

Why Connectionist Learning Algorithms need to be More Creative

Chris Thornton
Cognitive and Computing Sciences
University of Sussex
Brighton
BN1 9QH
UK

Email: Christopher.Thornton@firenet.uk.com
Tel: (44)1273 678856

May 21, 2003

1 Do connectionist systems really build representations?

The topic of creativity is an important one within connectionism. In general, connectionist systems are only as powerful as the learning algorithms they employ and these often need to ‘creatively’ construct internal representations. Of course, some researchers find the notion of *connectionist representation* hard to deal with. They feel that for something to count as a representation there must be an *agent* who makes explicit use of some system of *symbols* for the purposes of representing *phenomena* in a given *domain*. They see connectionist mechanisms (ie. neural networks) as conglomerations of activity-storing units and activity-passing connections. They understand how this sort of mechanism might perform certain types of computation but they cannot see how it could possibly have anything legitimately termed a representation. Such researchers may therefore be upset by the frequency with which connectionists use the term representation in relation to connectionist mechanisms.

But, in fact, there is strongly suggestive evidence that connectionist mechanisms develop internal structures that can, quite meaningfully be called representations. One of the best known demonstrations was provided by Geoffrey Hinton in his work with the *kinship* problem [1]. Hinton used the back-propagation learning algorithm to train a network to answer queries about two sets of isomorphic, family relationships. The relationships are summarised in

the two family trees shown in the figure below, from [1]. (In this figure, the symbol '=' means 'married to').

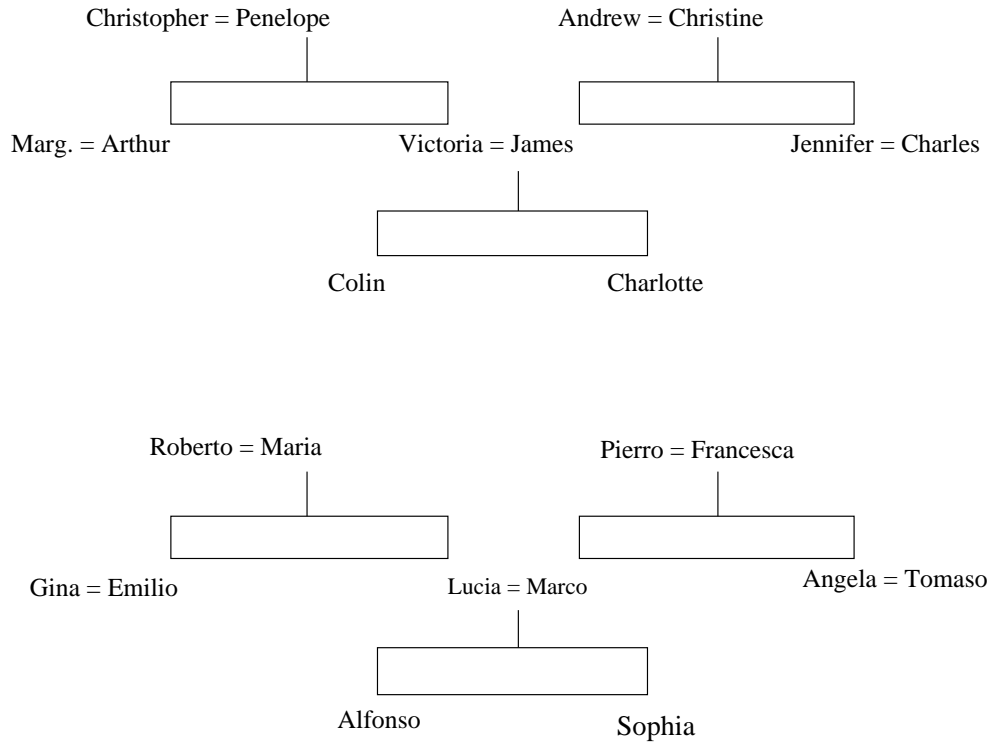


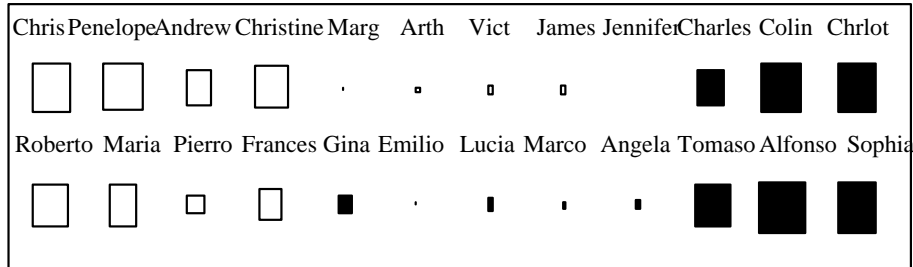
Figure 1:

Queries were of the form **Roberto Father-of ?**, ie. who is the father of Roberto? Using the back-propagation learning procedure [2] and a set of 100 examples, Hinton was able to train a network to produce reliable answer to this sort of query, even in the case where the answer did not appear explicitly in the examples used in the training. But the most striking outcome was not the performance of the network, but the structure of connection weights produced during learning.

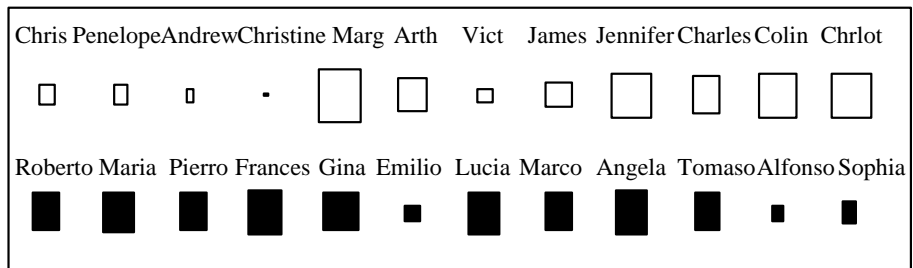
To roughly summarise, Hinton discovered that the training had produced units whose behaviour corresponded closely with *natural kinds* or *features* of the domain. He discovered, for example, a unit in the network that (following training) effectively encoded the nationality of the individual explicitly named in the query, even though no explicit information about nationality appeared anywhere in the training set of examples.

In a series of replications of this experiment, I have been able to reliably produce networks containing units that appear to correspond to a variety of

features.¹ Two such units are shown in the form of Hinton diagrams in the figure below.² The figure shows the weights from internal units 25 and 26 to the 24 input units representing the 24 individuals in the domain. By looking at how individuals have been **weighted** within the units we can see that unit 26 has learned to encode the nationality of the indicated individual while unit 25 has learned weights that will cause its behaviour to encode the age of the individual.



Unit 25 "Age"



Unit 26 "Nationality"

Figure 2:

This seems to be a clear-cut case of *representation*. As a result of learning, the network has created an internal structure containing components whose behaviour is correlated with the manifestation of a particular feature (or natural kind) of the domain. The utility of such representations is not difficult to comprehend. Clearly, in attempting to make an inference about who might be in a given relationship with a certain individual, it is extremely helpful to know that individual's nationality or age. For example, if I ask you, who is Jennifer's

¹Certain aspects of this work were carried out by Randall Stark.

²In a Hinton diagram, positive weights are shown as empty boxes, negative weights as filled boxes, while units are shown as enclosing rectangles, see [3].

father, it is helpful to know that Jennifer is English since this enables you to rule out all 12 Italians.

2 Analysing distributed representations

More recent work has produced evidence of connectionist representation of a new and very interesting type. A good example has been described by Jeffrey Elman in his work with the Lexicon network [4]. Elman constructed a network with the aim of finding out whether a connectionist learning algorithm might be able to produce representations for the structural properties of a lexicon and in particular the way in which words are arranged into grammatical categories. The training set was based on a corpus of 10,000 two and three-word sentences. These sentences were grammatically very primitive. However, they were generated so as to reflect semantic properties of the component words. A sample sentence would be a triple such as **man smash plate** — the conjunction of **smash** and **plate** here reflecting the semantic property that plates are things which can be smashed.

After the network had been trained to a satisfactory level of performance, Elman recorded the sequence of internal states of the network that were generated as he presented it with a series of words. He then applied a cluster analysis to the states and produced the dendrogram shown in the figure below.³ Each tip node in the dendrogram represents a particular class of similar internal states and is labelled with the word which was responsible for producing that class of internal states.

The point to note is the way in which the similarity structure among the internal states corresponds to and reflects linguistically meaningful features. For example, note that all the words in the upper, main branch are verbs. All the words in the lower, main branch are nouns. If we look at the structure in more detail we find groupings which correspond to animate nouns, inanimate nouns, and various semantic classes (eg. the class of nouns labelling food).

In this example, then, the learning has produced a network in which there is a direct correspondence between internal states of the network and natural kinds or features of the domain. This would seem to be a representation of a qualitatively different sort but certainly a representation.

3 The other side of the coin

The way some missionaries tell it, connectionism is the answer to all our problems. But of course this is not the case. Connectionist learning procedures are, in fact, quite prone to failure. They can fail *explicitly* — ie. they can fail to learn how to produce the right answers to given queries. Alternatively, they can fail *implicitly* — ie. they can fail to develop any sort of meaningful, internal representation.

³This is an approximation of the dendrogram provided by Elman.

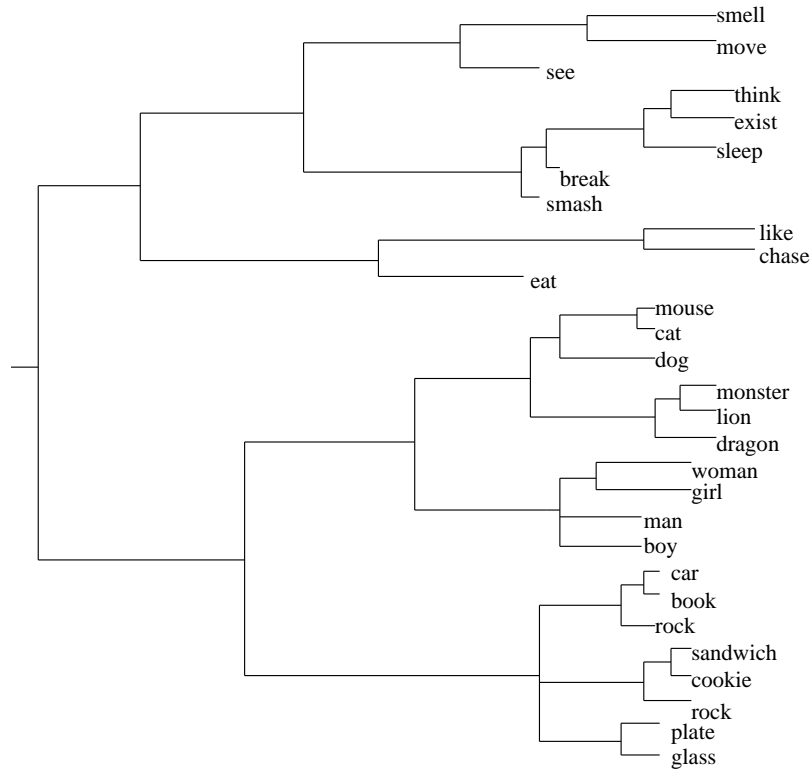


Figure 3:

In fact there appears to be a particular situation in which connectionist learning procedures are almost certain to fail — at least in the implicit sense. This situation occurs when the target mapping that is to be learned is based on the recognition (or exploitation of) a feature that is more than 1st-order, ie. which cannot be defined directly in terms of the primitive attributes appearing in the training examples. A n th-order feature is a function or predicate that can only be defined in terms of $(n-1)$ th-order features. For example, in the kinship domain, a 2nd-order feature might be the predicate

unusually-young(Age, Nationality)

which is true if the value of **Age** is much less than the usual value for an individual of the given nationality. This predicate can only be defined in terms of an age feature and a nationality feature and these are themselves 1st-order features of the kinship domain.

To find out why connectionist mechanisms fail in these ways we have to look in more detail at how connectionist representations work.

4 Representational boundaries

An accessible explanation for the functional properties of connectionist representations has been provided by Lipmann [5]. This is organised around the diagram shown in the figure below. The diagram is based on the observation that the units in neural networks effectively **define** boundaries in the input space. By this we mean that they are strongly active if the network's input vector appears on one side of a given planar boundary in the input space and firmly off if it appears on the other side. The internal structures of a network thus serve to define a set of boundaries in the input space. The more complex the internal structure of the network, the more complex the boundaries that can be defined as a result of learning. As the diagram shows, a network containing a single output unit effectively defines a single hyperplane in input space. As we increase the number of layers the complexity of the regions that can be defined increases. With three (arbitrarily large) layers of units we can effectively define any configuration of boundaries and therefore any regions we like. This allows us to learn an *arbitrary* mapping from input to output.

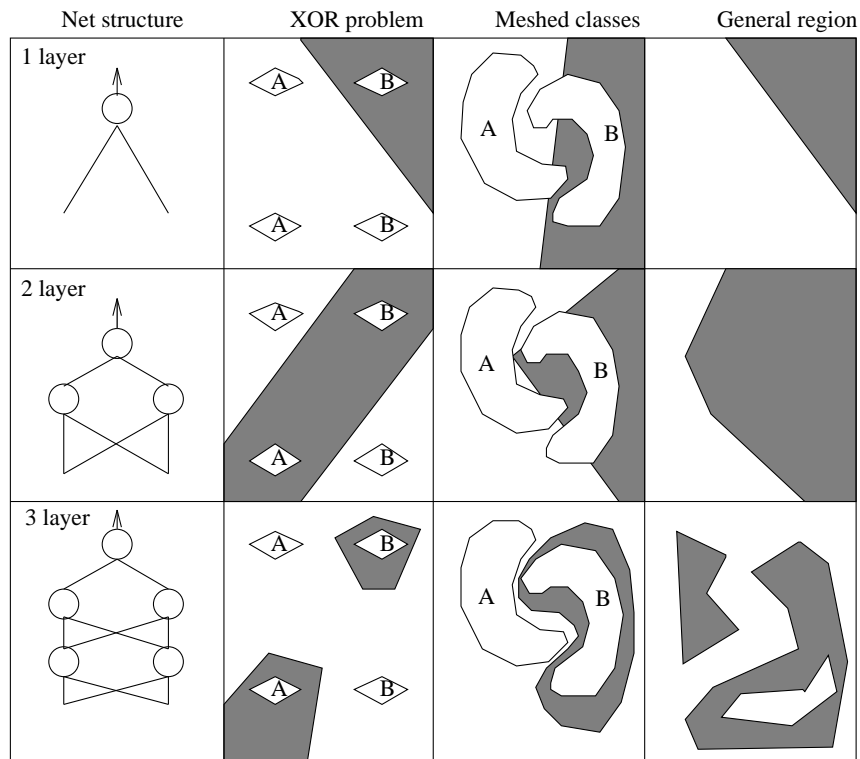


Figure 4: After Lipmann, 1987.

What this all boils down to is the observation that, given a sufficient amount

of internal architecture, neural networks can effectively learn any mapping we like [6]. From here it is a fairly short jump to the gleaming conclusion that neural networks can learn to *represent* anything at all. But my suggestion is that this is to go way beyond what is justified.

Where the target mapping is based on zeroth-order features (ie. statistical or attribute-based properties of the training set) then connectionist learning mechanisms usually function very well. Even where the mapping is based on 1st-order features then they may well succeed given favourable parameter settings and network structures. But when we are dealing with higher-order features (eg. **unusually-young**) they usually fail — at least in the implicit sense defined above. They may successfully form an implementation of the target mapping but the underlying representation is quite likely to have the form of a *kludge*. Let me use an example to explain what I mean.

Consider the following training set. Each training pair appears on a separate line. The input (eg. [0.42 0.62]) comes first and the target classification (either positive or negative) comes second. Before reading on see if you can determine the rule that underlies the mapping.

```
[0.42 0.62] +
[0.64 0.59] -
[0.87 0.13] -
[0.64 0.84] +
[0.07 0.86] -
[0.37 0.57] +
[0.24 0.44] +
[0.76 0.96] +
[0.37 0.96] -
[0.64 0.84] +
[0.08 0.65] -
[0.99 0.37] -
[0.41 0.61] +
[0.14 0.78] -
[0.23 0.43] +
[0.73 0.56] -
[0.89 0.57] -
[0.59 0.26] -
[0.25 0.45] +
[0.34 0.54] +
```

As you may have been able to work out, an input is a positive if its second component exceeds its first component by exactly 0.2. We can draw this training set out as a diagram by treating the input values as the coordinates of points. This produces a diagram in which each positive input corresponds to a point labelled + and each negative input corresponds to a point labelled -. The diagram appears in the right half of the figure below.

Using the back-propagation learning procedure and a network containing just two hidden units, we can deal with this training set quite successfully. The

weights and unit biases produced in one successful run are shown in the left half of the figure here. Examination of the weights of the two hidden units shows that the first hidden unit (unit 3) defines the hyperplane shown in the figure as a line of 3s while the second hidden unit (unit 4) defines the hyperplane shown as a line of 4s. Clearly these two hyperplanes have been **positioned** along the edges of the region of positives. If we look at the weight values in conjunction with the hyperplanes we can see that unit 3 is effectively turned on only by inputs falling above and to the left of its hyperplane. Unit 4 behaves the same way. Thus by arranging for unit 3 to strongly inhibit the output unit and for unit 4 to strongly excite it, back-propagation has obtained the desired behaviour from the output unit. The geometric analysis shows clearly that the

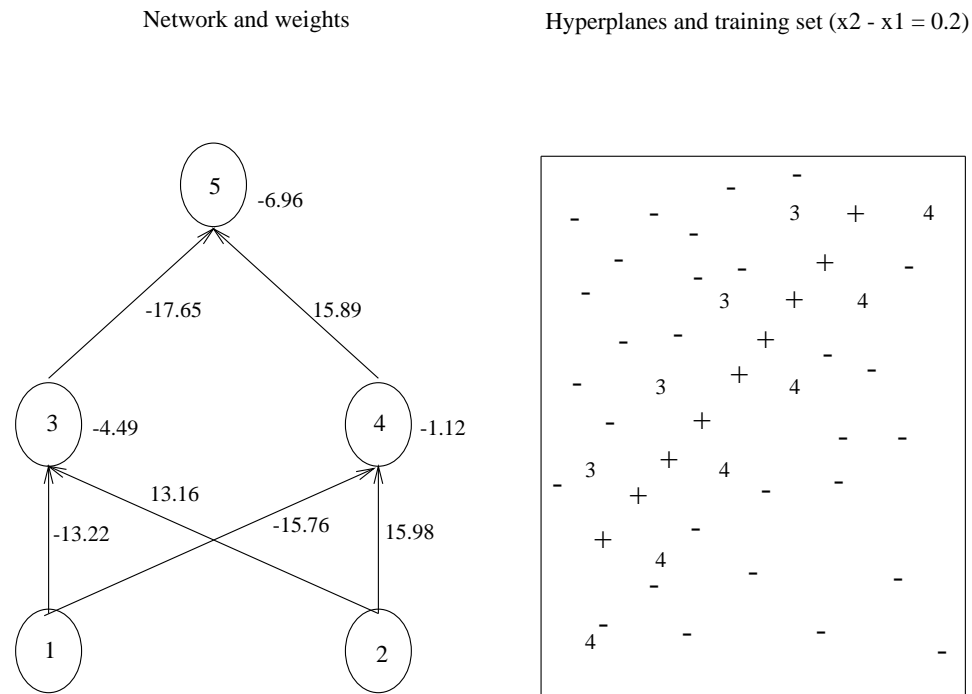


Figure 5:

representation required for this target mapping is a fairly straightforward one that needs a minimum of two hyperplanes (ie. two hidden units). We will now look at a more complex problem that requires a more elaborate arrangement of hyperplanes. The initial pairs in the training set for the problem are shown below.

[0.4 0.4] +
 [0.7 0.5] -

```

[0.18 0.9] +
[0.8 0.7] -
[0.4 0.4] +
[0.8 0.6] -
[0.1 0.4] +
[0.9 0.8] -
[0.07 0.7] +
[0.5 0.2] -
[0.2 0.6] +
[0.7 0.3] -
[0.2 0.2] +
[0.4 1.0] -
[0.3 0.9] +
[0.1 0.9] -
[0.4 0.8] +
[0.05 0.4] +
...

```

The rule here is that an input is a positive if its first component cleanly divides its second component. Geometrically the training set is related to the previous training set in the sense that the regions of positives tend to radiate out from the origin of the input space. However it is more complex since there are many more regions to be taken into account. If we try running back-propagation on this problem we find that it eventually finds a solution in which the hidden units track the relevant boundaries between regions of positives and negatives quite accurately. A representation that was produced in a network using 10 hidden units is shown in the figure below. In the upper, left part of the figure we see the network structure used, in the upper right part we see the geometric representation of the training set and in the lower part we see the representations for the boundaries defined by the hidden units (numbered from 3 to 10). Here back-propagation has produced an internal representation that is adequate for the purposes of generating satisfactory classifications. But it is questionable whether it really captures the regularities underlying the training set. The target mapping is based on the concept of *division* but none of the mathematical concepts that one might expect to be utilised in a representation of division are evidenced in this configuration of boundaries. Putting it another way, there is nothing about the arrangement of boundaries that back-propagation has constructed that would be of much use in dealing with a different, division-related problem (eg. deciding whether one number divides another number a certain number of times). The general implication seems to be that this representation, while being quite satisfactory for the given target mapping has the nature of a *kludge* since it is not in any sense based on the natural kinds, features or concepts that seem to be relevant with respect to the process of division.

A further complication of the problem used here can be introduced by saying that an input is a positive provided that its first element divides the second element or the complement of the second element (ie. the remainder when it is

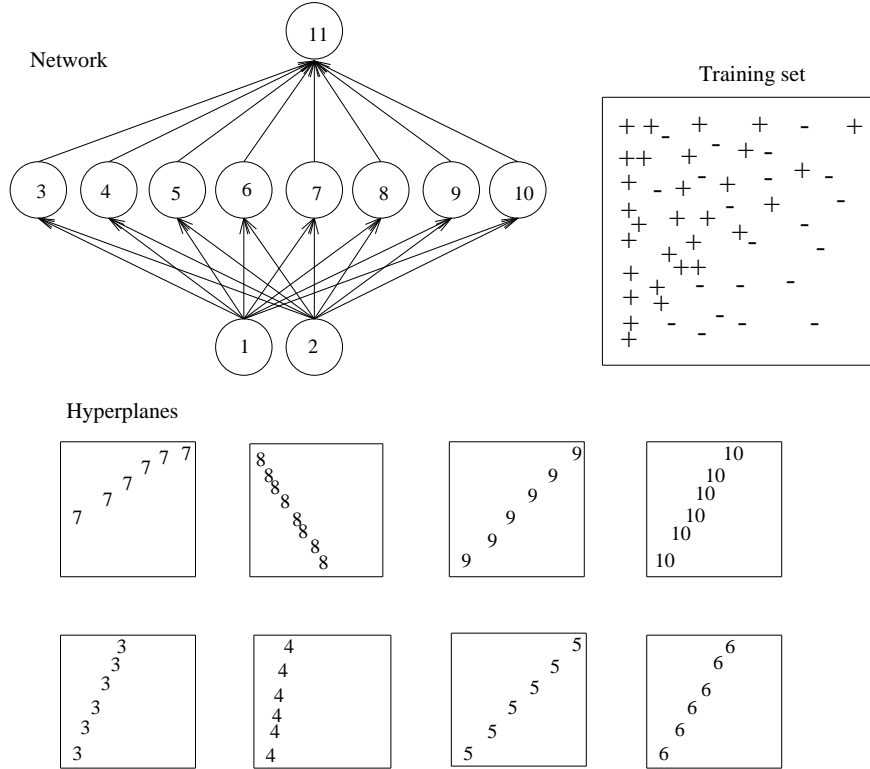


Figure 6:

subtracted from 1). In this case the positive regions radiate out from both the bottom, left corner of the input space and the top, right corner. In principle back-propagation can produce a satisfactory implementation of this mapping using the same strategy as before. However, the fact that the regions of positives and negatives actually overlap in this example means that, in practice, back-propagation cannot solve this problem using a reasonable number of hidden units (ie. < 20) in a realistic amount of training time.

5 Summary and concluding comments

To summarise, if we try to make a neural network learn a mapping that is based on higher-order features, at least three things can happen. A possible outcome is that the network will successfully learn the mapping. Rather more likely is that the network will successfully learn the mapping but only by producing a *kludged* representation, ie. a complex, bird's nest of boundaries that offers minimal generalisation and compression. The third possible outcome is that

the network will fail to learn the mapping completely, virtually regardless of the amount of training and internal architecture we make use of. Of course all the trials discussed here were conducted using just one learning algorithm (back-propagation) together with **experimenter-selected** parameter values. It is possible — though unlikely in my opinion — that with a different architecture or approach, different results might have been obtained.

The overarching aim of the paper has been to suggest that although connectionist learning mechanisms such as back-propagation can often form internal representations enabling satisfactory performance on higher-order mappings, these representations may well have the character of kludges. As they are typically not expressed in terms of the concepts and features that are pertinent to the regularities underlying the mapping, they are usually ineffective with respect to similar or closely related mappings. To improve on learning mechanisms such as back-propagation, it will be necessary, I feel, to introduce a creative aspect into the learning; ie. to devise mechanisms for producing required higher-level concepts and features within the learning process. This, unfortunately, turns out to be an extremely difficult task⁴ on which much more work is needed.

References

- [1] Hinton, G. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40 (pp. 185-234).
- [2] Rumelhart, D., Hinton, G. and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323 (pp. 533-6).
- [3] Hinton, G. and Sejnowski, T. (1986). Learning and relearning in boltzmann machines. In D. Rumelhart, J. McClelland and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition. Vols I and II* (pp. 282-317). Cambridge, Mass.: MIT Press.
- [4] Elman, J. (1989). Representation and structure in connectionist models. CRL Technical Report 8903, San Diego: Center for Research in Language (UCLA).
- [5] Lipmann, R. (1987). An introduction to computing with neural networks. *IEEE ASSP Magazine*, 4 (p. 14).
- [6] Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem theorem,. *Proceedings of IEEE First International Conference on Neural Networks*. Vol. 3, (pp. 11-14). San Diego.

⁴It usually goes under the name of *constructive induction*