ELSEVIER

Computer Standards & Interfaces 16 (1994) 279–293

COMPUTER STANDARDS & INTERFACES

Opening the black box of connectionist nets: Some lessons from cognitive science †

Noel E. Sharkey *, Amanda J.C. Sharkey, Stuart A. Jackson

Department of Computer Science, University of Sheffield, Sheffield, UK

Abstract

Connectionist networks are not simply opaque black boxes with useful and efficient computational properties. Rather they have important internal structure that must be harnessed if their full potential as a novel computing technology is to be realised. First, the importance and usefulness of opening the black box is discussed and research is reviewed on how internal representations have been studied and used in the Cognitive Science literature. In the second section, a simple method for the geometrical analysis of decision space is presented. This shows connectionist networks as transparent boxes in which their computational properties are clear. The paper finishes with an example of how decision space diagrams can be useful for investigating under what circumstances it is best to adapt old weights for use in novel tasks.

Key words: Connectionist representation; Black box computing; Decision space analysis; Neural networks; Cluster analysis

0. Introduction

A common view of connectionism within the computer science community is that it is, at best, a 'black box' technology. Nowadays it is possible to use off-the-shelf software to train a net on some particular input-output task by simply exposing it to the training data. This type of extensional programming does not require the user to know anything about the innards of the net. Once trained such a net can be slotted into a larger structured program as a subroutine that can be called for specialist purposes. However, such an approach is severely limited. Ignoring the internal computational properties of networks results in a failure to realize their full potential, in as much as it is not possible, for instance, to determine what the net is computing when the domain from which the training samples were chosen is very large or unbounded. More importantly, without an understanding of the internals of neural networks, all manipulation of representations will have to occur external to that 'black box'. As we shall show in Section 1, there is, in fact, much

* Corresponding author.

† This research was supported, in part, by an award from the Economic and Social Research Council, U.K., Grant No. R000233441.

SSDI 0920-5489(94)00009-6

computation that can be carried out internal to a network without recourse to a symbolic interface, e.g. by transforming or manipulating the internal representations of networks. This may be a novel form of computation and we need to understand it more. In Section 2, a simple geometric technique is presented to provide a method of visualising the computational space of multilayer nets. The utility of *decision space diagrams* is explored in some detail and the paper ends by examining the use of the techniques to find out how nets may adapt old weights for novel tasks.

1. Unit representations

One group of connectionists particularly interested in opening the black box are those who have an interest in human cognition and human internal representations; the cognitive connectionists. We first look at their work and some of its implications for understanding the computation in nets.

Cognitive connectionists have typically associated the term 'representation' with patterns of activation of simple processing units. This notion is a direct descendant of earlier spreading activation models in psychology, and it became more prevalent with the introduction of multilayer nets into cognitive science. Questions about representation provide a good starting point for examination of the black box, since they have been the subject of considerable investigation. Sharkey [25] has reviewed the diversity of representational formalisms in the connectionist literature, and they are summarised in the typology shown in Fig. 1. The major distinction in the typology is between the *localist* (e.g. [5]) and the *distributed* (e.g. [13]) branches. Although Hinton [12] points out that the definitions of localist and distributed representation are relative (they refer to the relationship between the terms of a descriptive language and a connectionist implementation), both representational classes nonetheless have definite advantages and disadvantages. The primary advantage of localist representation is its *semantic transparency*. Each unit is an explicit token, clearly labelled, and so it is easy to see what its function

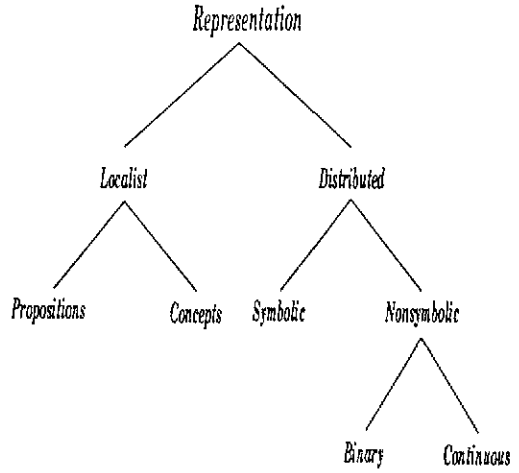


Fig. 1. A typology of connectionist representation, adapted from Sharkey [25].

is in the network. Largely due to their transparency, such representations have been popular with researchers from the AI tradition. Distributed representations, on the other hand, are more memory efficient, have more natural generalisation properties (c.f. [25]) and as shall be shown, are transparent if looked at in the right way.

The focus of the current paper is on multilayer networks such as that illustrated in Fig. 2 and its variants. These are trained using the backpropagation learning rule [21]. In many tasks, including those reported in this paper, the inputs and outputs of such a net are binary representation vectors (localist or distributed) representing some symbolic strings. In networks like the one shown in Fig. 2, the mapping from an input vector, v, to an output vector, o, occurs in two time steps. At time t_1 the input states are propagated to the hidden units, a vector h, by an update function: $f(v) = h$. Standardly, the update function for the jth hidden unit is $h^j = 1/1 + e^{-x}$, where x is the weighted sum of the inputs to the unit. At time step t_2 the hidden unit states are propagated to the outputs by the same update function, $f(h) = o$.

The representational class of most interest in this paper comprises those representations developed over the internal hidden units of the kind of

multilayer network shown in Fig. 2 (the *distributed-nonsymbolic-continuous* branch shown in Fig. 1). These are vectors of continuously valued activations, normally in the range 0 to 1. It is the identification of hidden unit vectors with 'internal representations' that has been the cause of a great deal of excitement among cognitive theorists. Their importance to us is that they form an intermediate computational step: their function is to represent a transformation of the input space into appropriately separable regions to allow the representation of linearly dependent pattern classes. Unlike the other types of representation outlined above, these may be thought of as representations of complex expressions in which the tokens of the input constituents are destroyed in their combination. This makes them opaque to the naked eye. However it is possible to use indirect methods to examine their nature and functionality.

One of the most common ways to study the internal computation in a net is to examine the Euclidean distance relationships between the vectors of hidden unit activations. This is done by propagating input activation states through the trained net onto hidden unit activation states and collecting the subsequent hidden unit vectors as data, $f(c_i) = h_i$, where i indexes an input vector and its corresponding hidden unit representation. This allows statistical analyses of the relationships between representations. For example, hierarchical cluster analysis enables visualisation of the pair-wise Euclidean distances between the hidden units, $\|h_1 - h_2\|$, in representation space (cf. [7], for a review of the clustering methods, and [11] for an interesting discussion of their use). The assumption behind the use of cluster analysis is that functionally similar (or 'semantically similar') input patterns will be close together in terms of Euclidean distance and will thus cluster together in hidden unit space.

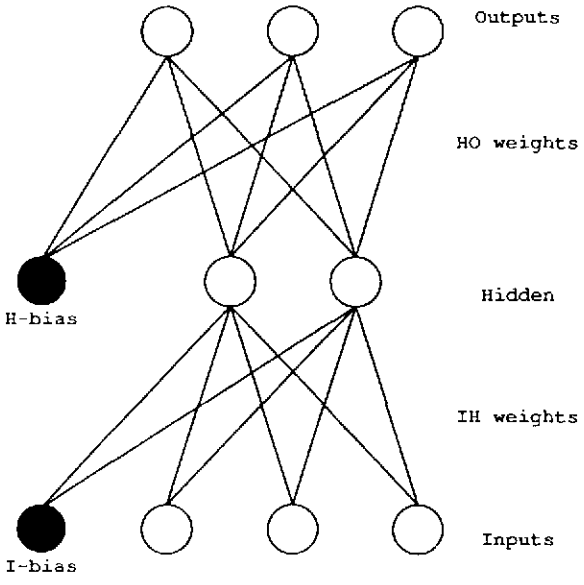


Fig. 2. A feedforward net with two weight layers and three sets of units.

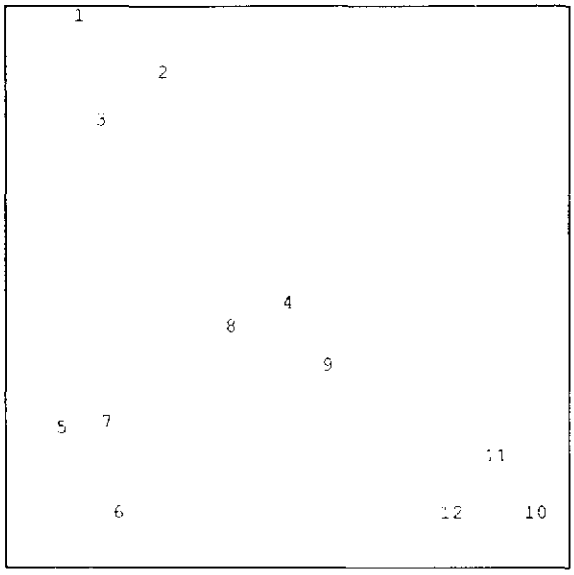


Fig. 3. A hidden unit space with numbered locations.

A hidden unit space is illustrated in Fig. 3 with the coordinates of twelve input patterns shown as numbered locations [1].

[1] Backpropagation tends to push the hidden unit vectors towards the axes of the space. It is also possible to find instances of backpropagation learning in which some points are located in the middle of the space, for example, where they are novel inputs patterns and they are orthogonal (or nearly orthogonal) to the trained patterns as illustrated in the examples.

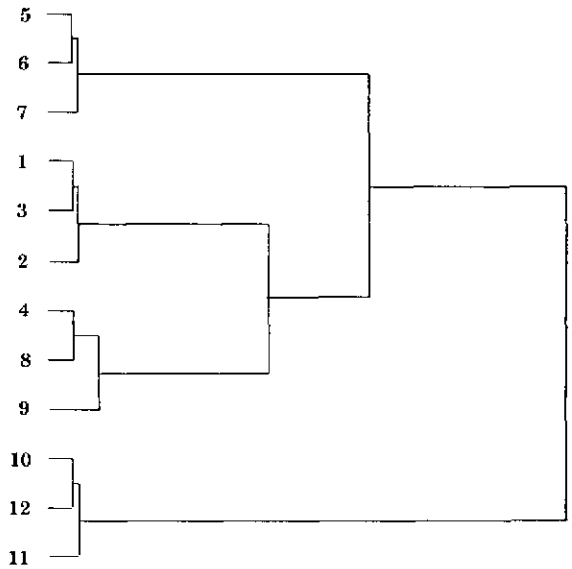


Fig. 4. A dendogram of the hierarchical cluster analsysis of the data points shown in Fig. 3.

These locations have not been, as yet, labelled according to their input or class. Since the labels do not have a causal role in the computation, their omission enables us to look at the data neutrally. *Post hoc* explanations of why, for example, 'tomato' was clustered with 'cucumber' instead of 'apple' can be very distracting. A visual inspection of Fig. 3 reveals four clusters of locations in the space: (i) 1, 2, 3; (ii) 4, 8, 9; (iii) 5, 6, 7; and (iv) 10, 11, 12. This is supported by a hierarchical cluster analysis of the squared Euclidean distance between the points. A dendogram of the analysis, shown in Fig. 4, only differs from the visual inspection in linking two of the clusters, (i) and (ii), in a central superordinate cluster [2]. We could speculate that the clusters represented types, for example, *animal, vegetable,* and *mineral*. The central (superordinate) cluster in the dendogram could be showing that the two clusters are subordinate types such as *mammal* and *non-mammal*. Alternatively, the clusters could represent *agents, verbs,* and *actions*, with the verbs subdivided into *transitives* and *intransitives*. This would depend entirely on the domain and the task.

Such detailed statistical analyses have been used extensively in the literature to argue that fully distributed unit representations can contain a rich variety of structural and semantic information. For example, it has been suggested that they can contain information about the typing of their constituents [6], their syntactic class [11], and their semantic class [22], as well as information about path and long range dependencies when trained on finite state grammars [23]. Much of this literature is reviewed in Sharkey [25] and will not be repeated again here.

We know quite a lot about the properties of hidden unit representation and the kinds of information they can encode, but what we need to know is whether these representations are in a form that allows them to be used in computations other than those in which they have been previously used. The alternative is that all of the manipulations are carried out external to the network in a symbolic interface: as mentioned earlier, the network is simply a subroutine in a larger conventional program. The problem for the connectionists according to some (e.g. [8]) is that continuously-valued distributed representations are not compositional and thus they cannot have the properties necessary to enable structure sensitive operations. Symbolic strings allow such structure sensitive operations because in their mode of combination the constituents are to-

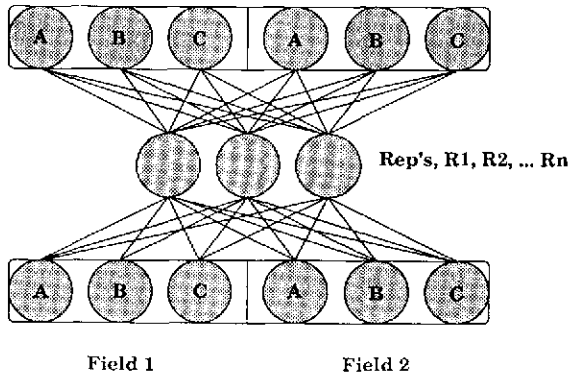


Fig. 5. A valency 2 RAAM net for encoding binary trees.

[2] Of course the utility of cluster analysis is clearer when it is a two dimensional reduction of a hidden unit space that is multi-dimensional and therefore more difficult to visualise.

kened whenever the complex expression is tokened. For example, in order to develop an expression from a sentence such as 'John kissed Mary', arbitrary symbols representing the constituents JOHN, KISSED, and MARY are combined in a contextually independent concatenation to produce the propositional representation, KISS (JOHN, MARY). Whenever this latter complex expression is tokened, its constituents, KISS, MARY and JOHN are also tokened. This makes the manipulation of the representations by a mechanism sensitive to the syntactic structure resulting from concatenative compositionality relatively easy.

However, as van Gelder [9] has argued, the symbolic method of compositionality is not the only one. Concatenating constituent tokens into a symbolic string is not a necessary condition of compositionality. Connectionism provides a means of combining tokens without those tokens appearing in the complex expression, and without the need for concatenation. As van Gelder [9] points out, continuously valued hidden units provide an instance of a novel style of compositionality, a merely functional or *non-concatenative* compositionality. Vectors of hidden unit activations are not concatenative representations, in the symbolic sense, since the constituent tokens are destroyed in a non-concatenative composition, what is termed the operation of *superposition* [3].

These ideas of non-concatenative compositionality and superposition have given rise to the *spatial systematicity assumption* (cf. [22]) which has been used extensively to develop a theory of systematic connectionist computation. Thus, in the connectionist scheme of compositionality espoused by e.g. van Gelder [9] the structural relations between 'distributed representations' are based upon, not *syntactic* similarity as in the symbolic tradition, but rather *spatial* similarity.

As an example of non-concatenative compositionality and spatial structure, van Gelder referred to the style of representations developed by Recursive Auto-Associative Memory (RAAM) networks [29]. The RAAM architecture is the same as the standard feedforward net with two layers of weights (for encoding and decoding the hidden unit representations) as shown in Fig. 5 and the standard back propagation algorithm is employed for learning. Pollack [20] has shown the power of the RAAM system for encoding a sequential stack with PUSH and POP and also for encoding and decoding syntactic trees. The whole trees are represented in a single layer of hidden units and can be decoded in cycles until the terminal symbols appear as the outputs. The difference between RAAM and the usual back propagation net rests on the method for presenting the input patterns. The input space of a RAAM net is divided into n partitions, with k units in each partition. The size of n depends directly on the maximum valency of the tree to be represented (in our simple example in Fig. 5, $n = 2$). This is an autoassociative net which means that it simply has to reproduce its input on the outputs i.e. what goes in, comes out. Since the output vector is identical to the input vector, both have nk units and there are k hidden units, as shown in Fig. 5.

The operation of a RAAM system will be described briefly here using the example of a simple binary tree, ((A B) (A C)), as illustrated in Fig. 6. [4].

The representation of the binary tree shown in Fig. 6 is formed as follows: First, A and B are presented in the two input vector partitions i.e. A is set to +1 in Field 1 and B is set to +1 in Field 2. Activation is passed through the net to the output units and compared with a training target which is the same as the input. The resulting hidden unit representation R_1 is kept to one side (on an external stack or some such). This is a hidden unit representation of the A/B subtree. Next, A and C are presented and auto-associated and the resulting hidden unit representation R_2 is put to one side. This is a hidden unit represen-

[3] There are methods for extracting the representations of the constituent from such hidden unit representations however, see Sharkey [26] and [27a].

[4] All syntactic structures can be expressed as binary trees when expressed in Chomsky Normal Form.

tation of the A/C subtree. Finally the representations of the two subtrees, R_1 and R_2, are presented as input, one in each field, and autoassociated as before. The resulting hidden unit representation R_{top} is a representation of the entire tree. This, according to van Gelder [9], is a non-concatenatively compositional representation, in that a RAAM net provides general, effective and reliable processes for producing an expression given its constituents. Moreover, it provides a recursive process for decomposing the expression back into its constituents again as shown in Fig. 7.

The compact distributed representation of the entire tree, R_{top}, can be decoded using only the upper portion of the net. This is called the RAAM decoder and is illustrated in Fig. 7. R_{top} is presented directly to the hidden units and passed on to the outputs. The trained net will produce R_1 and R_2 as output. Then R_1 and R_2 are presented in turn to the hidden units. The output for R_1 will be A/B, and the output for R_2 will be A/C. Pollack [20] presents a range of interesting simulation results which show RAAM to be a very effective method for encoding and decoding recursive structures and it has also been used as the back end of a computational sentence parser

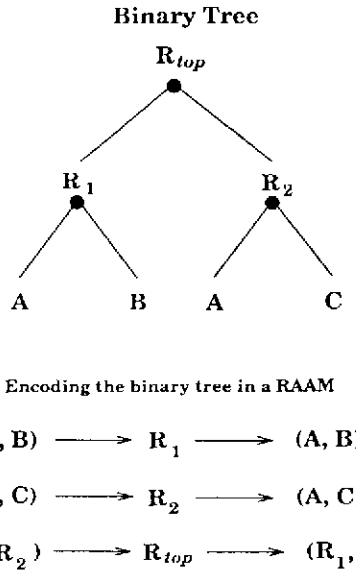


Fig. 6. An example of encoding a binary tree in a RAAM net.

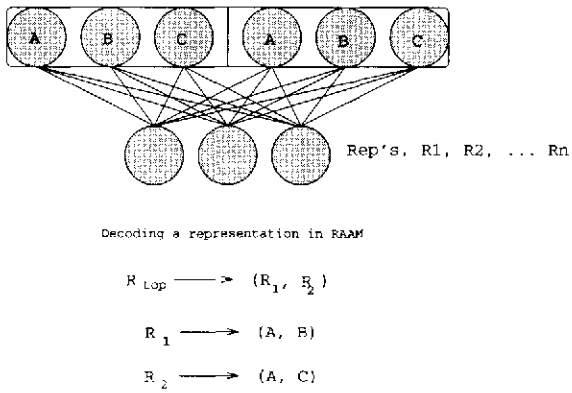


Fig. 7. A example of decoding the compact representation of a tree, R_{top}. The upper part of the figure shows a RAAM decoder net.

[29]. Although the RAAM method relies on an external stack, a purely connectionist implementation could be constructed. The important point is that Pollack has demonstrated how seemingly unstructured hidden unit activations can encode recursive representations in a compact form that has the properties of compositionality that van Gelder [9] requires.

In sum, the study of the internal computation of connectionist nets was started by examining the pairwise Euclidean distances between vectors of hidden unit activation. This has revealed a number of interesting properties of internal network computation. A further development was the realisation that such internal representations, although non-concatenative, could, in combination with the spatial systematicity assumption, provide a coherent account of systematic internal network computation. We will, in Section 2, however, explain why the Euclidean distance metric has limited utility in describing the computational properties of a trained network. In the meantime, it will be shown how internal representations may be used for tasks other than those on which they were trained for.

1.1. Structure sensitive manipulation

We have seen how RAAM encoded representations exhibit the properties needed for non-

concatenative compositionality. The question that must be asked now is do fully distributed RAAM representations allow direct structure sensitive operations, or do they first need to be decoded back into a symbolic form, as suggested by Fodor and McLaughlin [8]? A failure to answer this question in the affirmative would mean that in order to process language effectively, the connectionist theorist would have to convert hidden unit representations into symbolic form in order to use or manipulate them. Fortunately however, this is not necessary. Hidden unit activations do allow direct structure sensitive operations, as the growing number of examples of representation manipulation occuring internal to the black box in the connectionist literature attests.

Chalmers [3], for example, subjected the contentious notion of *systematicity* to empirical scrutiny by studying active/passive sentence transformations. First, a corpus of representations was developed for both active and passive versions of a number of sentences by training a RAAM network [20]. Second, the resulting corpus was divided into two distinct sets corresponding to the active and passive forms for each sentence. Third, a standard feedforward net with two layers of weights running the backpropagation algorithm was used to map the appropriate pairs of representations onto one another. This latter mapping net, which Chalmers refers to as the Transformation net, shown in Fig. 8, was easily trained to map the representations of any of the sentences in the training set onto their active or passive form (and decode them back into their sentence form using the RAAM decoder). A germane question that arises from these results concerns whether a mapping net can generalize its transformation behavior to novel examples. In one experiment, Chalmers trained the Transformation net on 75 sentence representations and reserved 50 others to test generalization. After training was successfully completed, a test of the 50 novel sentence representations resulted in a 100% correct generalization performance.

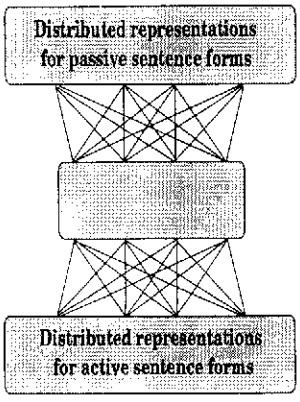


Fig. 8. A transformation net as used by Chalmers [3].

Subsequent studies have reported similar findings. For example, Niklasson and Sharkey [17] (1994) showed successful transformations for both the simple manipulation of logical formulae and manipulation using de Morgan's rule. Other successful use of such manipulations have occurred in domains involving plausible inference [2] English to Spanish translation [4], sequential parsing [29] and necessary inference [7]. Thus, it appears that hidden unit representations do enable structure sensitive operation. There is much to be gained from their study as such operations constitute a novel form of computation that expands the range and usefulness of multilayer nets.

In sum, we can conclude at the end of this section, that continuously-valued distributed unit representations constitute a unique and powerful class, distinct from both more traditional symbolic representations and also from less subtle connectionist representations. In virtue of the novel manner of their composition, they exhibit spatial similarity relations, usefully understood as similarities of location in representational space, which provide the domain for direct structure sensitive operations. In the next section we open the black box further by presenting a method for visualizing the entire computational space of a multilayer net.

2. Weight representations and decision space

As discussed in Section 1, the tradition in the cognitive science community has been to concentrate on representations which are defined over the hidden unit activations. But this is only half

the story. Perhaps a more informative way to think of computation in the kind of net shown in Fig. 2, is in terms of the whole network and how it computes some global function. Although examining hidden unit activations tells us about the behaviour of the network over the input weights, to understand the emergent computation of a whole network, we also need to consider the role of the output weights in relation to the hidden unit representations. This can be done by using a geometric analysis of the collective weights in a net. This move brings the output weights into the investigation of computation and, as we shall see, their collective role is to carve up the 'representation space' of the hidden units into different regions.

It is possible to visualize the role of the output weights in relation to the hidden unit representations by using what Sharkey et al. [27] call *decision space analyses* to provide global representations of network computation [5]. Such analyses have been used extensively for networks with a single matrix of weights between the inputs and outputs. These make a good starting place to explain the technique. Before discussing multilayer nets, we first examine some simple binary nets like those developed by McCullough and Pitts for the basic functions: P AND Q, P OR Q, and NOT P AND Q. The nets are shown in Figs. 9(a–c). The output function is the Heaviside or threshold function, where the output, $o = 1$, if $\mathbf{w}.\mathbf{v} > \theta$, where θ is a value between 0 and 1. The input space for the nets shown in Fig. 10 is two dimensional and can thus be described as a square with the binary inputs arranged on its vertices. Each vertex can be thought of as a vector from the origin. To see what the net computes, the weights to the output unit may also be plotted as a vector from the origin. Following Nillson [18], a decision line, perpendicular to the weight vector, can then be drawn through the input space by solving the equation, $xw_1 + yw_2 = \theta$, for x and y, where w_1 and w_2 are the two weights, and x and y are coordinates of the decision boundary. The

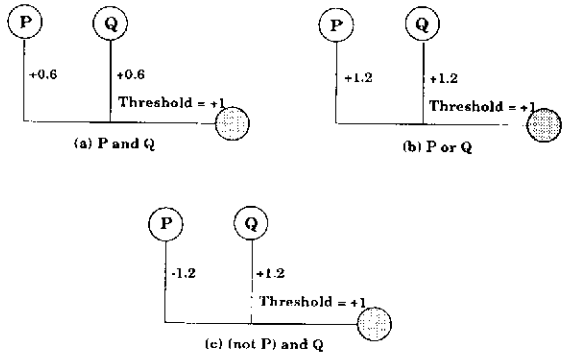


Fig. 9. Three 'McCullough-Pitts' binary nets for the three functions: P AND Q, P OR Q, (NOT P) AND Q.

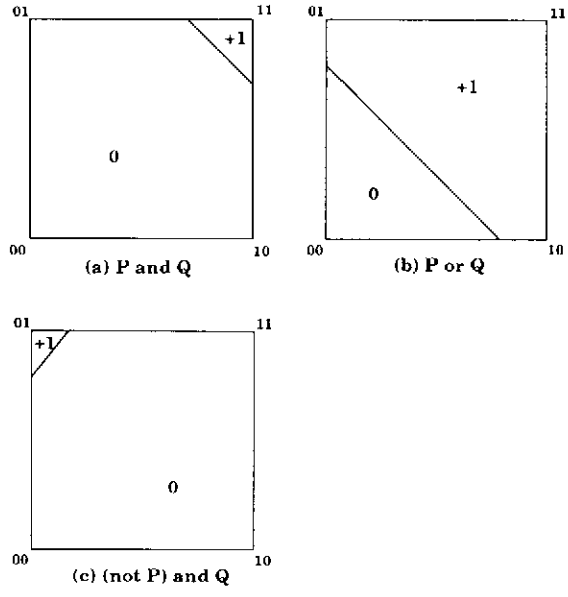


Fig. 10. The decision regions for the functions, P AND Q, P OR Q, (NOT P) AND Q. The corresponding nets are shown in Figs. 9(a)–(c).

[5] In the case of low dimensional hidden unit spaces, these decision spaces may be easily visualised in either 2 or 3 dimensions. With higher dimensions, understanding relies on more abstract geometrical analyses. In addition, although the practise of plotting output weights in input space has been explored by a number of other authors (e.g. [16,15,32]; the specific practise detailed here, of plotting (the decision lines implemented by the) output weights in hidden unit space, is a novel approach, yielding qualitatively different analyses from more 'standard' hyperplane techniques.

line divides the square into two *decision regions* that show which of the inputs will produce a +1 as output and which will produce a zero. Figs. 10(a–c) show the decision regions for the corresponding nets in Figs. 9(a–c). The diagrams entirely determine the computation of the nets. For example, if continuous rather than binary values were used as input to the nets, the decision line would show the regions of generalisation of the net i.e. which continuous valued inputs would map onto a +1 output and which would map onto a zero output. For nets with multiple outputs there would be one decision line for each output unit.

In a network with a single matrix of weights the input points are all fixed in advance and so training is limited to moving the decision lines around the input space until all of the points are captured in the required regions. Having such fixed input points restricts the functions that such a net can compute to those in which the pattern classes are linearly separable. This is a severe restriction since the ratio of linearly separable to linearly dependent pattern classes rapidly approaches zero as the dimensionality of the input space increases. For example, it is not possible to move a line around the 2D input space to separate the regions appropriately for the XOR function (11 → 0, 00 → 0, 10 → 1, 01 → 1). One solution is to translate the input points into a new space such that they can be separated appropriately by the decision line. This is the solution strategy used by backpropagation learning in multilayer nets.

As a simple example of the necessity of using more than a single matrix of weights, imagine that a network was designed so as to check the grammatical legality of simple sentences. It would take a word string as input, and would be required to output a 1 if the string formed a legal sentence, and a 0 otherwise. Restricting the encoding of the net to three lexical items, JEAN, BILL and LAUGHED (where each word is indicated by a single binary 'bit' in a 3-D input vector) results in only two legal sentences: Jean laughed, and Bill laughed. All single items (Jean, Bill, or laughed), and other combinations should produce a '0' as output. The geometry of the decision space is shown in Fig. 11. This illustrates clearly that it is not possible to divide this space appropriately for the sentence task using a net with a single layer of weights. Since there is only one output decision unit, there is only one decision plane, shown in Fig. 11 that can divide the space. All vertices of the cube that are on one side of the plane will produce an output of '1', while all vertices on the other side will produce a '0' as output. If the reader tries to mentally move the plane around inside the cube, so as to divide the space in Fig. 11 appropriately, such that the pairs of sentences JEAN LAUGHED and BILL LAUGHED are separated from all of the other vertices of the cube, they will very quickly discover that such a bisection is impossible. This is because any pattern from the 'legal' class may be written as a linear combination of the elements from the 'illegal' class, i.e. the pattern classes are linearly dependent.

In order to solve this problem, a second layer of weights is required, namely, the input to hidden (IH) weights (see Fig. 2). Their function is to map the input patterns onto a representation space in such a way as to create relations among

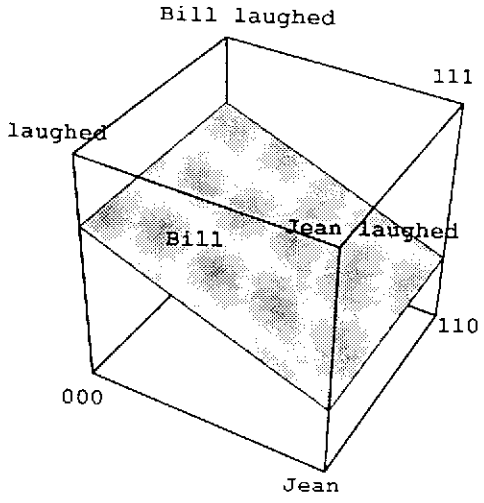


Fig. 11. Input space represented as a cube, with the binary patterns shown as vertices. It is not possible to separate "Bill laughed" and "Jean laughed" from the rest of the vertices using a plane.

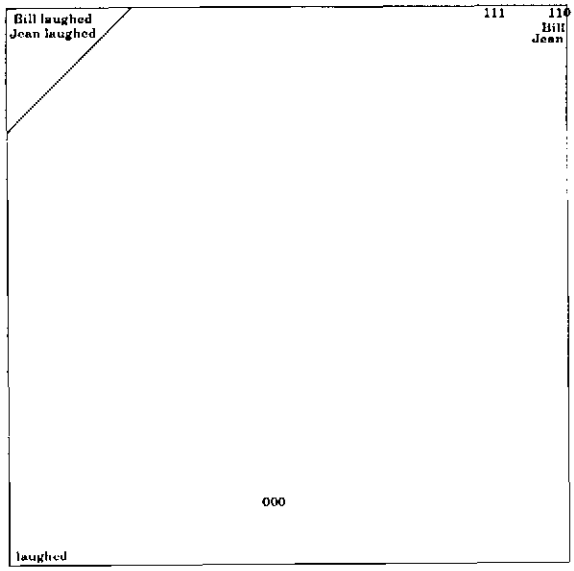


Fig. 12. Spatial relations of the transformed input patterns (from Fig. 11) in 2D Hidden Unit space.

the input patterns appropriate for the task [6]. By assigning hidden unit coordinates to the vertical and horizontal axes of a coordinate space, such a mapping can be illustrated, as shown in Fig. 12. Notice that the realignment of the inputs enables the output unit to draw an appropriate decision boundary through the space to make it possible to compute the simple sentence checker [7].

Two things that can be said of the weight representations are that:

(i) the function of the IH weights is to transform linearly dependent pattern classes (into linearly separable pattern classes) by creating a new representational space over the hidden units, and
(ii) the function of the HO weights is to carve up the resultant representational space appropriately.

It seems then that moving to the level of collective weight representations, as shown in the decision space diagrams, has a number of advantages over focussing exclusively on hidden unit representations as in distance statistics such as cluster analysis. One advantage is that decision space helps to elucidate the mechanisms of generalization. To illustrate, consider that if any point in Fig. 12 falls in the lower region of the square, one can see that the output will be 0, whereas if it falls in the upper region of the square, the output will be 1. If the generalization is incorrect, either the input point, or the decision line, or both, can be shifted. A second advantage is that the role of a given input pattern in the overall computation is clear. A limitation of using cluster analysis is that the pairwise Euclidean distances between vectors of hidden units can, at best, correlate with the computation in the net (recall that they use only half the net). Thus they can give an inaccurate picture of the computational roles of repre-

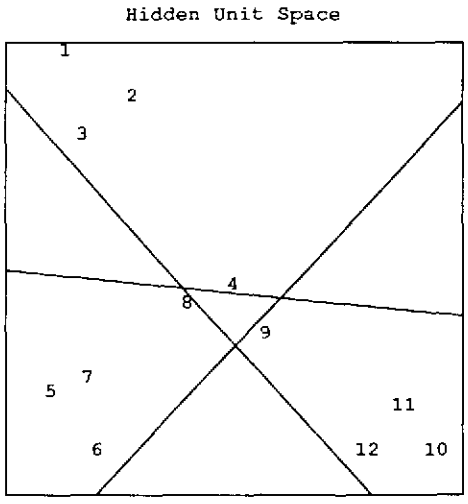


Fig. 13. A decision space for the numbered points shown in Fig. 3.

[6] It should be noted that this does not work for a linear output function on the hidden units. In such a case nothing could be computed with two weight layers that could not be computed with one.

[7] As stated earlier, the output from the multilayer net is determined by a squashing function $f(x)$, and so for drawing the decision boundary a threshold value, θ, must be chosen such that $w_0 + xw_1 + yw_2 + \cdots - nw_n = \theta$. To calculate the value of θ, a criterion value, c, is chosen for the squashing function and used such that the output = 1 if $f(x) \geq c$ and zero otherwise. To determine, θ, the value of c has to be unsquashed, as it were: $\theta = f'(c) = \log(-1 + (1/1 - c))$. This analysis may be extended to continuously valued output units by thinking of each output as implementing a series of decision boundaries each of which reflects a value on the output. The problem for learning then is to find the correct intersections between the boundaries.

sentations and, indeed, the generalisation properties of the net. For example, returning to the hidden unit space in Fig. 3 and the cluster analysis in Fig. 4, the cluster analysis agreed with the visual analysis that there were four main clusters and we speculated on what those cluster might mean. However, an examination of the decision space in Fig. 13 shows how the cluster analysis is only correlated with decision space. The decision space diagram shows that the input patterns are divided among three computational groupings: 1–4, 5–8, 9–12. Thus the correct computational roles of the representations can only be determined by their relation to the output weights; distance measures between the representations can provide only limited information of unknown accuracy.

In summary, we have examined how the computation of networks can be described in terms of decision space, how IH weights transform an input space, and how HO weights project decision boundaries on that space. The advantages of using whole net representations, as opposed to hidden unit representations, to open the black box were discussed. We shall now turn to consider how decision space diagrams can serve as a valuable tool for investigating how, and under what circumstances, a network can adapt previously learned weights in the performance of novel tasks.

3. Weight adaption

With the problems of long training times and the intractibility of scaling networks for very large problems, it is becoming increasingly necessary to set up initial conditions in a network that point the net in the right direction of a solution before training on the task begins. One approach has been to split the task into smaller components or modules (e.g. [29]). Another approach has been to gradually increase the complexity of the input vectors (e.g. [6]). Yet another approach, the one reported here, is to attempt to ascertain the conditions under which positive transfer occurs between one set of weights and a new solution (e.g. [29a]). The basic paradigm is to train a net on one set of associations and then, after training, to train on a second set of associations. According to the manipulations, the second set may either be learned faster than it would be normally, that is called *positive transfer*. Alternatively, it may be learned slower, that is call *negative transfer*.

The results of two studies are reported here (see [29a]) to show the utility of looking at whole net representations in decision space. Both studies make use of paired-associate training materials (see Table 1). There are two steps to determine transfer of weights from a task A to a target task B. First a net is trained on task B and the number of training cycles, β are recorded – this is the baseline measurement. Second a net is initially trained on task A and then the same weights are used for training on task B; the number of training cycles for task B in these circumstances, ρ is also recorded. The latter measure is essentially the *adaptation* time for the weights. In the studies reported here, both β and ρ were averaged over a number of training runs with the initial condition of each net being set up by a different random number 'seed'. When a net is trained on a task from the starting point of a set of prestructured weights (prestructured as a result of earlier training on prior material), either positive or negative transfer can result [8]. The measure of transfer, τ, was normalised as follows:

$$\tau = \frac{\beta - \rho}{\beta + \rho}$$

The coefficient of transfer, $-1 \leq \tau \leq 1$, thus provides a means of quantifying the relationship between tasks on a -1 to $+1$ scale. A τ near $+1$ means that very little learning time was required to adapt, whereas a τ near -1 means that adaption took much longer than standard training.

The idea behind the two studies was to investigate the determinants of transfer. Would varying the inputs result in positive or negative transfer, and would varying the outputs result in positive or negative transfer? In both studies, a 30-2-30

[8] More extensive treatment of the notion of transfer may be found in [29a,29b,29,30].

Table 1
Paired associate materials for Studies 1 and 2

Study 1 Initial	Inputs varied	Study 2 Initial	Outputs varied
EBA-ABC	JGF-ABC	ABC-EBA	ABC-JGF
DEC-BCD	IJH-BCD	BCD-DEC	BCD-IJH
AEB-CDE	FJG-CDE	CDE-AEB	CDE-FJG
CDA-DEA	HIF-DEA	DEA-CDA	DEA-HIF
BDE-EAB	GIJ-EAB	EAB-BDE	EAB-GIJ

net was used. All input and output vectors consisted of 3 vector elements being set to a +1 state and all the rest to zero. Both the input and the output vectors were partitioned into 3 fields with each field providing a localist representation for the letters A to J. The fields were used to preserve order information in the letter strings e.g. the strings AJC, CJA, CAJ, and JCA would all have different vector repesentations. In the first study the output vectors were held constant across both tasks A and B and the inputs were varied. In the second study the input vectors were held constant across both tasks A and B and the outputs were varied. The input and output sets for both studies are shown in Table 1.

The results from the studies were very clear. When the inputs were varied for training the second net (Study 1) the result was a strong positive transfer effect with $\tau = 0.88$ averaged over 20 nets using different random initial conditions for each of the nets used to record β. In contrast, when the outputs were varied for the second net a negative transfer effect resulted with $\tau = -0.04$ (also averaged over 20 nets). The purpose here is to show the usefulness of the decision space method in understanding the cause of the various transfer effects.

Fig. 14(a) shows the decision space for one of the nets used in Study 1 after it had been trained on the *Initial* training set (Table 1, Study 1). The initial inputs in hidden unit space are shown as stars. They are all on the plus side of the appropriate output decision boundaries. Only 15 of the decision lines, one for each positive output A to E in each position, are shown in the diagram. These appear as 5 decision lines because the three lines *cornering* each of the input representations turned out to be identical in every case.

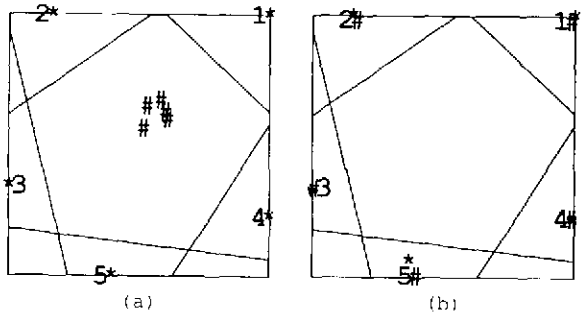


Fig. 14. A decision space diagram showing the effects of retraining a pre-trained net on novel inputs.

The other 15 decision lines not shown, are outside the space to make all of the representations on their zero side. The # points are the new input points on which the net has not yet been trained (the *Outputs varied* in Table 1, Study 1). In Fig. 14(b), the decision space diagram shows that the new points have been learned; they are now on the +1 sides of the appropriate boundaries. The diagram shows that there is no forgetting of the Initial input set and it also shows there was very little movement of the original decision boundaries – this is the cause of the positive transfer.

Negative transfer was obtained, as we have seen, in Study 2, when the outputs were varied from the original task. Fig. 15(a) shows the decision space diagram of the computation performed by the original net, and Fig. 15(b) shows

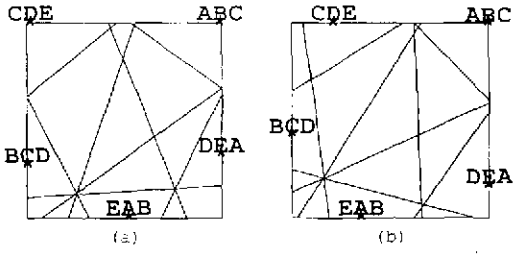


Fig. 15. Decision space diagrams showing positions of inputs in hidden unit space, and arrangement of hidden-output decision vectors for Study 2, in which the same inputs but different outputs were used to retrain the original net. 15(a) corresponds to the decision space developed by the Initial training set, and 15(b) corresponds to the retrained net. Only the relevant decision lines have been drawn: the random decision lines (where the required output is zero) would be located outside the squares). See the text for further details.

the net subsequently trained on a new set of outputs. In both cases there are 15 decision lines in the space (because some of these are identical only 7 are actually visible) and, as before, the other 15 are outside the space.

At first glance, the decision spaces in 15(a) and (b) appear very similar although the decision lines refer to different outputs (see Table 1). In fig. 16 the movement of the decision lines is shown for a selected few. If we look at individual decision lines as shown in Fig. 16, it is apparent that whilst the inputs maintain approximately the same positions in hidden unit space, the output decision lines have been moved. That is, although the overall arrangement of the decision boundaries remains the same, different decision lines have been recruited to take the same positions. Thus in Fig. 16(a) and (b) there is a line that cuts off the top right hand corner. However in (a) this line is the one responsible for outputing an initial 'E' in response to the input 'ABC', and in (b) the decision line is responsible for producing the output of an initial 'J'. The previously used 'E-' decision line has been moved out of the hidden unit space, to the right of the square.

In sum, by virtue of the use of decision space it is possible to ascertain the effect of varying the outputs or inputs of a net. As a result, our understanding of transfer effects and weight adaption is furthered. It is apparent that changing the outputs causes a distortion of the HO weights, and hence the HO decision boundaries (resulting in negative transfer), whilst the IH weights and the positions of the inputs in hidden unit space remains relatively constant. In changing the inputs only the IH weights are moved and the HO weights and concomitant decision lines remain constant (resulting in positive transfer). In the light of these illustrations it would be hard to maintain the position that the computation being

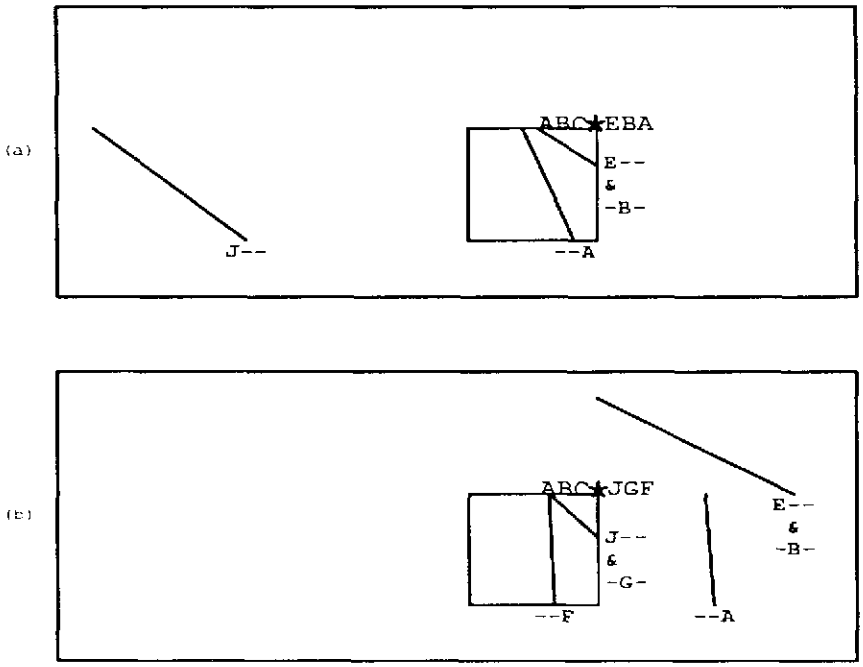


Fig. 16. Decision space diagrams showing movement of decision lines. In particular, it is possible to see the movement of the decision lines associated with output letter J, trained to output a zero for all inputs in the Initial training condition. The labelled star shows the position of one of the input trigrams (ABC) and its corresponding output trigrams across the different conditions. The dashed labels for the decision lines (e.g. E--, -B-, --A) indicate the position of the letter in its output trigram. See the text for details.

performed was in principle impenetrable and univestigable.

4. General conclusions

We have shown here that there is more to connectionist computing than treating networks as a black box technology. By opening the black box, there is much to be gained in utilising some of the novel properties of network computation. Cognitive Science research on internal unit representations was reviewed in Section 1. This revealed that a considerable and increasing research program has uncovered many of the important properties of hidden unit representations and has shown how these may be used in computations that are internal to the net. Such computations do not require symbolic intervention and are of the essence of connectionist computing. However, it was suggested here that the analysis of hidden unit representations provides only part of the story of how and what a network computes. In Section 2, a simple geometrical method for visualising computational space was presented. This makes it possible to inspect the entire computation of a multilayer network. The decision boundaries through representation space determine precisely which input points will output a +1 on a given output unit and which will not. Thus it is clear how the network will generalise to novel inputs. More importantly however, the decision space diagrams enable us to decide how best to control and adapt the global computational properties of connectionist networks as illustrated in Section 3.

The work reported here shows the beginning of the development of a theory of connectionist applications. This comes somewhere in the sandwich between the pure formal theory of connectionist nets and connectionist applications. The formal theory tells us much about the universality and limitations of particular learning algorithms (e.g. [31]) or about the general tractibility of network configuration (e.g. (19]). We can consider such research to be about the general *competence* of neural nets. What we are trying to begin here (and elsewhere e.g. [28]) is the development of a *performance* theory of connectionism; one that can be useful for particular modelling applications.

References

[1] A. Baldwin, Subsymbolic inference: Inferring verb meaning, in: R. Trappl, ed., Proc. *Eleventh European Cybernetics and Systems Conf.*

[2] D.S. Blank, L.A. Meeden, and J.B. Marshall, Exploring the symbolic/subsymbolic continuum: A case study of RAAM, in: J. Dinsmore, ed., *The Symbolic and Connectionist Paradigms: Closing the Gap* (Lawrence Erlbaum, London, 1992).

[3] D.J. Chalmers, Syntactic transformations on distributed representations, *Connection Sci.* 2(1) (1990) 53–62.

[4] L. Chrisman, Learning recursive distributed representations for holistic computation., *Connection Sci.* 3(4) (1991) 345–366.

[5] G.W. Cottrell, A connectionist approach to word sense disambiguation. Phd Thesis, TR154, Department of Computer Science, University of Rochester, NY, 1985.

[6] J.L. Elman, Representation and structure in connectionist models, CRL Technical Report 8903, Center for Research in Language, University of California, San Diego, CA, 1989.

[7] B. Everitt, *Cluster Analysis* (New York, Halstead Press, (1980)).

[8] J.A. Fodor and B.P. McLaughlin, Connectionism and the problem of systematicity: why Smolensky's solution doesn't work, *Cognition*, 35 (1990) 183–204.

[9] T. van Gelder, Compositionality: A connectionist variation on a classical theme, *Cognitive Sci.* 14 (1990) 355–384.

[10] T. van Gelder, Defining distributed representation, *Connection Sci.* 4(3, 4) (1992) 175–192.

[11] S.J. Hanson and D.J. Burr, What connectionist models learn: learning and representation in connectionist networks, *Behavioral and Brain Sci.* 13(3) (1990) 471–518.

[12] G.E. Hinton, Connectionist learning procedures, *Artificial Intelligence*, 40 (1989) 184–235.

[13] G.E. Hinton, J.L. McClelland, and D.E. Rumelhart, Distributed representations, in: D.E. Rumelhart, and J.L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1986 Vol I: Foundations (Cambridge, MA MIT Press,) 77–109.

[14] J.S. Judd, Neural design and the complexity of learning (MIT Press; Cambridge, MA, (1990).

[15] R.R. Leighton, The Aspirin/MIGRAINES Software Tools User's Manual, The MITRE Corp. Technical Report MP-91W00050, 1991.

[16] R.P. Lippman, An introduction to computing with neural nets, *IEEE ASSP* (April 1987), 4–22.

[17] L. Niklasson, N.E. Sharkey, Connectionism and the issues of compositionality and systematicity, in R. Trappl, ed., *Cybernetics and Systems* (Dordrecht, Kluwer Academic Press, 1992).

[17] (a) L. Niklasson and N.E. Sharkey (1994) The systematicity and generalisation of connectionist compositional representations, in: G. Dorffner, ed. (Chapman & Hall).

[18] N.J. Nilsson, *Learning Machines* (McGraw-Hill, New York, 1965).

[19] D. Partridge, *Engineering Artificial Intelligence Software* (Intellect Books, Oxford, 1992).

[20] J.B. Pollack, Recursive distributed representations, *Artificial Intelligence*, 46 (1990) 77–105.

[21] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations* (MIT Press, Cambridge, MA, 1986) ch. 8.

[22] M.F. St. John and J.L. McClelland, Parallel constraints satisfaction as a comprehension mechanism, in: R. Reilly and N.E. Sharkey eds., *Connectionist Approaches to Natural Language Processing* (Hove; LEA, 1992).

[23] D. Servan-Schreiber, A. Cleeremans and J.L. McClelland, Encoding sequential structure in simple recurrent networks, TR CMU-CS-88-183, Computer Science Department, Carnegie-Mellon University, 1988.

[24] A.J.C. Sharkey, Connectionism, nativism and language acquisition, *Proc. Child Language Seminar*, Plymouth (1993).

[25] N.E. Sharkey, Connectionist representation techniques, *AI Rev.*, 5 (1991) 143–167.

[26] N.E. Sharkey, Functional compositionality and soft preference rules, in: B. Linggard and C. Nightingale, eds., *Neural Networks for Images, Speech and Natural Language* (London, Chapman and Hall, 1992) 235–255.

[27] N.E. Sharkey, S.A. Jackson and D. Partridge, An internal report for connectionists, in: R. Sun & L. Bookman, eds., *Computational Architectures Integrating Neural and Symbolic Processes* (Kluwer, Dordrecht, 1993).

[27] (a) N.E. Sharkey and S.A. Jackson, Three horns of the representational trilemma, in: V. Hovanar & L. Uhr eds., Integrating Symbol Processors Connectionist Networks for Artificial Intelligence and Cognitive Modelling (Academic Press, New York, 1994).

[28] N.E. Sharkey and D. Partridge, The statistical independence of network generalisation: an application in software engineering, in: M.J. Taylor and P.G. Lisboa, eds., *Neural Networks: Techniques and Applications* (Ellis Horwood, Chichester, UK, 1992) 21–23.

[29] N.E. Sharkey and A.J.C. Sharkey, A modular design for connectionist parsing, in: Connectionism and Natural Language Processing, M.F.J. Drosaers and A. Nijholt, eds., *Proc. Workshop on Language Technology* 3, Twente (1992) 87–96.

[29] (a) N.E. Sharkey and A.J.C. Sharkey, Adaptive generalisation and the tranfer of knowledge, *AI Rev.*, Special Issue on Connectionism (1994).

[29] (b) N.E. Sharkey and A.J.C. Sharkey, Emergent cognition, in: J. Hendler, ed., *Handbook of Neuropsychology, Vol. 9: Computational Modeling of Cognition* (Elsevier Science, Amsterdam, The Netherlands, 1994).

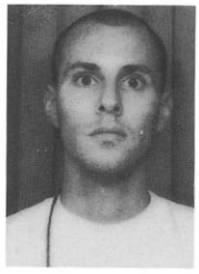
[30] A.J.C. Sharkey and N.E. Sharkey, Connectionism and natural language, in: *Grammatical Inference: Theory, Applications and Alternatives*, Electronics Division, Institute of Electrical Engineers, Digest No. 1993/092.

[31] H. White, *Artificial Neural Networks: Approximation and Learning Theory* (Blackwell Publishers; Oxford, UK).

[32] A.P. Wieland, Geometric analysis of neural network capabilities, *Proc. IEEE Int. Conf. Neural Networks*, Vol. 3 (1987) 385–392.

Noel Sharkey received his Doctorate in Experimental Psychology from the University of Exeter, UK, in 1982. Since then he has worked as a research associate in Computer Science at Yale University, USA, and as a senior research associate in Psychology at Stanford University, USA, where he has also served as a visiting assistant professor. In 1984 he took up a 'new blood' lecturship (English equivalent to assistant professor) in the Department of Language and Linguistics at Essex University, UK. His next appointment was back in Exeter as a Reader in Computer Science. He is currently employed as a Professor of Computer Science at the University of Sheffied. He is author of a number of papers and technical articles as well as being the main Editor of the journal Connection Science.

Amanda Sharkey is currently employed as a Research Fellow and Honorary Lecturer in the Department of Computer Science at the University of Sheffield. Her career has a multidisciplinary flavour; she has held posts in Computer Science, Psychology and Linguistics. Before obtaining her Ph.D. in 1989 from the University of Essex, she worked as a Research Assistant at Yale University, USA, Stanford University, USA, and MRC Cognitive Development Unit, London. Since then she has been employed on research grants at both the University of Essex and the University of Exeter. She has published in the fields of Neurocomputing, Psychology, and Language Development.



Stuart A. Jackson received a Ph.D. in Computer Science from the University of Exeter in 1992 for his work on theoretical connectionism and philosophy of meaning. His B. Sc. was from the University of Sussex in Experimental Psychology. He is currently a Research Fellow in the Department of Computer Science at Sheffield University, where he is engaged in transfer research using recurrent neural networks. His research interests include the philosophy of mind, the theory of neural computation, representation grounding and problems of meaning in cognitive modelling.