Transcending Turing Computability

B.J. MACLENNAN
Department of Computer Science, University of Tennessee, Knoxville, TN, USA; E-mail: maclennan@cs.utk.edu

Abstract. It has been argued that neural networks and other forms of analog computation may transcend the limits of Turing-machine computation; proofs have been offered on both sides, subject to differing assumptions. In this article I argue that the important comparisons between the two models of computation are not so much mathematical as epistemological. The Turing-machine model makes assumptions about information representation and processing that are badly matched to the realities of natural computation (information representation and processing in or inspired by natural systems). This points to the need for new models of computation addressing issues orthogonal to those that have occupied the traditional theory of computation.

Key words: analog computation, analog computer, biocomputation, computability, continuous computation, hypercomputation, natural computation, Turing machine

1. Introduction

Hypercomputation (Copeland and Proudfoot, 1999) may be defined as computation that transcends the bounds of Turing-machine computability, that is, super-Turing computation. Why would we suppose that such a thing is possible, when computation is commonly equated to Turing-machine computation (a common misinterpretation of the Church-Turing thesis; see Copeland, 2000)? One line of argument comes from Penrose (1989), who argues that human cognitive abilities exceed those of digital computers; specifically, mathematicians can decide Gödel’s (formally) undecidable propositions. However, since human cognitive abilities reside in the neural networks of the brain, one might conclude (or at least speculate) that analog neural networks have super-Turing computational power. Although Penrose’s argument has been criticized widely (e.g., MacLennan, 1990a, and other peer commentary to Penrose, 1990), there is other evidence in favor of hypercomputation. For example, there is now considerable theoretical work showing that certain classes of analog computers have super-Turing power (reviewed in Copeland’s introduction to this volume). However the variety of results suggests that the theoretical power attributed to analog computers may depend somewhat delicately on the assumptions made in the theory. Interesting though these investigations are, this paper will take a different approach to transcending Turing computability. First, however, we must recall the assumptions underlying the theory of Turing-machine computability.
2. Assumptions Underlying Turing Computability

2.1. Historical Context

It is important to remember that the theory of Turing computability arose out of questions of effective calculability in the formalist program in mathematics. The theory addresses the sort of calculation that could be accomplished by mechanically manipulating formulas on a blackboard of unlimited size, but using only finite effort and resources. Specifically, the mathematicians that developed the theory were interested in what could be proved in formal axiomatic theories. As a consequence, the theory makes a number of assumptions, which are appropriate to its historical purpose, but which must be questioned when the theory is used for other purposes. Some of the developers of the theory of Turing computability were quite explicit about their assumptions (e.g., Markov, 1961, chs. 1–2; see also Goodman, 1968, ch. 5), but, as is often the case, later investigators have accepted them uncritically.

The roots of these assumptions go much deeper, however, and reach into the foundations of Western epistemology. For example, in the *Laches* (190C), Socrates says, “that which we know we must surely be able to tell.” That is, ‘true knowledge’ or ‘scientific knowledge’ (*epistêmê*) must be expressible in verbal formulas; non-verbalizable skill is relegated to ‘mere experience’ (*empeiria*; e.g., *Gorgias* 465A). These notions were developed further by, among many others, Aristotle, who investigated the principles of formal logic, and Euclid, who showed how knowledge could be expressed in a formal deductive structure.

Key to this epistemological view is the idea that knowledge can be represented in a *calculus* and processed by means of it, an idea which goes back at least as far as the Pythagorean use of figurate numbers (geometric arrangements of pebbles — *calculi* — or other tokens) to *calculate* and to demonstrate simple theorems in number theory. The idea recurs throughout Western philosophy, for example in Hobbes’ assertion that reasoning is calculation (*Leviathan* I.5), and in Leibniz’ attempts to design a knowledge representation language and to develop a mechanical calculator for automating reasoning (Parkinson, 1966).

Models are idealizations of what they model; that is what makes them models. What is included in a model depends on its intended purpose, what it is supposed to talk about. It is a cliche to say that one should not confuse the map with the territory, but it is an apt analogy. Different maps give different information about the territory. If you try to get information from a map that it is not designed to provide, it may give you no answer or an incorrect answer. So also with models. If we ask them questions that they are not intended to answer, then they may provide no answer or even an incorrect answer.

The Turing machine and equivalent models of computation are good models for their purpose: studying the capabilities and limits of effectively calculable processes in formal mathematics. They are also good models of digital computers with
very large (i.e. approximately unlimited) memories (for their design was inspired by the Turing machine). However, before we apply the model to issues arising in analog computation in natural and artificial intelligence, we must look critically at the assumptions built into the foundations of the model to determine if they are sufficiently accurate. That will be our next task.

Many of the assumptions of Turing computability theory can be exposed by considering the engineering problems of constructing a physical TM (Turing machine) or by looking at other practical engineering problems in signal detection, pattern recognition, control, etc. If we do so, we will discover that the assumptions are problematic; they are not obviously true. This phenomenological exercise will be my strategy in the remainder of this section. (For a more detailed discussion, see MacLennan, 1994b.)

2.2. INFORMATION REPRESENTATION

The traditional theory of computation assumes that information representation is formal, finite, and definite (MacLennan, 1994b). Formality means that information is abstract and syntactic rather than concrete and semantic. Abstract formality means that only the form of a representation is significant, not its concrete substance. Therefore there is no limit to the production of further representations of a given form, since the supply of substance is assumed to be unlimited. (This infinite producibility is the ultimate source of the potential, countable infinities of formal mathematics; see Markov, 1961, loc. cit.) Syntactic formality means that all information is explicit in the form of the representation and independent of its meaning. Therefore information processing is purely mechanical. Since ancient Greek philosophy, finiteness has been assumed as a precondition of intelligibility. Therefore, representations are assumed to be finite both in their size and in the number of their parts. Definiteness means that all determinations are simple and positive, and do not require subtle or complex judgements. Therefore there is no ambiguity in the structure of a representation.

Because representations are finite in their parts, they must have smallest elements, indivisible or atomic constituents. Individual physical instances of these atomic constituents are often called tokens, each of which belongs to one of a finite number of types. For example, ‘A’ and ‘A’ are two different tokens of the letter-A type.

Tokens are assumed to be indivisible and definite with respect to their presence or absence. However, in the context of practical signal processing it is not always obvious whether or not a signal is present. For example, if we see ‘x’ in a badly reproduced document, we may be unsure of whether we are seeing ‘x dot’, the time-derivative of x, or just x with a speck of dust above it. Similarly, we may observe the practical problems of detecting very weak signals or signals embedded in noise (e.g. from distant spacecraft).
Types are assumed to be definite and finite in number. That is, in classifying a token, there are only a finite number of classes among which to discriminate; there are no continuous gradations. Furthermore, the classification is definite: it can be accomplished simply, mechanically, and with absolute reliability; there can be no ambiguity or uncertainty.

That such an assumption is problematic can be seen by considering the construction of a physical Turing machine. It would have to have a camera or similar device to detect the token on the tape and a mechanism to determine its type (e.g., letter-A, letter-B, etc.). Certainly any such process would have some probability of error (even if superb engineering makes it very small), but the model ignores the possibility of error. Even in everyday life and in the absence of significant noise, it might not be obvious that ‘1’ and ‘l’ are of different types, as are ‘0’ and ‘O’.

We construct digital computers so that the assumptions about tokens and types are reasonably accurate, but in a broader context, pattern classification is a complex and difficult problem. In the real world, all classifications are fuzzy-edged and there is typically a continuum between the classes.

Next, we may consider compound representations comprising two or more tokens in some relation with each other. As examples, we may take the configuration of characters on a Turing-machine tape or the arrangement of symbols in a formula of symbolic logic. As with the tokens and types of the atomic constituents, we may distinguish the individual physical instances, which I’ll call texts, from their formal structures, which I’ll call schemata. The schema to which a text belongs depends only on the types of its constituents and their formal relations. Typically there is a countable infinity of schemata, but they are built up from a finite number of types and basic formal relations. For example, we have the countable infinity of Turing machine tape configurations (sequences of characters on a finite stretch of tape).

Texts are assumed to be finite and definite in their extent; that is, we can definitively determine whether they are present and where they begin and end (in space, time, or some other domain of extension). Typically, no a priori bound is placed on the size of a text; for example, the TM tape can increase without bound (although bounded tapes may be entertained for specific purposes). Practically, however, there are always bounds on the extent of a text; the ‘stuff’ that physically embodies texts (whether TM tape or bits in computer memory) is never unlimited. Indeed, the limits may be quite severe.

Schemata are assumed to be finite in ‘breadth’ (extent) and ‘depth’ (number of components). That is, as we analyze a schema into its parts, we will eventually reach a ‘bottom’ (the atomic constituents). This is a reasonable assumption for mathematical or logical formulas, but is problematic when applied to other forms of information representation. For example, an image, such as an auditory signal or a visual image, has no natural ‘bottom’ (level of atomic constituents). Don’t think of digital computer representations of these things (e.g., in terms of pixels or samples), but look out your window or listen to the sounds around you.
Phenomenologically, there are no atomic constituents. That is, continua are more accurate models of these phenomena than are discrete structures. (Therefore, in particular, not all computational systems are accurately modeled by Gandy’s (1980) *discrete deterministic mechanical devices*, which he shows to be equivalent in power to TMs.)

Similarly to the types of the atomic constituents, the basic formal relations from which schemata are constructed are assumed to be reliably and definitely determinable. Digital computers are designed so that this assumption is a good one, but in other contexts it is problematic. For example, if someone writes ‘$2^n$’, does it mean ‘twice $n$’ or ‘the $n$th power of 2’? Many basic relations, such as spatial relations, exist in a continuum, but the traditional theory of computation assumes that they can be perfectly discriminated into a finite number of classes.

2.3. INFORMATION PROCESSING

Like information representation, Turing computation assumes that information processing is formal, finite, and definite. Thus a computation is assumed to comprise a finite number of definite, atomic steps, each of which is a formal operation of finite effort and definite in its application. However, these assumptions are problematic even in a Turing machine, if we imagine it physically implemented. For example, there will always be some possibility of error, either in the detection and classification of the symbol on the tape, or in the internal mechanism that moves the tape, changes the internal state of the control, and so forth. Also, the assumption that the steps are discrete is an idealization, since the transition from state to state must be continuous, even if there is a ‘digital’ clock (itself an idealization of what can physically exist, as already observed by Turing, 1950, section 5). A flip-flop does not change state instantaneously.

Again, my goal is not to claim that these idealizations are always bad; certainly, they are sometimes accurate, as in the case of a modern electronic digital computer. Rather, my goal is to expose them as idealizations, so that we will not make them mindlessly when they are inappropriate. For example, information processing in nature is much more continuous. Certainly, when I write the word ‘the’ there is a sense in which the writing of the ‘t’ precedes the writing of the ‘h’, but the steps are not discrete, and the writing of each letter (as a process of motor control) interpenetrates with the writing of the preceding and following letters (see, e.g., Rumelhart et al., 1986, vol. 1, ch. 1). Therefore, we will have to consider information processing that cannot be divided into definite discrete atomic operations.

One of the important characteristics of computation, in the Turing sense, is that it can always be expressed in terms of a finite number of discrete, finite rules. This is accomplished by specifying, for each fundamental (schematic) relation that can occur, the fundamental relations that will hold at the next time step. By the assumptions of Turing information processing, there can be only a finite number
of such fundamental relations, so a finite number of rules suffices to describe the process. As a consequence, these computations can be expressed as programs on which universal machines (such as a universal Turing machine) can operate.

However, underlying the expression of information processing in terms of such rules lies the assumption that a finite number of context-free features suffices to describe the computational states (or schemata) on which the computation depends. Practically, however, many features are context-sensitive, that is, they depend on the whole text or image for their interpretation. For example, the interpretation of partially obscured letters or sounds depends on their surrounding context (of letters or sounds, but also of meaning; see for example Rumelhart et al., 1986, vol. 1, ch. 1). When we try to describe natural information processing (e.g., cognitive processes) with increasing accuracy, we require an exponentially increasing number of rules, an observation made by Dreyfus long ago (Dreyfus, 1979).

2.4. INTERPRETATION

Since the theory of Turing computability arose in the context of the formalist school of the philosophy of mathematics, the texts were often representations of propositions in mathematics. Therefore the domain of interpretation was assumed to be some well-defined (e.g., mathematical) domain, with definite objects, predicates, and propositions with determinate truth-values. While this is a reasonable assumption in the theory’s historical context, it is problematic in the context of natural cognitive processes, where propositional representations may have less definite interpretations. Indeed, as will be discussed later, in natural intelligence many representations are non-propositional and their pragmatic effect is more important than their semantic interpretation. In contrast, the traditional theory ignores the pragmatics of representations (e.g., whether a representation is more easily processed).

The conventional theory of interpretation assumes a determinate class of syntactically correct well-formed formulas. This class is important since only the well-formed formulas are assumed to have interpretations. Typically, the well-formed formulas are defined by some kind of formal generative grammar (essentially a non-deterministic program — a finite set of discrete, finite rules — for generating well-formed formulas).

In contrast, in practical situations well-formedness and interpretability are matters of degree. Linguists distinguish competence, the hypothetical grammatical knowledge of a language user, from performance, the user’s actual ability to interpret an utterance, and they focus their attention on competence, but from the practical perspective of natural cognition, performance is everything. In the natural context, the interpretation of an utterance may be a basis for action, and its ability to perform that pragmatic role is the foundation of interpretability.

The approach to interpretation pioneered by Tarski (1936) constructs the meaning of a well-formed formula from elementary units of meaning (objects, predic-
ates, functions), corresponding to the atomic units of the formula, by means of
definite constructors paralleling the constituent structure of the formula. However,
we have seen that in many important contexts the representations (e.g., visual input,
tactile input) have no natural atomic units, and the meanings of the basic features
are generally context-sensitive. To put it differently, Tarski’s recursive approach
assumes a discrete constituent structure with a definite ‘bottom’; this assumption
is a poor model of many important information representations.

2.5. Theory

Traditionally, the theory of computation looks at a calculus from the outside and
addresses such issues as its consistency and completeness. However, natural and
artificial intelligence often must process information that is non-propositional, and
pragmatic effectiveness is often more relevant than consistency or completeness.

Of course, the fundamental issue in the theory of Turing computability is whether a computation eventually terminates: something is computable only if it is
computable in a finite number of steps. (This applies to real numbers, in that the
successive digits or bits of the number, or successive rational approximations to
it, must each be computable in a finite number of steps.) This emphasis on the
finiteness of the computation can be traced to the theory’s historical context, which
was concerned with modeling finite proofs (any valid derivation from the axioms
is a proof, no matter how lengthy, provided it is finite). However, ‘eventual ter-
mination’ is of little value in many practical applications, for which information
processing must return useful results in strictly bounded real time. Furthermore,
useful information processing need not be terminating. For example, many robotic
applications use non-terminating control processes, which must deliver their results
in real time.

Traditionally, the theory of Turing computability has focused on the power of a
calculus, normally defined in terms of the class of mathematical functions it can
compute (when suitably interpreted). However, in many important applications
(e.g., control problems), the goal is not to compute a function at all, and it may
distort the goal to put it in these terms. That is, a function has an input, from which
it computes an output at some later time. In real-time control, however, output
is a continuous result of a process evolving (from an initial state) in continuous
response to continuously changing inputs. The latter is a simpler description of a
control system than is a TM, and a better model for investigating the robustness of
its computations in the presence of noise and other sources of error and uncertainty.

Since Turing computation makes use of discrete information representations
and processes, continuous quantities cannot be manipulated directly. For example,
a real number is considered computable if, in effect, it can be approximated dis-
cretely to any specified accuracy (i.e., its discrete approximations are recursively
enumerable). However, analog computational processes directly manipulate con-
tinuous quantities, and so the discrete computational model is very far from the
reality it is supposed to represent. Certainly, noise and other sources of error limit the precision of analog computation, but such issues are best addressed in a theory of continuous computation, which better matches the phenomena (e.g., Maass and Sontag, 1999). Of course, analog processes may compute approximations to a real number, but then the approximations themselves are real numbers, and often the process is one of continuous approximation rather than discrete steps. Progress in the right direction has also been made by Blum and her colleagues, who have extended (traditional, discrete) computational processes to operate on the reals (e.g., Blum et al., 1988), but the programs themselves are conventional (finite rules operating in discrete time).

The foregoing illustrates some of the questions that are assumed to be interesting and relevant in the theory of Turing computation, but we have seen that other issues may be more important in the analog computational processes found in natural and artificial intelligence.

2.6. Ubiquity of Assumptions

Before considering models of computation that transcend Turing computability, it will be worthwhile to note how difficult it is to escape the network of assumptions that underlie it. Historically, formal logical and mathematical reasoning were the motivation for the theory of Turing computation. These activities make use of discrete formulas expressing propositions with well-defined truth-values. This sort of ‘codifiable precision’ is the purpose for which formal logic and mathematics were developed. Therefore, we must use the language of logic and mathematics whenever we want to talk precisely about analog computation.

However, when we do so we find ourselves caught in the web of assumptions underlying Turing computation. For example, in point-set topology (i.e., topology which treats spaces, functions, etc. as sets of points) and in set theory we take for granted the self-identity of a point and its distinguishability from other points. Points are assumed to be well defined, definite. That is, two points are either equal or not — there is no ‘middle’ possibility — although of course they may be near or far from each other.

The dubiousness of this assumption is revealed, as before, by considering practical situations, for we can never determine with absolute accuracy whether or not two points are distinct. Practically, all points are fuzzy. (But how do we express this fuzziness mathematically? By associating a probability with each point in the space! That is, even with fuzzy points there is an underlying notion of identity.)

We can hardly avoid thinking of the real continuum but as made up of idealized points, which are like idealized tokens. Practically, however, ‘points’ may be far from this ideal. Discrete knowledge representation and inference is also taken for granted in our formal axiomatization of theories. As part of the historical mathematical program of reducing the continuous to the discrete, we use finite, discrete axioms to define uncountable sets, such as the real continuum. Yet the Löwenheim–Skolem Paradox suggests that any such axiom system must be inadequate for
completely characterizing a continuum. (The paradox, which dates to 1915, shows that any such axiom system must have a countable model, and therefore cannot uniquely define an uncountable continuum.)

Arguably, these assumptions underlie all rational discourse (e.g., even fuzzy logicians seek determinate theorems about fuzzy logic), but there are forms of knowing (i.e., forms of information representation and processing) that are not accurately approximated by rational discourse. Therefore, I am not arguing for the abandonment of logic and mathematics, but indicating the fact that their very structure biases our understanding of other kinds of knowing. These kinds of knowing are very important in natural and artificial intelligence, and should be understood from their own perspective, not through the distorting lens of discrete computation. Therefore we need a theory of continuous computation, which can contribute to an expanded epistemology, which addresses nonverbal, nondiscursive information representation and processing (MacLennan, 1988).

3. Natural Computation

3.1. Definition

Natural computation is computation occurring in nature or inspired by computation in nature; two familiar examples are neural networks and genetic algorithms (see, e.g., Ballard, 1997). Natural computation is quite similar to biocomputation, which may be defined as computation occurring in or inspired by living systems.

There are several reasons that it is important to understand the principles of natural computation. The first is purely scientific: we want to understand the mechanisms of natural intelligence in humans and other animals, the operation of the brain, information processing in the immune system, the principles of evolution, and so forth. Another reason is that many important applications of computer science depend on the principles of natural computation. For example an autonomous robot, such as a planetary explorer, needs to be able to move competently through a natural environment, accomplishing its goals, without supervision by a human being. Natural computation shifts the focus from the abstract deductive processes of the traditional theory of computation to the computational processes of embodied intelligence (see, e.g., Lakoff and Johnson, 1999). In the following subsection I will consider some of the key issues that a theory of natural computation should address.

3.2. Some Key Issues

One of the principal issues of natural computation is real-time response. If a bird detects some motion on the periphery of its field of vision, it must decide within a fraction of a second whether or not a predator is stalking it. Such hard real-time constraints are typical of natural computation, which must deliver usable results
either in bounded real time or continuously (as in motor control). Eventual termination is an important issue in the traditional theory of computation, because it is the basis for the definition of computability, but it is irrelevant to natural computation.

Furthermore, the traditional theory of computational complexity (e.g., NP-completeness) studies how the termination time of algorithms varies with the size of their inputs. For example, an algorithm will be considered linear if its running time is at most proportional to the size of the input. However, the theory intentionally ignores the constant of proportionality, since the complexity class is supposed to be independent of specific hardware implementation (i.e., it treats disembodied computation). Therefore, an algorithm that, for a size $N$ input, takes $N$ milliseconds is considered to be of the same complexity as an algorithm that takes $N$ hours (or $N$ centuries!); they are all “order $N$,” $\Theta(N)$. This is a useless map for finding one’s way in the wilderness of natural computation.

On the other hand, in natural computation the size of the input is usually determined by the structure of the sense organs or other ‘hardware’, so it is fixed. For example, there are about a million nerve fibers in our optic nerves, which our visual systems are able to process in the required fraction of a second. How our visual systems would handle twice, ten times, or a hundred times that number of inputs, is not a very interesting or relevant question. Therefore, in natural computation we are mostly concerned with nongeneral algorithms, that is, algorithms designed to handle inputs of a specific, fixed size. Or, in the terminology of linguistics, concrete performance is critical; abstract competence is unimportant.

Natural computation must exhibit tolerance to noise, error, faults, and damage, both internal to the system and external, in the environment. The real world is messy and dangerous, and natural computational systems need to be able to respond robustly.

The real world is also unpredictable, and natural computational systems must expect to encounter situations that they have not been explicitly designed to handle. Traditional AI systems, based on discrete, rule-based knowledge representation and processing, are often brittle in the face of novelty; that is, they behave stupidly. Because novelty is expected in natural environments, autonomous systems must respond to it in a flexible way, bending rather than breaking. Therefore most natural computation is continuously adaptive; since the environment is continually changing, so must an autonomous agent’s response to it. The adaptation may be gradual or rapid, but representations of computational processes (‘programs’) must accommodate it.

In natural computation we are generally interested in ‘good enough’ answers rather than optimal solutions, which are usually a luxury that cannot be afforded in a demanding real-time environment. Indeed, broad (robust) suboptimal solutions are often preferable to better, tightly defined optima, since the latter are more brittle in the presence of noise and other sources of uncertainty. In Herb Simon’s terminology, natural computation is satisficing rather than optimizing (Simon, 1969, pp. 64–65).
With this overview of some key issues in natural computation, we can look at the sort of idealizing assumptions that might underlie a theory addressing those issues. Some of them form the basis of a theory of continuous formal systems (or simulacra; see MacLennan, 1993a, 1994a, b, d, 1995).

4. Directions Towards a Theory of Natural Computation

4.1. Information Representation

We may begin by considering idealizations of information representation that are appropriate to natural computation.

4.1.1. All Quantities, Qualities, etc. Are Continuous

First, all quantities, qualities, etc. are assumed to be continuous (analog), as opposed to discrete (digital). Certainly this applies to sensory input: think of continuously varying intensities, frequencies, and so forth: they are all continuous (mathematical) functions. It also applies to motor output, which is necessarily continuous, even when it is abrupt. Information representations within the nervous system, between sensation and motion, are also continuous. Although the nerve impulses are ‘all or nothing’, the information is usually represented by the frequency and phase of the impulses, both of which are continuously variable. Further, in the ‘graded’ responses that take place in the dendrites, the continuous shape of the waveforms of the impulses is significant. Finally, the synaptic connections between neurons, where memory is believed to reside, have continuously variable ‘efficacies’, which are complex functions of the number, distribution, and placement of chemical receptors.

4.1.2. Information Is Represented in Continuous Images

Information in natural computation is generally extended continuously in either space or time (or both); that is, information is represented in continuous images. For examples, consider a sound (a pressure wave varying continuously over time), or a visual scene (a pattern of light and color varying continuously over space and time), or the tactile input over the surface of an animal’s body. Similarly, the motor output from an animal varies continuously in time over continuous muscle masses (see below on the issue of muscle fibers). Within the brain, information is often represented in cortical maps, across which neural activity varies continuously in space and time. Position in such maps may represent continuously variable features of sensory input or motor output, such as frequency, orientation, and intensity (MacLennan, 1997, 1999).

The fact that neurons, sensory receptors, muscle fibers, etc. are discrete does not contradict spatial continuity, since the number of elements is so large that the ideal of a continuum is a good model (MacLennan, 1987, 1994b, 1999). For example, since there are at least 15 million neurons per square centimeter of cortex,
even small cortical maps (several square millimeters) have enough neurons that a
continuum is a good approximation. Mathematically, information is most directly
and accurately described as a time-varying vector or field.

4.1.3. Images Are Treated as Wholes

If we think about the preceding examples of sensory input and motor output, we can
see that images are generally processed in parallel as wholes. Any segmentation
or ‘parsing’ of the image is secondary and a continuous (mathematical) function
of the image as a whole. For example, the separation of foreground information
from background information in visual or auditory input depends continuously
on the entire image. Furthermore, images cannot be assumed to have meaningful
atomic constituents in any useful sense (e.g., as individually processable ‘atoms’
of information). Mathematically, we may think of a continuum as comprising an
infinite number of infinitely dense infinitesimal points, but they bear their meaning
only in relation to the whole continuum.

Images cannot be assumed to be decomposable in any single unambiguous
way (as can discrete representations, typically), since there is no ‘preferred’ way
in which they were constructed (MacLennan, 1993a, 1994b). That is, we think
of discrete representations as being constructed from atomic constituents, but for
continuous representations the whole is primary, and any decompositions are sec-
ondary. (The same applies even for such decompositions as Fourier or wavelet
decompositions. In these, an image is represented as a weighted sum of fixed
images (e.g., pure sinusoids or wavelets), and therefore the weights or coefficients,
which depend on the image, can be considered its components. However, these
‘components’ are continuous quantities and are functions of the entire image or
extended regions of it.)

4.1.4. Noise and Uncertainty Are Always Present

In nature, nothing is perfect or exact. Even approximate perfection is rare. There-
fore, all images (both external and internal) should be assumed to contain noise,
distortion, and uncertainty, and processing should be robust in their presence. In-
deed, as in quantum mechanics, it is generally misleading to assume that there
is one ‘correct’ image; each image should be treated as a probability distribution
(a fuzzy or indeterminate image). (The mathematics of the Heisenberg uncertainty
principle is directly applicable to the nervous system; for a survey, see MacLennan,
1991; see also MacLennan, 1999.)
4.2. INFORMATION PROCESSING

4.2.1. Information Processing Is Continuous in Real Time

In natural computation, information processing is generally required to deliver usable results or to generate outputs continuously in real time. Because natural computation must deliver results in real time using comparatively slow components (neurons), the structure of the computations is typically shallow but wide; that is, there are relatively few (at most about a hundred) processing stages from input to output, but there is massively parallel processing at each stage. In contrast, Turing computation is typically deep but narrow, executing few operations (often only one) at a time, but executing very large numbers of operations before it produces a result.

Furthermore, processes in nature are continuous, rather than proceeding in discrete steps. Certainly, the nervous system can respond very quickly (as when the bird decides to flee the predator) and (approximately) discontinuously, and neurons can exhibit similar abrupt changes in their activity levels, but these changes can be approximated arbitrarily closely by continuous changes. As in the theory of Turing computation we use discrete processes to approximate continuous change, so in the theory of natural analog computation we may use continuous approximations of discrete steps. Thus there is a kind of complementarity between continuous and discrete models (MacLennan, 1993b, d, 1994d), but continuous processes more accurately model natural computation.

4.2.2. Information Processing Is Usually Nonterminating

In natural computation, real-time control processes are more common than the computation of function values. Therefore, most computations are nonterminating, although they may pass through temporary equilibria. Rather than ‘eventually’ computing a result, natural computation must produce a continuous, unending signal in real time.²

4.2.3. Noise, Error, Uncertainty, and Nondeterminacy Must Be Assumed

Since noise, error, damage, and other sources of uncertainty must be presumed in both the external environment and the internal operation of a natural computation system, information processing is typically nondeterministic; that is, we have a continuous probability distribution of computational states. Therefore, the correctness of an answer is a matter of degree, as is the agent’s confidence in it, and hence its proclivity to act on it.

4.2.4. There Is a Continuous Dependence on States, Inputs, etc.

Since processes should be insensitive to noise and other sources of error and uncertainty, they should be continuous in all respects (i.e., continuous functions of input, internal state, etc.).
4.2.5. Processes Need Not Be Describable by Rules

We must consider information processes that are orderly, yet have no finite description (even approximate) in discrete formulas, such as mathematical equations. It may be surprising that such processes even exist, but a simple cardinality argument shows that it must be so (MacLennan, 2001): the set of programs that could be used to compute or approximate a real number, is countable, but the set of real numbers in uncountable. Therefore, most real numbers are not Turing-computable. Thus, even if a continuous process is governed by differential equations, it may not, in general, be expressible in finite formulas, since the coefficients might be real numbers that are not computable or approximatable by a Turing machine. Although we cannot write down such equations (since we cannot write down the coefficients, or even write down programs to approximate them), nevertheless they are real quantities that may govern information processes.

On the other hand, such processes may be finitely expressible by the use of continuous representations, which I have called guiding images (MacLennan, 1995). For example, a continuous potential surface, such as a potential energy surface, can constrain a continuous process. Thus, a deterministic process might be defined to take the path of most rapid decrease of energy; a nondeterministic process might be allowed to progress along any energy-decreasing trajectory. Whenever a continuous process is constrained by fixed continuous quantities or functions, these can be considered guiding images of the process.

4.2.6. Processes May Be Gradually Adaptive

As previously discussed, natural computation must deal with novelty in its environment. Therefore, typically, information processing must adapt — slowly or quickly — to improve the system’s performance. This is possible because the guiding images that organize the process can change continuously in time. Since rule-like behavior is an emergent phenomenon, gradual adaptation can lead to reorganization of an entire system of apparent rules (MacLennan, 1995).

4.2.7. Processes Are Matched to Specific Computational Resources and Requirements

We are primarily concerned with processes that can handle prespecified input and output channels and run on prespecified hardware, and that can meet the required real-time constraints. Asymptotic complexity is largely irrelevant. Or, to put it in linguistic terms, performance (versus competence) is everything.
4.3. **INTERPRETATION**

4.3.1. *Images Need Not Represent Propositions; Processes Need Not Represent Inference*

In natural computation, images need not represent propositions, and processes need not represent inference. However, images may have a nonpropositional interpretations and information processing may correspond systematically with processes in the domain of interpretation. (This is, indeed, the original meaning of *analog* computation; see also MacLennan, 1993c, 1994d.)

4.3.2. *Interpretability and Interpretations Are Continuous*

As previously remarked (Section 2.4), in a natural context the interpretability of an image depends on its suitability as a basis for action or further processing. However, we have seen (Subsection 4.2.4) that natural information processing is modeled best as a continuous mathematical function of a continuous representational space. Therefore, within the context of natural computation, interpretation of an image space is a continuous function of those images. (Of course we, as outside observers, can interpret images discontinuously, or in any way we like. But the interpretations that are most relevant to natural computation will be continuous.)

Mathematically, a continuous map (function) of a continuum must be a continuum. Therefore, in natural computation, a domain of interpretation must be a continuum. (See MacLennan, 1994b, for a discussion of the topology of images spaces and maps on them.) As a consequence, when an image is interpretable, the interpretation must be a continuous function of the image, so there can be no discrete changes of meaning. Furthermore, if some images are interpretable and others are uninterpretable, there must be continuous variation between these extremes, and thus degrees of interpretability. In other words, well-formedness (as a precondition of interpretability) must be a matter of degree. This is one basis for the robust response of natural computation to noise, error, and uncertainty. However, it also means we need a different, continuous way of describing the well-formedness of images. For example, one can define continuous-time nondeterministic processes for generating images that are analogous to grammars for discrete languages (MacLennan, 1995).

4.3.3. *Pragmatics Is Primary; There Need Not Be an Interpretation*

Finally, we must note that natural computations need not be interpretable. Pragmatics is primary; the computation is fulfilling some purpose for the agent. Semantics (interpretation) and syntax (well-formedness) are secondary. The trajectory of natural information processing may pass through phases in which it is more or less interpretable, while still accomplishing its pragmatic end. To give a simple example, the Nekker Cube is a simple line-drawing of a cube; it is an ambiguous figure, which our visual systems can interpret as a three-dimensional object in two different ways. However, under continued viewing, our perception switches
spontaneously between these interpretations, and between them must pass through states that are uninterpretable as 3D objects. This is a simple, perceptual example of the common cognitive operation of reinterpretation.

4.4. THEORY

4.4.1. Unimportant Issues

First, it will be worthwhile to remind the reader of the issues traditionally addressed by the theory of Turing computation that are unimportant, or less important, in the theory of natural computation.

As previously discussed, termination (upon which definitions of computability are based) is not an interesting question since: (1) many useful information processes do not terminate, and (2) ‘eventual termination’ is irrelevant, since information processing must satisfy continuous, real-time constraints. Even when we choose to address traditional decision problems, we must do it in the context of continuous information representation and processing (e.g., MacLennan, 1994b, c).

For the same reasons, asymptotic complexity and complexity classes (such as ‘NP-complete’) are uninteresting. First of all, “the constants matter” when we are operating in real time; the difference between milliseconds and minutes is critical! Second, we are not concerned with how the performance of the algorithm scales with larger inputs, since it will not have to process inputs larger than those actually provided by the hardware. It doesn’t matter whether an algorithm is $O(N)$, $O(N^2)$, $O(2^N)$ (that is, with running time linear, quadratic, or exponential in the size of the input), so long as the algorithm meets the real-time constraints for the particular $N$ that it must process.

Universal computation — the ability to have a programmable universal Turing machine — is important both in the traditional theory of computation and in practice, for it is the basis for programmable digital computers. Whether there could be a corresponding notion of a universal analog computer is certainly an interesting question, which has been addressed in several contexts (e.g., Shannon, 1941; MacLennan, 1987, 1990b, 1999; Pour-El, 1974; Rubel, 1981, 1993). However, it is not central to natural computation, for natural computation systems are typically constructed from the interconnection of large numbers of special-purpose modules.

Even ‘abstract thought’ is special-purpose compared to other information processing done by brain modules. That is, it makes use of specialized linguistic faculties in the brain, which are inadequate for many of the other needs of natural computation, such as sensorimotor coordination. The traditional discrete model is inadequate even as a model of abstract thought, which in us is flexible, holistic, and context-sensitive, and builds upon nonpropositional communicative and information processing skills. (See for example MacLennan, 1988, 1994b, for further discussion of the inadequacy of discrete calculi as models of human reason.) Many
of our abstract concepts build upon nonpropositional knowledge inherent in our embodiment (cf. Lakoff and Johnson, 1999).

4.4.2. Important Issues

Finally, we can enumerate a few of the issues that a theory of natural computation should address.

One important issue is a natural computation system’s generalization ability and flexibility in response to novelty. Natural computation systems should not behave stupidly, as many rule-based systems do, when confronted with the unexpected. Therefore, such systems must be able to discover pragmatically useful structure that can be a basis for reliable extrapolation.

As already stated many times, the theory must address the behavior of the system in response to noise, error, and other sources of uncertainty, and these effects must be assumed from the beginning, not added onto a fictitious ‘perfect’ system.

We need to know how to optimize performance subject to fixed real-time and resource constraints. Given the hardware, how do we get the best results for the widest variety of inputs most quickly? The generality of natural computation algorithms derives from the procedures for fitting the process to the hardware and real-time constraints.

Another important problem is adapting processes to improve their performance. That is, the theory must address learning algorithms and means for avoiding the pitfalls of learning (rote learning, destructive learning, instability, etc.). Related is the issue of designing processes that adapt when their hardware is degraded (by damage, age, etc.).

Finally, we observe that the ‘power’ of natural computing is not defined in terms of the class of functions it can compute, nor in terms of numerical ‘capacity’ (number of memories, associations, etc. that can be stored). Rather, power is defined in terms of such factors as real-time response, flexibility, adaptability, and robustness. Some of these factors may be difficult to quantify or define formally (e.g., flexibility), but that is why we need the theory.

5. Conclusions

In conclusion, it is worthwhile to return to the theme of transcending Turing computability. The traditional theory of computation, by which I mean the theory based on the Turing machine and its equivalents, is oriented toward posing and answering a certain class of questions. Among these questions is the ability of various kinds of abstract discrete computing machines to compute various classes of functions, in the sense that when provided with a (discrete) input they will, in finite time, produce the correct (discrete) output. Continuous inputs, outputs, and processes must be modeled indirectly, via discrete approximations. Within this theoretical framework we may formulate one notion of ‘transcending Turing computability’: that is, being able to compute a larger class of functions than those computable by
TMs. Although this notion is certainly interesting, it is not, I think, in the long run, the most important sense in which Turing computability may be transcended.

In this article I have argued that natural computation, which is of crucial scientific and engineering importance, poses different questions from the traditional theory of computation. In natural computation, for example, information representation and processing are more accurately modeled by continuous mathematics, and concrete performance is more important than abstract computability and asymptotic complexity. The TM model makes idealizing assumptions that are appropriate to the questions it was designed to address, but it is a poor match to the central issues of natural computation. For example, to ask whether a TM can have the capabilities of a natural computation system, we should first ask whether a TM can respond continuously in real time to continuously varying input, and behave robustly in the face of noise, error, and uncertainty. But even to formulate the question is to see how inadequate TM theory is to answer it.

When an idealized model deviates too much from the phenomena it is intended to model, we run the risk that any answer it provides will be more about the model than about the modeled phenomena. We will be less likely to be misled if we turn our attention to new models that make idealizing assumptions more compatible with natural computation and the questions it poses. I believe that we are currently in a situation comparable to the study of effective calculability one hundred years ago (before the pioneering work of Turing, Church, Post, Markov, and all the others): we have an informal understanding of the phenomena of interest and some mathematical tools for addressing them, but we lack a comprehensive theory. This should be our goal, if we truly want to transcend Turing computability.

In brief, I claim:

Turing machine theory is not wrong but irrelevant.

This is, of course, an overstatement. Turing machine theory is relevant to questions of effective calculability in logic and mathematics, and to the classes of functions computable by digital computers. However, the assumptions of TM theory are not a good match to natural analog computation. Therefore, although it is important to engage the traditional issues (such as computability), it is also imperative to transcend them. New paradigms bring new questions as well as new answers. Turing computability asked one kind of question, but natural computation is asking a different kind.

Notes

1 Indeterminate identity statements are a possible solution to this problem. Copeland (1997) presents a possible formulation, which also illustrates the difficulty of escaping from underlying assumptions of determinate identity.

2 Certainly Turing machines are capable of generating unending sequences of outputs, but in the typical formulation (e.g., Turing, 1936): (1) the outputs depend only on the initial input (tape configuration), not on an unending sequence of inputs; (2) the sequence is discrete, not continuous; (3)
there is no fixed bound (strict upper bound) on the time to compute each sequence element, as there is in a control system.

References


