit john mary). Either way, the very same symbol is used for John whatever role he plays in the proposition. In logic and in computer programming, the standard way of representing conjunctions is by composing symbolic expressions out of individual symbols. Given a conventional general purpose computer memory, it is easy to store arbitrary compositions of symbols.

If, however, we want to be able to retrieve a proposition from a partial description of its contents, the advantage of always using the very same representation for John is less clear. If the partial description includes the information that John is the agent, we would like this to pick out just those propositions which have John as agent. This is more specific than the propositions which have an agent and also have John in some role. It is the conjunction of John with agent that forms the retrieval cue, and so in a neural implementation it would make sense to have a specific representation for this conjunction. This conjunctive representation can then cause the effects required for completing the whole pattern of activity that represents the proposition. So, even if there is a representation in which John and agent are represented separately, it may be necessary to form a conjunctive representation for retrieval.

A similar argument can be made in other domains. In representing the graphemic structure of the word 'chip', for example, it would be possible to use a representation such as ((4 p) (2 h) (3 i) (1 c)) but for a task like filling in the blank in 'c-ip' it is inefficient to access all words which contain c and i and p. The identities and roles of the letters need to be conjoined to form more



(a)



(b)

Fig. 3.1 (a) Part of a semantic net. (b) Three groups of units which have two *alternative* stable states that represent the two propositions in the semantic net.

specific access cues. This argues that we need quite different neural representations of (1 c) and (4 c). For the purposes of access to the whole word, these representations need have nothing in common.[1] Indeed, parallel network models of reading use separate representations of letters in different positions within the word (McClelland and Rumelhart 1981).

The second problem with role-specific representations is how to recognize the identity of the various different role-specific representations of the same concept. An efficient way to do this is to have a single, canonical representation for each concept and to have a mechanism for translating between role-specific representations and the canonical one. Hinton (1981) shows how this idea can be implemented in a neural net. It will not be discussed further in this paper.

Choosing the role-specific representations

From now on, we assume that a concept playing a role within a larger structure is represented by a pattern of activity in a group of role-specific units, and we focus on the issue of how this pattern should be chosen. A simple solution is to use patterns selected at random, perhaps with the additional constraint that no two patterns are too similar. The use of random patterns is quite common in research in this area (Hopfield 1982; Willshaw 1981). It makes analysis easier and it is a sensible default in the absence of any other ideas about representation. However, it entirely eliminates one of the most interesting aspects of distributed representations: the ability to capture the similarity between concepts by the similarity of their representations, and the

consequent ability to generalize new information in sensible ways. We illustrate this point in the simple domain of family relationships.

Figure 3.2 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

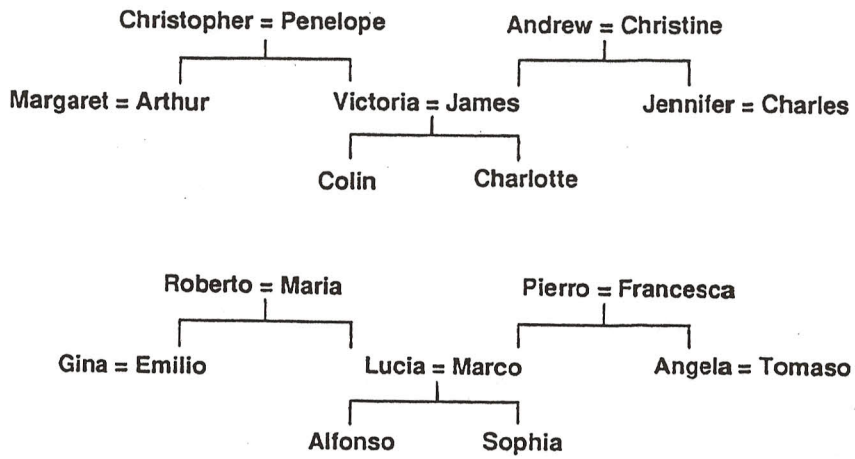


Fig. 3.2 Two isomorphic family trees. The symbol '=' means 'married to'.

role **relationship** and a group for the role **person2**. The net may also require further groups of units in order to achieve the correct interactions between the three role-specific groups. Figure 3.3 shows a network in which one further

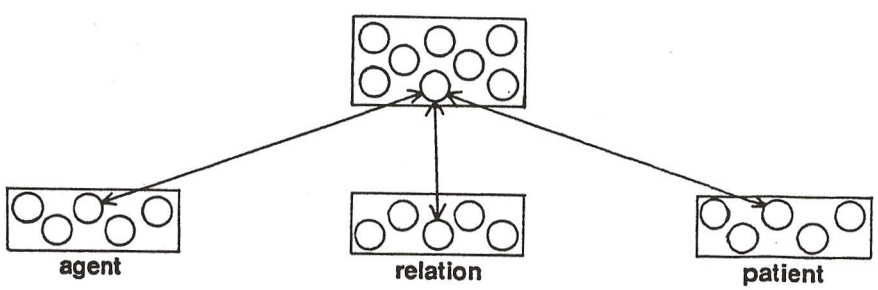


Fig. 3.3 An extra group of units can be used to implement higher-order constraints between the role-specific patterns.

group 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 microinference. 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 microinference that is applicable in many different propositions, the network makes better use of the information carrying capacity of its units than if it dedicated a single extra unit to each proposition. This is an example of the technique of coarse coding described in Hinton, McClelland, and Rumelhart (1986). In describing how a microinference 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.

Microinferences 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 microinference 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 microinferences 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 microinferences 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.[2]

Giving the network the freedom to choose representations

The network shown in Fig. 3.3 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 which is neutral with respect to the various possible internal representations. Figure 3.4 shows how this can be done. The network

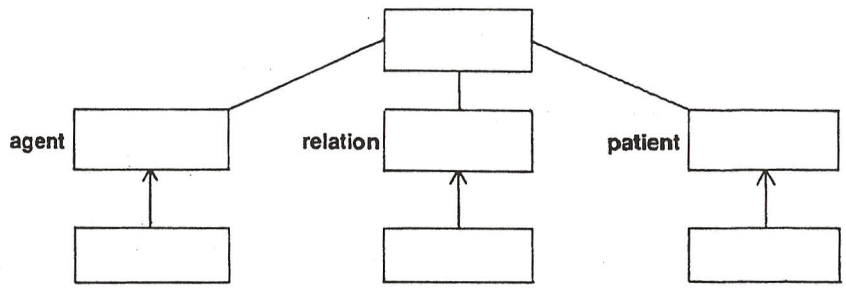


Fig. 3.4 The state of each role-specific group can be fixed via a special input group. By varying the weights between the special input groups and the role-specific groups the network can develop its own role-specific representations instead of being forced to use representations that are pre-determined.

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 similar.[3]

A network that learns distributed representations

In our attempts to show that neural networks can learn sensible distributed representations we have tried several different learning procedures. The most successful of these, so far, is the 'back-propagation' procedure described in Rumelhart, Hinton, and Williams (1986), and the simulation we present uses back-propagation. This learning procedure, which is briefly outlined in the following section, assumes that the units have real-valued outputs between 0 and 1 and which are deterministic functions of their total inputs, where the total input, x_j, to unit j is given by

$$x_j = \sum_i y_i w_{ji} \tag{3.1}$$

A unit has a real-valued output, y_j, which is a non-linear function of its total input.

$$y_j = \frac{1}{1+e^{-x_j}} \tag{3.2}$$

The units are arranged in layers with a layer of input units at the bottom, any number of intermediate layers, and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden: all connections go from lower layers to higher ones.

An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying Equations (3.1) and (3.2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

To use the back-propagation learning procedure we need to express the task of learning about family relationships in a form suitable for a layered network.[4] 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. 3.5. It has a group of input units for the

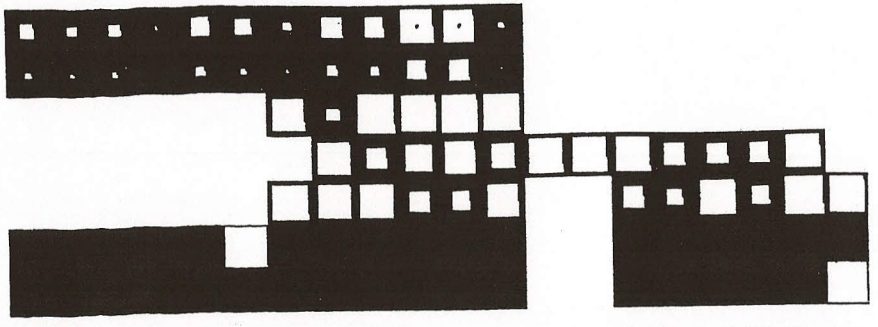


Fig. 3.5 The activity levels in a five-layer network after it has learned. The bottom layer has 24 input units on the left for representing **person1** and 12 units on the right for representing the relationship. The white squares inside these two groups show the activity levels of the units. There is one active unit in the first group (representing Colin) and one in the second group (representing has-aunt). Each of the two groups of input units is totally connected to its own group of 6 units in the second layer. These two groups of 6 must learn to encode the input terms as distributed patterns of activity. The second layer is totally connected to the central layer of 12 units, and this layer is connected to the penultimate layer of 6 units. The activity in the penultimate layer must activate the correct output units, each of which stands for a particular **person2**. In this case, there are two correct answers (marked by black dots) because Colin has two aunts. Both the input and output units are laid out spatially with the English people in one row and the isomorphic Italians immediately below.

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 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, such as uncle, there may be several possible fillers for the **person2** role which are compatible with a given filler of the **person1** role. In a stochastic network it would be reasonable to allow the network to chose 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. 3.2. We trained the network on 100 of these instances. The training involved 1500 sweeps through the 100 examples with the weights being updated after each sweep. The details of the training procedure are given in the following section. After this substantial experience of the domain, 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. A typical example of the activity levels in all layers of the network is shown in Fig. 3.5.

The fact that the network can learn the examples it is shown is not particularly surprising. 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?

The representations

Figure 3.6 shows the weights on the connections from the 24 units that are used to give a neutral input representation of **person1** to the six units that are used for the network's internal, distributed representation of **person1**. These weights define the 'receptive field' of each of the six 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 and that most of the other units ignore

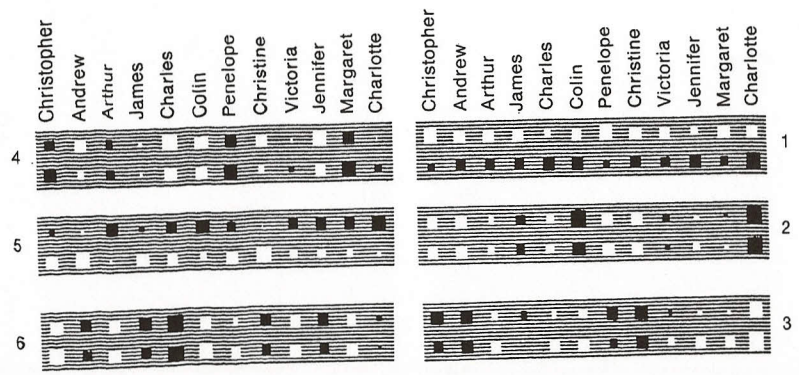


Fig. 3.6 The weights from the 24 input units that represent people to the 6 units in the second layer that learn distributed representations of people. White rectangles stand for excitatory weights, black for inhibitory weights, and the area of the rectangle enclodes the magnitude of the weight. The 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.

this distinction. This 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 explictly 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]

It is initially surprising that none of the six units encodes sex. This is probably because of the particular set of relationship terms that were used. Each of the 12 relationship terms completely determines the sex of **person2** so the sex of **person1** is redundant. Figure 3.7 shows that one of the six units which encodes the 12 possible relationships is entirely devoted to predicting the sex of **person2**. 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.

Microinferences and scientific laws

There is an interesting analogy between the way in which the network represents propositions and the way in which scientists represent the structure

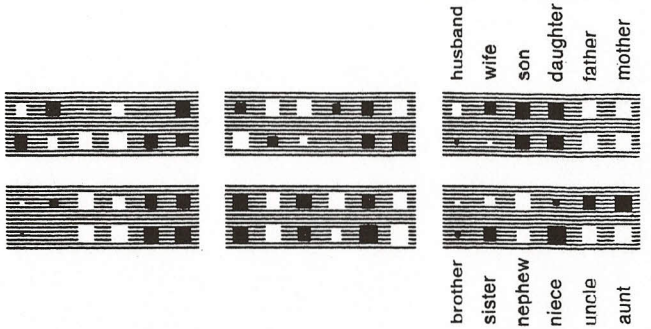


Fig. 3.7 The weights from the 12 input units that represent relationships to the 6 units in the second layer that learn distributed representations of the relationships.

of the natural world. In order to express the regularities in the data, a scientist must describe them using appropriate terms. For example, substances with widely different appearances much be grouped together into such categories as 'acid', 'salt', or 'base'. If the appropriate terms are given in advance, the task is much easier than if the terms themselves must be discovered by searching for sets of terms that allow laws to be expressed. The gradient descent procedure used by the network also has its analogue in scientific research. Initial definitions of the descriptive terms can be used to formulate laws and the apparent exceptions can often be used to refine the definitions.

Naturally, there are also many important differences between the way scientists proceed and the way learning occurs in the network. Scientists would not normally be satisfied if their theory consisted of a very large number of statistical 'laws' and they needed a computationally intensive procedure to decide what the laws predicted.

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. Figure 3.8 shows the activity levels of all 24 output units for each of the four test cases after training.

Any learning procedure which relied on finding direct correlations between

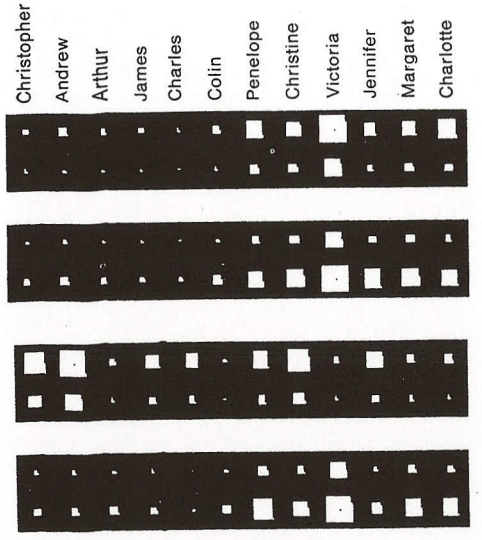


Fig. 3.8 The activity levels of the output group in the four test cases that were not shown during training. The dots are on the correct answers. Notice that in every case the network has a slight tendency to activate the isomorphic person in the other family tree.

the input and output vectors would generalize very badly on the family tree task. Consider the correlations between the filler of the **person1** role and the filler of the **person2** role. The filler of **person1** which 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.

The good generalization exhibited by the network shows that the structure which it has extracted from the training examples agrees with the structure which we attribute to the domain. We would like to be able to say that the training set implicitly contains the information about how to generalize and that the network has correctly extracted this implicit information. But this requires a prescriptive domain-independent theory of how a set of examples should be used for making generalizations. Such a theory would constitute the 'computational level' of understanding for learning research (Marr 1982), and would be a major advance which could guide research at the algorithmic and implementation levels. Unfortunately, we know of no such theory and so we are restricted to showing that the learning procedure produces sensible generalizations in particular domains.

The back-propagation learning procedure

The aim of the learning procedure is to find a set of weights which ensure that for each input vector the output vector produced by the network is the same as (or sufficiently close to) the desired output vector. If there is a fixed, finite set of input–output cases, the total error in the performance of the network with a particular set of weights can be computed by comparing the actual and desired output vectors for each case. The error, E, is defined as

$$E=\frac{1}{2}\sum_c\sum_j y_{jc}-d_{jc})^2 \tag{3.3}$$

where c is an index over cases (input–output pairs), j is an index over output units, y is the actual state of an output unit, and d is its desired state. To minimize E by gradient descent it is necessary to compute the partial derivative of E with respect to each weight in the network. This is simply the sum of the partial derivatives for each of the input–output cases. For a given case, the partial derivatives of the error with respect to each weight are computed in two passes. We have already described the forward pass in which the units in each layer have their states determined by the input they receive from units in lower layers using Equations (3.1) and (3.2). The backward pass which propagates derivatives from the top layer back to the bottom one is more complicated.

The backward pass starts by computing $\partial E/\partial y$ for each of the output units. Differentiating Equation (3.3) for a particular case, c, and suppressing the index c gives

$$\frac{\partial E}{\partial y_j}=y_j-d_j \tag{3.4}$$

We can then apply the chain rule to compute $\partial E/\partial x_j$:

$$\frac{\partial E}{\partial x_j}=\frac{\partial E}{\partial y_j}\frac{dy_j}{dx_j}$$

Differentiating Equation (3.2) to get the value of dy_j/dx_j gives

$$\frac{\partial E}{\partial x_j}=\frac{\partial E}{\partial y_j}y_j(1-y_j) \tag{3.5}$$

This means that we know how a change in the total input, x, to an output unit will affect the error. But this total input is just a linear function of the states of the lower level units and the weights on the connections, so it is easy to compute how the error will be affected by changing these states and weights. For a weight, w_{ji}, from i to j the derivative is

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ji}}$$

$$= \frac{\partial E}{\partial x_j} y_i \tag{3.6}$$

and for the output of the ith unit the contribution to $\partial E/\partial y_i$ resulting from the effect of i on j is simply

$$\frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial y_i} = \frac{\partial E}{\partial x_j} w_{ji}$$

So, taking into account all the connections emanating from unit i, we have

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} w_{ji} \tag{3.7}$$

We have now seen how to compute $\partial E/\partial y$ for any unit in the penultimate layer when given $\partial E/\partial y$ for all units in the last layer. We can therefore repeat this procedure to compute $\partial E/\partial y$ for successively earlier layers, computing $\partial E/\partial w$ for the weights as we go. The amount of computation required for the backward pass is of the same order as the forward pass (it is linear in the number of connections).

One way of using $\partial E/\partial w$ is to change the weights after every input–output case. This has the advantage that no separate memory is required for the derivatives. An alternative scheme, which we used in the research reported here, is to accumulate $\partial E/\partial w$ over all the input–output cases before changing the weights. The simplest version of gradient descent is then to change each weight by an amount proportional to the accumulated $\partial E/\partial w$.

$$\Delta w = -\varepsilon \frac{\partial E}{\partial w} \tag{3.8}$$

This method does not converge as rapidly as methods which make use of the second derivatives, but it is much simpler and can easily be implemented by local computations in parallel hardware. It can be significantly improved, without sacrificing the simplicity and locality, by using an acceleration method in which the current gradient is used to modity the velocity of the point in weight space instead of its position.

$$\Delta w(t) = -\varepsilon \frac{\partial E}{\partial w(t)} + \alpha \Delta w(t-1) \tag{3.9}$$

where t is incremented by 1 for each sweep through the whole set of input–output cases, and α is an exponential decay factor between 0 and 1 that determines the relative contribution of the current gradient and earlier gradients on the weight change. Equation (3.9) can be viewed as describing the

behaviour of a ball-bearing rolling down the error-surface when the whole system is immersed in a liquid with viscosity determined by α.

The learning procedure is entirely deterministic, so if two units within a layer start off with the same connectivity and the same weights there is nothing to make them ever differ from each other. We therefore break symmetry by starting with small randon weights.

The learning procedure often works better if it is not required to produce outputs as extreme as 1 or 0. To given an output of 1, a unit must receive an infinite total input and so the weights grow without bound. All the examples of back-propagation described in this paper use a more liberal error measure which treats all values above 0.8 as perfectly satisfactory if the output unit should be on and all values below 0.2 as perfectly satisfactory if the output unit should be off. Otherwise, the error is the squared difference from 0.8 or 0.2.

There are many aspects of the learning procedure which make it highly unsuitable as a model of learning in real neural networks. There are ways of removing the prohibition on recurrent connections (Rumelhart, Hinton, and Williams 1986) and it may be possible to overcome the need for an externally supplied desired output vector. But the back-propagation phase is central to the learning procedure and it is quite unlike anything known to occur in the brain. The connections are all used backwards, and the units use a different input–output function. We therefore view this learning procedure as an interesting way of demonstrating what can be achieved by gradient descent, without claiming that this is how gradient descent is actually implemented in the brain. Nevertheless, the success of the learning procedure suggests that it is worth looking for other more plausible ways of doing gradient descent.

The learning parameters used for the family tree simulation

We tried several different values for the parameters ε and α in Equation (3.9). We finally chose to use $\varepsilon = 0.005$ and $\alpha = 0.5$ for the first 20 sweeps through the 100 training examples and $\varepsilon = 0.01$ and $\alpha = 0.9$ for the remaining sweeps. The reasons for varying the parameters during learning and the methods used to choose reasonable parameters are discussed in more detail in Plaut, Nowlan, and Hinton (1986). All the weights were initially chosen at random from a uniform distribution between -0.3 and $+0.3$.

To make it easier to interpret the weights, we introduced 'weight-decay'. Immediately after each weight change the magnitude of every weight was reduced by 0.2 percent. After prolonged learning the decay was balanced by $\partial E/\partial w$, so the final magnitude of each weight indicated its usefulness in reducing the error. Weight decay is equivalent to modifying the error function so that, in addition to requiring the error to be small, it requires the sum of the squares of the weights to be small. A side-effect of this modification is that it sometimes causes two units to develop very similar sets of weights with each

weight being half as big as it would be if the job was done by a single unit. This is because $(0.5w)^2 + (0.5w)^2 < w^2$.

To achieve negligible error without weight decay required 573 sweeps through the 100 training examples. The weights shown in Fig. 3.6 were obtained by allowing the learning to run for 1500 sweeps with weight-decay.

Acknowledgements

The research reported here was supported by grants from the System Development Foundation. This paper first appeared in the *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, Massachusetts, 1986 which was published by Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Notes

1. In a conventional Lisp program, it is easy to separate the representation from the procedures used for retrieval and so it is easy to use a representation that is not in a form that is helpful for retrieval. In a neural net it is probably more important to choose representations that can directly cause the required effects without the intervention of a complex interpreter. This is one of the many differences in representation considerations that follows from the difference between the von Neumann architecture and a massively parallel network.

2. 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 idealization.

3. The words of a natural language seem to work in a very similar way. They stand for concepts while indicating very little about how those concepts should be represented internally. Monomorphemic words with similar meanings do not generally have similar forms. So a pattern of activity based on the form of the word is not a good way of capturing the similarities between meanings. There must be a process that maps word forms into word meanings. This process must be far more complex than the simulations we present here because many word forms, like 'bank', are ambiguous and so the process of going from the input representation to a representation of the word meaning cannot be performed separately for each word. The meaning of whole phrases or sentences must be used for disambiguation (Waltz and Pollack 1985).

4. Rumelhart *et al.* 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. We have not applied this version to the family relationships task.

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 large network and so we opted for the simpler case.

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.

References

Barlow, H. B. (1972). Single units and sensation. A neuron doctrine for perceptual psychology? *Perception*, **1**, 371–94.

Hinton, G. E. (1981). Implementing semantic networks in parallel hardware. In *Parallel models of associative memory* (eds. G. E. Hinton and J. A. Anderson). Erlbaum, Hillsdale, NJ.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In *Parallel distributed processing: explorations in the microstructure of cognition* (eds. D. E. Rumelhart, J. L. McClelland, and the PDP research group). Bradford Books, Cambridge, Mass.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, April, **79**, 2554–8.

Kohonen, T. (1977). *Associative memory: a system-theoretical approach.* Springer, Berlin.

Marr, D. (1982). *Vision.* Freeman, San Francisco.

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

Plaut, D. C., Nowlan, S. J., and Hinton, G. E. (1986). *Experiments on back-propagation.* Technical Report CMU-CS-86-126, Carnegie-Mellon University, June.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructure of cognition* (eds. D. E. Rumelhart, J. L. McClelland, and the PDP research group). Bradford Books, Cambridge, Mass.

Waltz, D. L. and Pollack, J. B. (1985). Massively parallel parsing: a strongly interactive model of natural language interpretation. *Cognitive Science*, **9**, 51–74.

Willshaw, D. J., Buneman, O. P., and Longuet-Higgins, H. C. (1969). Nonholographic associative memory. *Nature*, **222**, 960–2.

Willshaw, D. (1981). Holography, associative memory, and inductive generalization. In *Parallel models of associative memory* (eds. G. E. Hinton and J. A. Anderson). Erlbaum, Hillsdale, NJ.