BEYOND THE TURING LIMIT?

This paper deals with the *Church-Turing Thesis*, its proper understanding, and the question whether there is computability beyond the *Turing Limit*. These notions are introduced and then related to some forms of non-standard computation, especially oracle machines and analog computation. I will present in part the work of Hava Siegelmann[1]. Her research deals with the computing power of (neural) networks, resp. of analog computation generally. This research nowadays is referred to in the cognitive sciences and connectionism.

Is there computation *beyond the Turing Limit*? What about the *Church-Turing Thesis*? There are rumours that analog computation is beyond the capabilities of Turing machines and this is considered to be a refutation of the Church-Turing Thesis (CT). All here depends on what you mean by "computation" and "machine". Section I repeats some standard definitions relating to computability. Section II discusses the content of (CT). Section III outlines analog computation, which is related then to the proper understanding of (CT). Section IV compares *Siegelmann`s Thesis* to (CT).

I COMPUTATION – SOME DEFINITIONS AND STANDARDS

A computable function f is a rule between objects which can be explicitly presented by *finite* means (usually [coded by] natural numbers) that specifies how to get the second element from the first. If the objects are built from a finite stock and range(f) is binary, f is a *characteristic function* and domain(f) is a *language*. A non-computable function is an infinite set or ordered pairs for which no 'reasonable' rule can be provided. 'reasonable' obviously depends on an intuitive notion of computation or algorithm. A function can also be uncomputable because there are not any resources (reasonable or not) to express a rule of pairing. Automata are *theoretical machines*. Automata that cannot be built by our finite resources are *notional* machines only.

One type of automata is *stronger* than another if the first can compute a set of functions the second cannot compute. The concept *machine* should be kept apart from the concepts *algorithm* and *finite*. Abstractly an automaton is quintuple M=<Q,I,O,f, h> consisting of the sets of states Q, the input space I, output space O, and the functions f:Q×I→Q (the next-state map) and h:Q →O (the output map). Let Σ be a set of possible input letters, then I= $\Sigma\cup\{\$\}$, "\$"designating the end of a string. Σ^* is the set of words. Any input $x\in\Sigma^*$. $\varepsilon\in\Sigma$. $\Sigma^+= \Sigma\backslash\{\varepsilon\}$. $\Sigma^{\leq n} =\cup_{k\leq n} \Sigma^k$. x(t) designates the state of all variables at t, i(t) summarises the content of the inputs at t, so that x(t+1) = f(x(t),i(t)), y(t) is the total output at t, y(t)=h(x(t)).

[1] *Neural Networks and Analog Computation*. (Boston et. al. : Birkhäuser, 1999). Section III reports some of her results and follows her outline of the field of analog computation and types of networks.

Automata depend on time resources. Each x of the domain of computation we associate a measure of length |x|. We then define a partial function $T:N\rightarrow N$. T(|x|) is defined only when the computation of all domain elements of this length halts. A machine M *computes in time T*, if for all inputs x for which T(|x|) is defined, M halts after performing not more than T(|x|) steps of computation. So we get *time complexity classes*.

The standard automaton of computation is the (deterministic) Turing machine (TM). The input is binary. Q is finite. At each step the symbol "a" under the head is read, the state is checked and one of the elementary operations (moving [$m\in\{L,R\}$], writing [$b\in\{0,1,\varepsilon\}$, changing control state [$q'\in Q$]) is executed. Transitions are described by a *function* *g*(a,q)=(b,m,q'). If the control reaches the halting state, the machine stops, the binary sequence up to the first "ε" right of the head being the output. TMs can be arithmetically encoded. TMs are countable (say by a standard numbering τ). τ(i) is the i^th TM, $\varphi_{\tau(i)}$ the function computed by it. The existence of a universal TM ***M*** which computes the function $\varphi(i,x)= \varphi_{\tau(i)}(x)$ is an existence theorem for a *general purpose digital computer*.

As a model of the comparisons to come let us consider non-deterministic Turing machines. A non-deterministic TM (NTM) differs from a deterministic TM (DTM) in that in at least one step of the computation there is a *choice* regarding what to write, how to move and to proceed to which state. NTMs have a transition *relation*. An input is accepted by a NTM if it is accepted by *some* transition.

1. *Theorem*:

There is a DTM which computes *f* $\Leftrightarrow$ there is a NTM which computes *f*.

Non-deterministic TM *are not stronger* than DTM. ("$\Leftarrow$" is the interesting result.) But non-deterministic TMs are more efficient than deterministic TMs. (By using choices they can 'guess' results.) An *efficient computation* is defined as one that requires polynomial time on a DTM. This complexity class is **P** (halting on x in time $c|x|^k$ for some constants c, k). **EXP** is the class of functions taking exponential computation time on any DTM. Although *solving* some problem might be in **EXP**, *recognising a solution* can be in **P**. **NP** (*Non-deterministic Polynomial time*) is the class of functions computed by a NTM in polynomial time. Some functions *supposedly* in **EXP** are in **NP**. (It´s open whether **P** = **NP**.) Comparing TMs to other types of computation one can consider the *speed up* even if no new functions are computed.

Most functions cannot be computed by TMs (either DTMs or NTMs). There are only countably many TMs but uncountably many functions $f:N\rightarrow N$ from natural numbers to natural numbers. Irrational numbers cannot be finitely represented, so cannot be a finite input. Some problems can be proven to be unsolvable (e.g. the Halting Problem [HP]).

II THE CHURCH-TURING THESIS

The *Church-Turing Thesis* (CT) can be expressed in different ways, for example:

(CT') Everything that is computable is recursive(ly computable).[2]

(CT'') Everything that is computable is computable by a Turing machine.[3]

The first expression refers to the recursive *functions*, the second to Turing *machines*. The *Church-Turing Thesis* is generally seen as outlining an upper limit on computability. Nothing seems to be computable that is not Turing computable. Seen from a naive point of view –the naiveté of which will be explained shortly– the existence of super-computability would falsify the *Church-Turing Thesis*. But (CT) involves a precise and definite notion of an effective procedure (an *algorithm*) and a corresponding concept of computability:

• (CT) identifies the *intuitive* notion of computability with a *formally explicated* notion (being computable by a TM).

• The cornerstone of this is the idea of an *algorithm M* on *discrete* symbols ω:

(i) executing steps each of which is mindless,

(ii) where each computation of *M*(ω) ends after finitely many steps (if *M*(ω) is defined at all),

(iii) *M* is implementable by different devices.

To avoid misunderstandings CT therefore can be expressed properly as:

(CT) Everything that is intuitively computable is computable by a DTM.

CT is not talking about any computability that could be nor about any type of machine that could be – keep that in mind!

There are at least three aspects of (CT):

(i) *reconstruction*: DTM is said to characterise the *intuitive* notion of computability,

(ii) *conceptual*: by proving equivalences between formal models of algorithms we grasp an *absolute* concept of computation,

(iii) *epistemological*: (CT) outlines (some) structures and limits of the *mind* (being a bridge to meta-logical results).

We are concerned here mostly with the first aspect, sometimes with the third. Both are related to the *cognitive sciences*: Given (CT), the arithmetization of formal languages and

Montague`s Thesis[4]

(MT) Natural language are equivalent to some formal languages.

we get:

(CTM) Anything that could be computed or said can be computed or generated

---

[2] Cf. Alonzo Church, "An Unsolvable Problem of Elementary Number Theory" (*American Journal of Mathematics*, 1936:345-63), p.356.

[3] Cf. Alan Turing, "On Computable Numbers, with an Application to the *Entscheidungsproblem*" (*Proceedings of the London Mathematical Society*, 1936:230-65), p.230.

[4] Cf. *Formal Philosophy* (New Haven: Yale University Press, 1976, 2nd Edition), p.188.

by a deterministic Turing machine.

Now consider the *representational theory of the mind*[5]:

(RTM) The mind can be characterised by symbol manipulation.

All in all we get:

(AI) The mind can be simulated by a (deterministic) TM.

(AI) expresses the basis of classic *artificial intelligence*. So controversies about (CT) and super-computability can be seen as having immediate effects in the philosophy of mind. Since some claim the brain – which need not be identical to the mind (!) – to be an analog net we have to consider the computing power of such nets.

(CT) is phrased in terms of the standard notions of computability (i.e., recursive functions or Turing machines). These define steps of computation which are discrete and effective (e.g. writing a symbol, moving one to the left). But could there not be a different concept of computation altogether? Asking for the *Turing Limit* means asking whether there are other concepts of computability and whether a machine corresponding to such a concept of computability might compute functions which cannot be computed by a deterministic TM. If there is such a machine, it is *beyond* the Turing limit! We can ask at least three questions now:

(1) What other types of computability are there, if there are any.

(2) Does the existence of this type of computability put (CT) in doubt or even falsify it?

(3) Can there *really be* such machines?

The last question asks whether some type of theoretical machine could ever be built. David Deutsch[6] introduced a physical interpretation of the (CT):

(DCT) No *realisable physical device* can compute functions that are not TM-computable.

To prepare the assessment of the relation between analog computability of some sort of nets and (CT) let us first consider the question of the importance of oracle machines to (CT). An oracle machine (OTM) is a TM supplemented by a further move: looking up for any input x whether for a given set A: $x \in A$ or $x \notin A$. The set A being any set, an OTM can decide *any* set! (How it is done does not matter.) Given an OTM we can introduce the functions that are computable *relative to* an OTM including an oracle on a set A. A function f:N→N is A-computable iff there is some oracle computable function $g:2^N \times N \to N$ such that $g(A,x)=f(x)$, $x \in N$. OTMs can be used to introduce a preorder of *Turing reducibility* (with equivalence classes of *Turing degrees*).

[5] Cf. Zenon Pylyshyn, *Computation and Cognition* (Cambridge/MA: MIT, 1995, 5th Edition), pp.23-48.

[6] „Quantum theory, the Church-Turing principle and the universal quantum computer“ (*Proceedings of the Royal Society of London*, 1985:96-117).

2. *Theorem*:

If the set A of an OTM *M* is recursive φ_M can be computed by a DTM *M'*. If the set A of an OTM *M* is non-recursive, there is no DTM *M'* such that $\varphi_M = \varphi_{M'}$.

So OTM is really *stronger* than DTM. OTMs solve HP, and decide First Order Logic. (The Turing degree of a non-recursive, r.e. set A contains sets which are not even r.e.)

Do Oracles Matter? Oracles are stronger than DTMs. They are *beyond* the Turing Limit. But they are mysterious – by definition.

(ad Q1) OTM defines a new type of computation. (In fact a hierarchy of degrees.)

(ad Q2) Are oracles related in any way to the claim of (CT)?

At least one author has claimed that: Jack Copeland[7] accusing a lot of cognitive scientists of a "wide spread" *Church-Turing fallacy* (a supposed misuse of (CT)). Polemics and Copeland`s sophistic reading of Church and Turing aside the first of the central questions is:

(a) What is meant by "machine" (in CT)?

If (CT) was the claim that any machine could be simulated by a DTM it would be false. (Since oracles cannot.) But that is definitely not the claim made! (CT) deals only with machines that are algorithmic (and therefore intuitively computable). Oracles *are not* – by definition.

(b) What does *mechanism* claim?

If mechanism only claimed the mind to be *some* machine, the mind could be an oracle! To give mechanism some bite it has to claim that the mind is computable. It has to claim something like (AI).

What can we learn from Copeland`s attack? – We should keep in mind:

(i) (CT) is falsified only if some of the functions which are computed by a new type of computation are *intuitively* computable.

That something can be *expressed finitely* does not make it *effective*! (E.g. "Divide this by the largest prime.") Any machine requiring non-finite input does not compute effectively either.

(ii) If analog machines are not machines in the sense of (CT), we could try to express an anlog to (CT) for this kind of device.

(iii) If analog machines are not machines in the sense of (CT) we have to reconsider mechanism only if we consider what they compute to be part of the mind (in distinction to the brain).

Even if the brain was of a non-standard computability type and further on that type was beyond the Turing Limit, this would *not* falsify (CT) or narrow mechanism, since the mind need not be the brain. Maybe the brain is an analog device that *implements on some level* a TM (as Daniel Dennett claims[8]).

[7] Jack Copeland, "Narrow versus Wide Mechanism" (*Journal of Philosophy*, 2000:5-32), Jack Copeland and Richard Sylvan, "Beyond the Universal Turing Machine" (*Australasian Journal of Philosophy*: 46-67).

If the analog processes are not *cognitively penetrable* they need not belong to the mind.[9]

Now – are there any oracles? Otherwise OTMs would be notional machines only.

If there are irrational quantities in nature there could be a quantity 0,101010001... coding the halting function! If some mechanism M could measure this quantity to *any* specified number, M is an oracle. But even that does not mean that M could be *built* by us – (DCT) need not be violated. We cannot built a device such that we assign some irrational value to some parameter/gadget. That requires infinite precision, which is practically impossible.

There are other machine models that are beyond the power of the (universal) Turing machine, e.g. *Coupled Turing Machines* or Copeland`s *Accumulator Machines*, but they do not have finite input, so are beyond what is intuitively computable. These machines are notional machines. Networks on the other hand are real – but what can they do?

III (NEURAL) NETWORKS AND THEIR COMPUTING POWER

Neural Networks are interconnections of simple processors or *neurons*. The directed connections are characterised by a positive or negative real (a *weight*). Each neuron computes a scalar function of its weighted inputs (the *response*). The output of each neuron is sent to other neurons in the network. Feed-forward networks are arranged in multiple layers, such that the output of one layer is the input of another layer. Their computation ends in a fixed number of steps. Recurrent networks on the other hand allow loops and thus memory of past computations can be present in the network. Let us introduce some definitions of network computations:

Output neurons take binary values only. Input arrives on two binary input lines: the data line D and the validation line V (indicating whether the data line is active). So: $u(t) = (D(t),V(t)) \in \{0,1\}^2$ for each t.

There are two output processors: G for data and H for validation. Encode the word $\omega = \omega_1 \ldots \omega_k \in \{0,1\}^+$

by $u_\omega(t) = (D_\omega(t),V_\omega(t)),\ t \in N$

$D_\omega(t) = \omega_k$ if t = 1…k, 0 otherwise

$V_\omega(t) = 1$ if t = 1…k, 0 otherwise

ω is binary coded previously. A word $\omega \in \{0,1\}^+$ is *classified in time r* by a formal net starting from the initial state x(1)=0 if the input lines (D_ω,V_ω) take the values $D_\omega = \omega 0^\infty$ and $V_\omega = 1^{|\omega|}0^\infty$, and the output line $H_\omega(t)=0$ for t<r and $H_\omega(r)=1$. If $G_\omega(r)=1$ then ω is *accepted*, if $G_\omega(r)=0$ ω is *rejected*. A language $L \subseteq \{0,1\}^+$ is *accepted* by a formal network *N* if every $\omega \in L$ is accepted by *N* and every $\omega \notin L$ is rejected or not classified by *N*. L is *decided* by *N* if L is accepted by *N* and its complement is rejected by *N*. L is *decided in time T* by *N* if $\omega \in \{0,1\}^+$ is correctly classified in time t< T(|ω|). Let $\psi: \{0,1\}^+ \rightarrow \{0,1\}^+$ be a partial

8 *Consciousness Explained* (London: Bradford Books, 1991), pp.209-26.

9 Cf. Pylyshyn, *Computation and Cognition*, pp.130-46.

function and *N* be a formal net with input (D,V) and output (G,H). ψ is *computable by N* if for every $\omega \in \{0,1\}^+$:

1. If $\psi(\omega)$ is undefined, then $H_\omega = 0^\infty$

2. If $\psi(\omega)$ is defined, then $(\exists r \in N)$ (*response time*) such that:

$G(t) = \psi(\omega)[t-r+1]$ if $t = r\ldots(r+|\psi(\omega)|-1)$, 0 otherwise

$H(t)= 1$ if $t = r\ldots(r+|\psi(\omega)|-1)$, 0 otherwise

The response time is the time the net needs to start delivering output (bit by bit). The function $\psi(\omega)$ is *computable in time T* if for every $n \in N$ $T(n)$ is defined iff $\psi(\omega)$ is defined on all inputs of length n, and there is a net *N* such that for every $\omega \in \{0,1\}^+$: if $\psi(\omega)$ is defined, the response time is at most $T(|\omega|)$.

If the response function is binary a *finite network* cannot accomplish more complex computations than a *finite automation* (i.e. it can compute only *regular* languages). So either we need *infinite* networks or the activation functions must allow for *non-binary* – maybe even continuous – *values*. Recurrent Nets (RN) have n neurons, the ith has an activation value $x_i(t)$ at t, there are m external binary inputs u_i and the connections carry weights a_{ij}, b_{ij}, c_{ij}. The dynamics is a map (*net function*): F:$R^n \times \{0,1\}^m \rightarrow R^n$ componentwise, with *response* σ: $x_i(t+1)= (\sum^n(a_{ij}x_j(t)) + \sum^m(b_{ij}u_j(t)) + c_i)$.

These networks do not work like NTMs or probabilistic TMs. Once weights, the output processors and σ have been specified, the behaviour is completely defined for a input coding. A net is *non-determined* if it additionally contains a *guess line* (guessing a symbol like a NTM), accepting x if *some* computation accepts x. The weights of the nets may assume different kinds of values. If the weights can assume arbitrary *real* values we have *Real-weighted Recurrent Networks* (RRN).

RRN correspond to *Advice Turing Machines* (ATMs). So before we continue talking about different kinds of recurrent networks we take a look at the computing power of ATMs.

An ATM is an extended TM. It is a TM supplemented by a further input tape with *advice* on the input which is computed at the moment. All inputs ω of some length $|\omega|$ have the same advice sequent. Let $\omega \in A \subseteq \Sigma^*$ and ν:$N \rightarrow \Sigma^*$ be the advice function. Then $\nu(|\omega|)$ is the advice. The length of the advice is bounded as a function of the input and measures the level of non-computability. Given a class of languages C, bounding functions h_i in H, that $L_i \in C/H$ means that language L_i can be computed in complexity class C given advice computed by $h_i \in H$. **P/poly** is accepted by ATMs with polynomial advice computing in polynomial time.

3. *Theorem*:

If *exponential* advice is allowed to an ATM *any* language L is computable.

Proof: L is decided by fixing the advice for an input x_i of length n to have “1” in location i (given locations $1,2,\ldots 2^n$.) iff $x_i \in L$.

Obviously ATMs are beyond the Turing Limit. Let α be the advice of an ATM.If α cannot be generated from a finite rule (an algorithm) call α „nonuniform“.

4. *Theorem*:

If the advice α of an ATM *M* is nonuniform *M* accepts sets that are not accepted by any DTM (i.e. are not computable).

5. *Theorem*:

RRN = ATM

Proof (Outline): Let the concatenation of the advice be r= $\nu(1)$ $\nu(2)$... and $\delta(r)$ a Cantor encoding of r (an irrational). Fix $\delta(r)$ as a weight of net *N*. Subnet N_1 receives input ω, measures $|\omega|$, and retrieves $\nu(|\omega|)$ from $\delta(r)$. ω and $\nu(|\omega|)$ are output for the other subnet N_2 which simulates a 2 tape TM on that finite input.

The decisive step is being able to fix the real weight $\delta(r)$. Since RRNs are equivalent to ATMs they compute functions that are not Turing computable. RRN are *beyond* the DTM limit. If exponential computing time is allowed, one can specify a RRN for each binary language: The class of languages recognised by a RRN in exponential time, **NET_R(EXP)**, includes *all discrete language*. The weights of RRNs, however, assume arbitrary real values. So it is practically impossible to *built* a net having such weights (this requires *infinite precision*). Updates also require infinite precision. There might *be* such nets (if there are irrational quantities in nature) but–at least–we cannot realise them. (DCT) is still in force! RRNs have to be considered, therefore, as ideal, *notional* machines. Variants that do not rely on infinite precision are *bounded precision neurons* and *stochastic nets*. *Linear precision* means that for up to q steps of computation *only* the first O(q) bits of both the weights and attraction values (of the neurons) influence the result. If a network computes σ in time T(n) the *T(n)-truncated version* computes the same response function σ on any input of length n (i.e. a truncated result.) Truncated values are not irrational values:

6. *Theorem*:

Computation within linear precision is not beyond DTM-computation.

Some irrationals are step-wise computable. *Recursive reals* are those the decimal expansion of which is the characteristic sequence of a recursive set.

7. *Theorem*:

Networks having recursive real weights compute only DTM-computable functions.

But even these reals give us speed up:

8. *Theorem*:

In polynomial time RRNs with recursive weights compute P/poly$\cap$REC.

Since P $\subset$ P/poly$\cap$REC (REC contains the recursive languages) these networks provide *speed up* without the use of non-recursive weights, although they *are not* computationally stronger than DTM.

A *stochastic network* (SRN) has additional input (stochastic lines). For all t>0 the stochastic line l_i has value 1 with probability p_i. The number of steps in all computations on ω is the same. The final decision on ω considers the ratio of *rejects* and *accepts* on ω. The probabilities are *real values* in [0,1]. A language $L \subseteq \{0,1\}^+$ is ε-recognised in time T by a stochastic network *N* if every word $\omega \in \{0,1\}^+$ is classified by every computation path of *N* in time T(|ω|) with bounded error probability $e_N(\omega) < \varepsilon < \frac{1}{2}$. So the decision on ω takes into account the ratio of *rejects* and *accepts* with an error probability of the whole computation. SRNs have real probabilities, but their weights can be integers, rationals or reals.

9. *Theorem*:

SRNs with integer weights are not computationally stronger than deterministic nets with integer weights (i.e. are *below* the Turing Limit, computing regular languages only).

10. *Theorem*:

Stochasticity does *not* add to RRNs computing power or speed up.

Adding stochastic lines does not matter in case of irrational real weights as RRN is already way beyond the Turing Limit. The interesting case are nets with rational weights, which have to be introduced first: Consider nets in which the neurons take *countably different values*; so do the weights. These values can be represented by **Q**. They are *Rational Recurrent Nets* (QRN).

11. *Theorem*:

A QRN *N* is finitely describable and can be simulated by a DTM *M*.

12. *Theorem*:

L is recognised by QRN *N* ⇔ L is recognised by a NTM *M*.

Put shortly: (T) QRN = DTM = NTM = NQRN

Non-deterministic QRN (NQRN) are *not* stronger than deterministic QRN. (T) rather *supports* (CT) by a new equivalence proof w.r.t. a concept of computation. QRNs *aren`t even faster* than TMs. There is no gain in complexity reduction. As seen stochastic lines do not matter with integer or irrational weights, but they do in case of QRN yielding a class SQRN.

13. *Theorem*:

QRN with stochastic lines compute BPP/log*. (<u>B</u>ounded error <u>P</u>robabilitic <u>P</u>olynom)

BPP/log* ⊂ P/poly, and BPP/log* already contains *non-computable f.*

14. *Theorem*:

DTM ⊂ SQRN ⊂ ATM.

So SQRN are *beyond* the Turing Limit. BPP/log* is one class in the ATM-hierarchy. It presents a further concept of computability [answering our (Q1)]. If the probabilities are *rationals* stochastic QRN compute BPP which *is* computable! Again: the stochastic lines with real probabilities *cannot be built* requiring *infinite* precision, as says (DCT). So let us then ask: What net is the brain?

In their book *The Computational Brain* Patricia Churchland and Terrence Sejnowski[10] are not precise what *kind* of computation occurs in the brain. On the one hand they claim it to execute vector mappings that are finite and deterministic. Being such it would not be beyond the Turing Limit. On the other hand they consider continous valued units. These might be RRNs. If the brain at the neuronal level can be finitely specified, it is not a RRN that is beyond the Turing Limit. If the brain is a RRN with at least one irrational weight it cannot be rebuilt by us. Must the brain master some task which requires such an RRN? – It seems difficult to say what type of network the brain is!

IV SIEGELMANN`S THESIS

(CT) deals with *algorithmic* computation. Natural process might be considered computational processes which are *continuos*. Taken thus there is a class of *analog computations*. RRNs being *one type* of these. Continous/"chaotic" processes in nature can only be *non-approximately simulated* by RRNs or other devices beyond DTM. With respect to analog/continous computation Hava Siegelmann claims[11]:

(ST) No possible abstract analog device can have more computational capabilities up to polynomial time than RRNs.

This means there is an *absolute concept* of analog computation. As (CT) claims for algorithmic computation. Has (ST) the same status as (CT)? On the one hand as (CT) it cannot be proven. Proven equivalences between abstract analog devices and RRNs give inductive evidence for it. On the other hand (ST) does not seem to relate an intuitive concept with a formal one.

Besides networks and oracles there are other forms of non-standard computation: *molecular computation*, *cellular automata* (CA) or *quantum computation* (QC).[12] Each of them represents a new type of computation. But their relationship to (CT) and standard computation is somewhat like in the case of (analog) networks.

So there are different types of computability some of which are *beyond* the Turing limit. There could be an *absolute* concept of analog computation (Siegelmann`s Thesis). (CT) is not falsified by super-computability since super-computability is not algorithmic in the usual sense. (DCT) is not falsified since the machines with super-computability cannot be built. Even if the brain was a super-computer the mind need not be, so (RTM) and (AI) would not be falsified even by that. Not only super-computability devices, but *also* Turing-computable networks, however, provide significant speed up. You might speculate whether this is a reason to assume that the mind is realised in such a structure.

(Manuel Bremer)

---

[10] (Cambridge/MA:MIT, 1996, 4th Edition).

[11] Cf. *Neural Networks and Analog Computation*, p.154.

[12] See as an overview: Tino Gramß et al., *Non-Standard Computation*. (Weinheim et. al., 1998).