Psychol Res (1990) 52: 153 – 157

Psychological Research

Are multilayer feedforward networks effectively Turing Machines?

Wilemm J. M. Levelt

Max-Planck-Institut für Psycholinguistik, Wundtlaan 1, Postbus 310, 6525 XD Nijmegen, The Netherlands

Summary. Can connectionist networks implement any symbolic computation? This would be the case if networks have effective Turing machine power. It has been claimed that a recent mathematical result of Hornik, Stinchcombe, and White on the generative power of multilayer feedforward networks has that implication. The present paper considers whether this claim is correct. It is shown that finite approximation measures, as used in Hornik et al.'s proof, are not adequate for capturing the infinite recursiveness of recursive functions. The result is therefore irrelevant to the issue at hand.

There can be little doubt that connectionism is interested in learnability and knowledge representation. In fact, there is hardly a more central concern in connectionism than to show that any complex domain of knowledge can be acquired by PDP networks. Given this preoccupation, it is surprising that almost no effort is spent on a formal analysis of these issues. In fact, the standard approach is quite unprincipled, namely by example and computer simulation. In dealing with some domain of cognitive functioning such as syntax, the strategy is to try to teach a network some more ore less interesting syntactic set. If it works, the conclusion is drawn that the result suggests that PDP networks can learn syntax.

Such generalizations, however, are totally unwarranted. Those who have more than 30 years of memory available may remember how the same kind of argument blossomed and perished when language was modelled as a Markov process. It was never taken to be alarming that there were long-distance dependencies in language, dependencies that could bridge several intervening elements. The solution was to increase the order of the approximation accordingly; or in terms of finite automata: to have states relate to ordered pairs, or triples, or quadruples of previous in- or output elements. There was always hope, until Chomsky proved that the generative power of finite automata was *in principle* insufficient for the generation of natural languages. A recent paper by Servan-Schreiber, Cleeremans, and McClelland (1988) deals with long-distance dependencies and their representation in recurrent nets. It is *l'histoire qui se repète*. Long-distance dependencies can be represented in recurrent networks by making the states also reflect the *history* of the previous output. It is still the pre-Chomskian style of argument after all these years.

What is needed for network processing is something akin to what automata theory is for symbolic processing. One should prove what kinds of functions can be computed by different kinds of automata. The theorems of automata theory have been of foremost importance for the theory of symbolic computation. Not only did they make it feasible to determine the simplest architectures for the representation of different kinds of knowledge, such as the push-down automaton for context-free grammars. But in addition they made it possible to define and proof learnability for various triples of architecture, knowledge domain, and presentation schedule (cf. Levelt, 1974, and numerous subsequent publications).

Formally, connectionism is in exactly the same ball park. It makes little sense to spend years implementing a domain of knowledge in a network that cannot contain it. It makes even less sense to study learnability in such cases. In fact, this formal approach was that of the pioneers Minsky and Papert, who proved which functions their perceptrons could or could not compute.

In the present paper I shall consider whether a recent proof on the generative power of connectionist networks carries the implication that networks can represent any symbolic computation.

Hornik et al.'s result

Recently Hornik, Stinchcombe, and White (1989) proved that multilayer feedforward networks with one hidden layer are capable of approximating any Borel-measurable[1] function to any desired degree of accuracy, provided that enough hidden nodes are available.

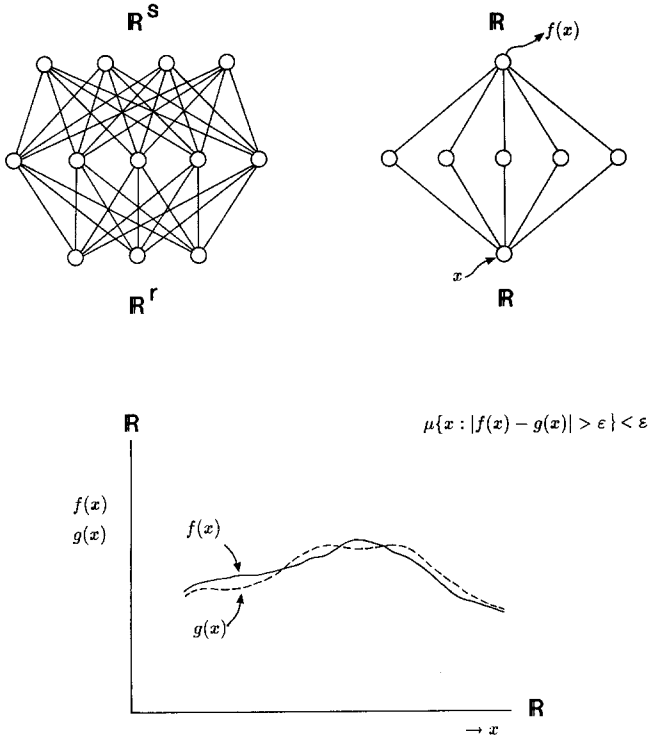

Fig. 1. Feedforward multilayer networks with several input and output nodes (top left) or with single input and output nodes (top right). At the bottom: $g(x)$ is the function to be approximated, and $f(x)$ the approximating function computed by the network, with μ as the measure of approximation.

This is an important result. It shows that multilayer networks can indeed compute an impressive class of functions. It also led to immediate euphoria. Elman (1989), for instance, wrote: "Put simplistically, they are effectively Turing machines. In principle, then, such networks are capable of implementing any function that the Classical system can implement." And surely, if multilayer feedforward networks are effectively Turing machines, they can implement any explicit symbolic computation, since we know that Turing machines can.[2] These networks would then form a universal language for symbolic computation, and this would undermine recurrent claims that they are *in principle* incapable of certain kinds of symbolic computation.

In order to find out whether Hornik et al.'s result has this implication we shall have to look a bit more carefully into what the authors have proven. Their final theorem concerns networks with r input units and s output units. The functions therefore map real-valued vectors in R^r to real-valued vectors in R^s (see Figure 1, top left).

For the sake of simplicity, however, I shall discuss Hornik et al.'s result by way of examples in *RxR* (Figure 1, top right), i.e., for networks with one-dimensional input and output vectors. The theorem says that for any measurable function g and any arbitrary ε there is a network that produces a function f such that the difference metric of f and g is smaller than ε (Figure 1, bottom). This difference metric will turn out to be important, and needs some further comment. The authors take the probability ε that $f(x)$ and $g(x)$ differ by more than ε (i.e., "significantly"), and they require that that probability (of a significant difference) is smaller than ε. So there may be occasional big differences between f- and g-values, but it shouldn't occur too often. How can one talk about the "probability" that a big difference occurs? That presupposes that we know the probability distribution of x occuring as input to the network. It doesn't matter when f and g differ more than ε for input values that are rarely presented to the network. It is, however, not essential that the difference metric is a probability measure. Any other *finite* measure will do.

The heart of the matter is, of course, in the construction of this approaching function f. A hidden node does two things. It adds the weighted activations from the input nodes. That, however, is not relevant for the single input node/output node network in Figure 1; nor is it relevant for the proof. The other thing the hidden node does is to perform a nonlinear transformation of input activation into output activation. This nonlinearity is essential, but almost any nonlinearity will do, as Stinchcombe and White (1989) have shown. The output node, finally, adds the weighted activations it receives from the hidden nodes. So it produces the sum given in Figure 1. Given a finite difference metric and enough hidden nodes, this sum can approach any measurable function, and most of the proof goes in showing that this is the case.

The approximation of discrete recursive functions

Let us now turn to the question of whether these networks are also effectively Turing machines. Turing machines can recognize all and only the recursively enumerable sets, also called the type-0 languages. Recognizing means that for each element in the set, say each grammatical sentence if the set is a language, the machine will halt in a final state, i.e., it will say "yes, this is a grammatical sentence" after only a finite sequence of moves or transitions. When it receives a string that doesn't belong to the set, it may say "no, this doesn't belong to the set," but it may as well run forever without producing an answer. The latter case is unpleasant in the case of natural languages, because you can never know whether the machine is dealing with a very complicated grammatical sentence that just takes a long time to check, or whether the machine is running on forever on an ungrammatical string, an element in the complement of the set. Levelt (1974) and others have argued that this cannot be a good model for natural lan-

[1] Let B be the smallest collection of subsets of the real numbers R such that B contains all the closed intervals $\{x | a \leq x \leq b\}$ and satisfies the conditions: (i) The empty set belongs to B; (ii) For all $A \in B$ the complement of A belongs to B; (iii) For any finite or countably infinite subset A of B the union $\cup_{A \in A} A$ belongs to B. The elements of B are called the *Borel sets* of R. A real valued function f on R is called a *Borel measurable* or *Baire function*, if for all $a \in$ R the set $\{x | x \in \mathrm{R}, f(x) < a\}$ belongs to B.

[2] More precisely: in the course of the last half-century the question whether there is a Turing machine for any explicit or effective symbolic computational procedure has been answered by *defining* an effective procedure by one that can be Turing-computed.

Table 1. A vocabulary V, a recursive grammar G, an enumeration of strings of increasing length over V, and the characteristic values of these strings

V = {John, Peter, went, and}
G = S → N + went
N → N and N
N → John, Peter

String		Characteristic value
1	John	0
2	Peter	0
3	went	0
4	and	0
5	John John	0
6	John Peter	0
7	John went	1
8	John and	0
9	Peter Peter	0
10	Peter went	1
11	Peter and	0
12	Peter John	0
13	went and	0
	etc.	etc.
21	John Peter went	0
22	John Peter and	0
	etc.	etc.
85	John Peter went and	0
86	John and Peter went	1
	etc.	etc.
341	John Peter went and John	0
342	John John Peter went and	0
	etc.	etc.

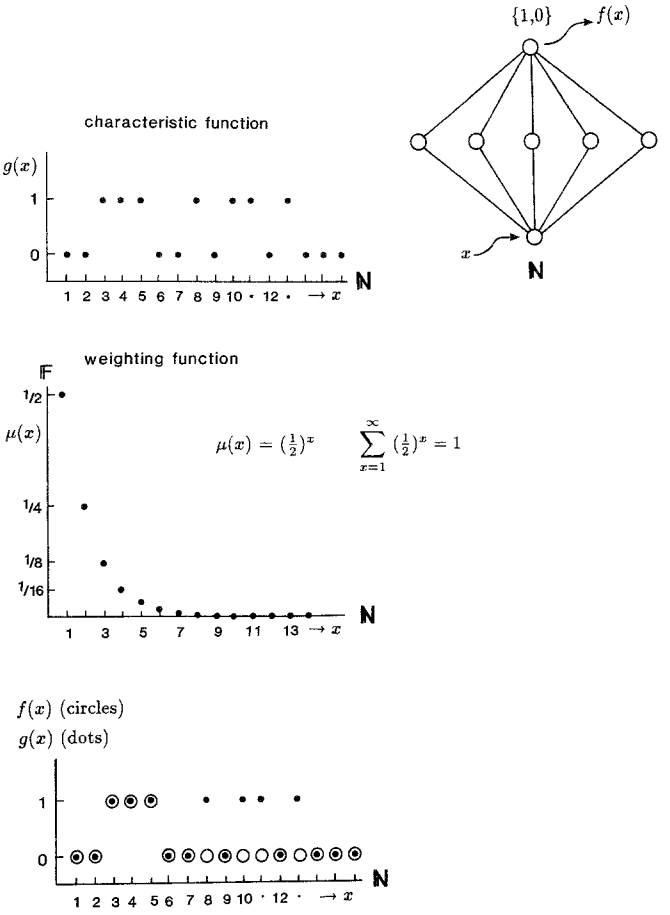

Fig. 2. Feedforward multilayer networks with several input and output nodes (top left) or with single input and output nodes (top right). At the bottom: $g(x)$ is the function to be approximated, and $f(x)$ the approximating function computed by the network, with μ as the measure of approximation.

guages. Language users typically have as strong intuitions about the grammaticalness of strings as about the ungrammaticalness of strings. So natural languages are not just *recognizable*, they are probably also *decidable*. This means that the Turing machine should always produce either a "yes" or a "no" answer after a finite number of transitions. Such sets or languages are called *decidable* or *recursive*. Among them are all context-free and all context-sensitive languages. I shall now limit the discussion to these recursive sets, because – for the sake of exposition – I shall argue from linguistic examples. But if the conclusion holds for recursive sets, it holds for recursively enumerable sets as well, precisely the sets for which Turing machines can be recognizers.

A recursive set is characterized by a recursive function. And the question we have to ask is whether networks can be approximators of recursive functions, as they are of measurable functions. Each recursive language has a *characteristic function* that is recursive. One can, obviously, enumerate all strings that can be composed from the language's vocabulary. Just begin enumerating all 1-word strings, then all 2-word strings, etc. Table 1 presents a small recursive grammar over a 4-word vocabulary and (the beginning of) an enumeration of the strings that can be composed out of this vocabulary.

There will be an infinite number of such strings. Another way of saying that the strings can be enumerated is that one can assign natural numbers to them: string 1, string 2, etc. (see Table 1). The characteristic function for a recursive set or language assigns the value 1 or 0 to each of these string numbers. The value is 1 if the string is in the language; it is 0 if the string does not belong to the language. Table 1 presents these characteristic values for the recursive set defined by grammar G. String 1 has characteristic value 0, string 2 has characteristic value 0, etc., string 7 has characteristic value 1, and so on. More generally, the characteristic function for a language is a mapping from N to $\{1, 0\}$.

Remember that the right-hand network of Figure 1 computes a function from R to R. We are now considering a special case: a function that is defined for every natural number, but for none of the reals in between. That function is also measurable in the Borel sense. Hence, the Hornik et al. (1989) theorem entails that for any recursive set's characteristic function and each ε there is a network with one input and one output node that approximates the characteristic function to a degree $< \varepsilon$. You put in the number of the string, and it appropriately produces a 1 or a 0 as the case may be.

So is it correct to conclude that for each discrete recursive function there is a network that can approach it to any degree of accuracy? No, it isn't. We shall have to look very

Table 2. A recursive grammar and strings of increasing recursion it generates

G = S → if N says S, he is lying
S → N says S
S → it is raining
N → John, Mary, Peter

Examples of generated strings, with degree of recursion

Degree	Examples	Weight
1	if John says it is raining, he is lying	μ_1
2	if John says Mary says it is raining, he is lying	μ_2
3	if John says Mary says Peter says it is raining, he is lying	μ_3
	etc.	etc.
	For $\mu_1 = \mu_2 = \mu_3 = \ldots$	$\Sigma\mu_i = \infty$

carefully into what it might mean to approximate a recursive function in the sense defined by Hornik et al. (1989).

As we saw above, we need a *measure* of approximation between the target function and the approximating function. That (dis)similarity metric should be a finite measure (such as a probability measure, where the total probability of the outcomes is 1).

Let us, by way of example, construct such a case for the characteristic recursive function g in Figure 2. We take as a measure the function μ on NxF, where $\mu(x) = (1/2)^x$. It is also shown in Figure 2. The sum of this infinite progression is 1, i,e., finite. The function μ can be used as a measure for the approximation of two characteristic functions. If the functions differ by more than ε for $x = 1$, that will contribute $1/2$ to the difference metric; if they differ more than ε for $x = 2$, that will contribute $1/4$ to the difference metric; et cetera. Hence the maximal difference between any two such functions is 1; that is the case when their values differ by more than ε for every x.

How are we going to construct an approximating function f, given an arbitrary $\varepsilon \leq 1$? (For $\varepsilon > 1$ any characteristic function on N will do because $|g(x) - f(x)| < \varepsilon$ for all x). That is relatively easy. We will construct an f that is a perfect fit to the first n values of g, where n is chosen in such a way that $\Sigma(1/2)^x \geq 1 - \varepsilon$. In this way the weight of the remaining differences can never reach the value ε. Hornik et al. (1989) showed that such an f can be constructed with a network containing n hidden nodes. But it is irrelevant what f does for numbers greater than n. Even if it differs from our characteristic function g for all subsequent numbers, the total weight difference of the two functions will be smaller than ε.

In short, if g is the characteristic function to be approximated, and ε the desired measure of approximation, then take the following steps: (i) define a finite difference metric, (ii) take a finite set of points $\{a_1, a_2 \ldots a_n\}$ such that for the remaining set of points the difference metric will be smaller than ε for any approximating function, (iii) create a network with n hidden nodes that computes a function f which is an exact representation of g on the domain $a_1 \ldots a_n$ (this is always possible). Then f is an approximation of g in the sense of Hornik et al. (1989). So indeed, recursive functions can be approximated by multilayer feedforward networks in the sense defined by Hornik et al. (1989).

But is this what we were after? The issue was whether connectionist nets are effectively Turing machines, i.e., mechanisms that can *ipso facto* represent natural languages, or perform any explicit symbolic computation. What has been shown, however, is that for any recursive language (or set) there is a network with n hidden nodes that can exactly represent the characteristic values (0 or 1) of n strings. The idea of the approximation is to choose n large enough so as to make the cumulative weight of the discrepancies for all other values smaller than ε. In other words, the network fools us by correctly replicating a finite set of characteristic values *that we are supposed to value highly*, but it can miss all the other values. It is therefore precisely the unlimited recursion of these sets that is not captured in this way. The classical computational models of mind were designed to account for this very *unlimited productivity* of symbol systems.

So how did it come about that we got ourselves fooled by the network? How did it occur that it does not represent the productivity of a recursive function, and still approaches it within ε? The reason is that Hornik et al. (1989) require a weighting function, an approximation metric, that is *finite*. Only then is it possible to approximate a recursive function by reproducing a finite number of its values. But one should reject this limitation if one wants to evaluate whether the network is able to reproduce the infinite productivity of a recursive function. What one should value highly is that this unlimited recursion is captured, not that some finite set of "important" characteristic values is correctly reproduced.

This is easily shown from a final example, presented in Table 2. The recursive grammar in that figure generates strings of increasing degrees of recursion. We can now define a metric μ, with value μ_1 for strings of degree 1, μ_2 for strings of degree 2, etc. If we want our metric to capture the infinite productivity if the recursive grammar, we should find it as important that strings of degree 1 are correctly generated as that strings of degree 2, degree 3, and so on are correctly generated by our simulating network. That is, $\mu_1 = \mu_2 = \mu_3 = \ldots$ But then, obviously, $\Sigma\mu_i = \infty$, and we don't have a finite metric anymore.

A remark on learnability

Connectionists make life harder for themselves than is necessary. Their standard approach is to demonstrate that networks can represent some domain of knowledge by showing that this domain of knowledge can be taught to the network. Though the logic is correct, it is also cumbersome. What we have learned from automata theory is that there is no simple relation between what an automaton can generate (or represent) and what the same automaton can learn. There are, in fact, surprising incongruencies between the generative power of automata and their learning capacity – even on the most lenient definitions of learnability (see Levelt, 1974, for a review of these matters). There is no good reason to believe that similar incongruencies won't arise for networks. Networks may have a far better representing than learning capability. But to find out, one has to develop a formal theory of learnability instead of

endulging in endless computer simulations (see Levelt, 1990, for further comments on this issue). Hornik et al.'s theorems on the generative power of networks could form the starting point for such a formal learnability theory.

Conclusion

The question addressed in this paper is whether Hornik et al.'s important theorems on the generative power of multiple-layer feedforward networks can lead to the conclusion that these networks are effectively Turing machines. The answer is two-way. There is a trivial sense in which the argument holds. Given a finite measure for weighing the difference between a characteristic function of a recursive set and any approximating function the network produces, one can construct a network that comes arbitrarily close to the recursive function. But this is done by having the network simulate a finite set of characteristic values, namely those that contribute most to the metric. One needs, at most, n hidden nodes to simulate n such characteristic values correctly.

But in a broader sense the answer is "no." The network fools us by being correct on a finite set of strings whose correct representation we are supposed to value highly. If, however, we want to capture by our approximation metric the infinite recursiveness of a recursive function, then a finite metric won't, do, and Hornik et al.'s results are irrelevant to the issue.

Acknowledgements. I am grateful to Jeff Elman, who supplied me with the necessary ammunition for this paper, in particular with the Hornik et al. results. I also wish to thank my mathematician brother Ton Levelt, who helped me understand the Hornik theorems and who functioned as a most creative sounding board throughout the writing of this paper. Still, he cannot be held accountable for any flaws in my argument.

References

Elman, J. L. (1989). Representation and structure in connectionist models. CRL Technical Report 8903.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward network are universal approximators. Discussion Paper 88 – 45R. Department of Economics, UCSD.

Levelt, W. J. M. (1974). *Formal grammars in linguistics and psycholinguistics* (3 vols.). The Hague: Mouton.

Levelt, W. J. M. (1990). On learnability, empirical foundations, and naturalness. Commentary on S. J. Hanson & J. Burr. What connectionist models learn: Learning and representation in connectionist networks. *Brain and Behavioral Sciences* (in press).

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. L. (1988). Encoding sequential structure in simple recurrent networks. Report CMU-CS-88-183.

Stinchcombe, M., & White, H. (1989). Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. Department of Economics, UCSD.