ways – one exhilarating but false, one prosaic and true. H&B appear to be exhilarated by the first reading, but adduce evidence that points only to the second.

The false but exhilarating reading is that any computable function can be computed by some network; in other words, networks have "Turing machine" power. Any finite network with units with a finite number of levels of activation (such as those used by McCulloch & Pitts) is equivalent to a finite state machine (Minsky 1967) however, and hence has considerably less than Turing machine power.

The result of McCulloch & Pitts to which H&B advert is that a network of "neural unit[s] computing a 'majority logic' or 'polymorphy' logic (m out of n) could be used, in principle, to implement any logical [i.e., Boolean] function" (Hanson & Burr, sect. 4.2 para. 1). Since any Boolean function of n inputs is a finite mapping $f:\{0, 1\} \rightarrow \{0, 1\}$, it can be performed by a finite state machine, for example by table look-up. Therefore, McCulloch & Pitts's result does not imply that networks have Turing machine power.

Similar considerations apply to the other general computational result that H&B adduce, that "there is no arbitrary computational limit on what [it] is possible to represent" (Hanson & Burr, section 4.7, paragraph 3), since "with at least one layer of a sufficient number of hidden units and a continuous, monotone fan-out function, any real valued function or mapping from R^n to R^m can be constructed" (Hanson & Burr, sect. 2.3, para. 12 [point 7]). Yet this cannot be quite right. After all, even a simple mapping $g:R^n \rightarrow R^m$ such that $g(r) = 1$ only if r is rational and $g(r) = 0$ otherwise is not network computable. (To see this, consider what the nature of the decision boundary would have to be, given that the rationals are dense and of measure 0.) Indeed, the class of mapping from R^n to R^m is not even Turing computable, since the reals are uncountable, there are uncountably many such mappings, but only countably many Turing machines.

The result H&B mention is surely more limited: that with a single layer of hidden units any appropriately well-behaved mapping from R^n to R^m can be approximated arbitrarily closely (g, as defined above, will not be well-behaved!) The analogy that H&B draw with Fourier analysis is apt. A Fourier series cannot construct any real valued function, but a sufficiently large number of terms of the Fourier series (hidden units) can approximate any appropriately well-behaved function arbitrarily well.

So a Turing machine reading of H&B's claim cannot be sustained. Yet a footnote attached to their claim suggests that H&B may intend this reading: "Since networks with sufficient resources can implement any real valued or Boolean function mapping . . . it is trivial to equate a network of a certain size and type with any Turing machine computing the same function" (Note 6). I doubt that they intend the uninteresting conclusion as stated: that any function that is network computable is Turing computable. Presumably, they intend the converse: that any Turing-computable function is network computable. This is the exhilarating but false reading of the original claim.

The true but prosaic reading of H&B's claim is that every computation performed by an actual digital computer can be performed by some network. A digital computer is, at bottom, simply a network of Boolean logic gates, each of which may be replaced by the appropriate network fragment. (This *is* guaranteed by McCulloch & Pitts). The resulting machine is a network which precisely mimics the digital computer.

The two conclusions that we have drawn may seem contradictory. If a digital computer is a finite state, how is it that the model of classical computation is the Turing machine? This is a subtle issue, but a key point is that although any actual computer is finite, it may be idealized into two parts: a finitely specifiable processor (a finite set of rules) and a potentially unbounded memory (storing arbitrarily complex representations; Pylyshyn 1984). Symbolic computation is best understood by idealizing away from memory and other resource limitations. Abstractions

Connectionism and classical computation

Nick Chater

Department of Psychology and Centre for Cognitive Science, University College London, Gower Street, London, WC1E 6BT, England

Hanson & Burr provide an informed discussion of connectionist learning, and, in particular, suggest a variety of interesting geometrical and statistical methods for analyzing the results of such learning. While generally sympathetic to H&B's paper, I shall focus here on one aspect that I find tendentious – their treatment of the relationship between connectionism and classical, symbolic computation.

H&B remark that "it is possible in principle for a net to implement any sort of classical computation (McCulloch & Pitts 1943)" (sect. 2.3, para. 4). This claim can be read in at least two

such as programming languages, theorem provers, and other pieces of software have Turing machine power, even though any concrete implementation must necessarily be resource limited. (There is a stack space-overflow, the set of formulae is too large to be stored, etc.).

These considerations may be taken as evidence for the view (Fodor & Pylyshyn 1988; Pinker & Prince 1988) that connectionist networks may be thought of as an alternative *hardware* in which to implement symbolic computation (since they are powerful enough to mimic the logic-gate hardware of actual digital machines), but that they cannot be seen as *alternatives* to structured, symbolic computation (idealized as having Turing-machine power). This conclusion need in no way undermine the significance of connectionism for cognitive science, however. The challenge for connectionism is to implement symbolic processes while retaining their computationally attractive network properties. In such implementations, symbolic processes will not be autonomous (Chater & Oaksford 1990) from the network substrate, but may exploit properties of that substrate as primitives (such as massively parallel constraint satisfaction). This challenge has long been recognized in the connectionist community, and has stimulated a wide variety of interesting schemes (Derthick 1987; Hinton 1981; 1987; Rumelhart 1986; Shastri 1985; Smolensky 1987; Touretzky & Hinton 1985). Meeting this challenge should be a concern of all advocates of symbolic models of cognition. After all, symbolic mental processes must be implemented, in real time, in a massively distributed system of simple, highly interconnected processing units – the brain.