Computing Machines Can't Be Intelligent (...and Turing Said So)

PETER KUGEL
Computer Science Department, Boston College, Chestnut Hill, MA 02467-3808, USA; E-mail: Kugel@bc.edu

Abstract. According to the conventional wisdom, Turing (1950) said that computing machines can be intelligent. I don't believe it. I think that what Turing really said was that computing machines — computers limited to computing — can only *fake* intelligence. If we want computers to become *genuinely* intelligent, we will have to give them enough "initiative" (Turing, 1948, p. 21) to do more than compute. In this paper, I want to try to develop this idea. I want to explain how giving computers more "initiative" can allow them to do more than compute. And I want to say why I believe (and believe that Turing believed) that they will have to go beyond computation before they can become genuinely intelligent.

Key words: abstract machines, Artificial Intelligence, Cognitive Science, hypercomputation, intelligence, limiting computability, models of mind, Putnam-Gold machines, trial-and-error machines

1. What I Think Turing Said

People who try to make computers more intelligent say they are trying to produce "Artificial Intelligence" (or "AI"). Presumably, they want the word "artificial" to suggest that the intelligence they are trying to create will — like artificial vanilla — not have developed naturally.

But some of their critics are convinced that anything that looks like intelligence in a computer will have to be artificial in another sense — the sense in which an artificial smile is artificial. Which is to say fake. Computers, they believe, cannot be genuinely intelligent because they lack a certain *je ne sais quoi* that genuine intelligence requires.

The more extreme of these critics believe that what computers lack is fundamental. Perhaps they believe that intelligence requires an immortal soul. Perhaps they feel that it can only be implemented in flesh and blood. Perhaps they believe that it requires human experiences or human emotions. Such critics believe that computers cannot be genuinely intelligent, period.

Other critics of AI are a bit more generous. They believe that computers cannot be genuinely intelligent until... Perhaps they believe that computers cannot be genuinely intelligent until they have access to better parallel processing or to special neural networks. Perhaps they believe that computers will need access to special units that can perform analog operations or utilize quantum effects be-

Minds and Machines **12:** 563–579, 2002.

fore they can become intelligent. These more moderate critics of AI believe that computers will only become capable of genuine intelligence when....

In this paper, I want to suggest that both AI and its critics may be right. I believe that computers already have everything they need to become genuinely intelligent, but that no amount of work along the lines that AI research typically takes today will get them there. I believe that today's AI is riding the right horse (the digital computer), but that it's taking it down the wrong road (computation).

And I believe that the problem lies, not in our computers, but in ourselves. Our computers can become genuinely intelligent, but only if we use them differently than we are using them today.

If you step back from the details, it is not hard to see where the problem lies. Turing (1948, p. 21) saw it more than fifty years ago when he suggested that, in order to be intelligent, computers would have to be given what he called "initiative".

Turing (1948, pp. 21–23) discussed "initiative", but he did not really define it other than to say that it was not "discipline" — the ability to follow orders given by others. Then he gave an example (Turing, 1948, p. 22):

> A very typical sort of problem requiring some sort of initiative consists of those of the form 'Find a number *n* such that....' This form covers a very great variety of problems. For instance problems of the form 'See if you can find a way of calculating the function which will enable us to find the values for arguments....' are reducible to this form.

Finding such a number is, he wrote, is "clearly equivalent to finding a program".

It is not surprising that intelligence in computers might require them to be given some initiative. You have to let people exercise initiative if you want them to behave intelligently. You cannot expect intelligence from people if you insist that they do exactly what you tell them to do in exactly the way you tell them. Yet, as long as we use computers only to compute, we are doing precisely that.

Computing is, of course, what computers were invented to do, but that does not mean that that they cannot do other things. Screwdrivers were invented to drive screws, but that does not mean that they cannot be used to pry open paint cans.

One of the things it means for a computer to compute is that it must give us answers with a sense of finality. If you ask a computer to multiply 1234 by 567, you expect it to give you the answer (699,678) and stop. You want a result that you can take back to your laboratory or accounting department.

You don't want the computer to act up. You don't want it to add "...but I'm not sure." And you certainly don't want it to call you back later — perhaps after you have used the number it gave you to approve a drug or report your company's earnings — to say it has "changed its mind". If anybody is going to change their mind when you're dealing with a computer, you want it to be you. ("Sorry, I made a typing mistake. I wanted you to multiply 1234 by 667.")

And that, I claim, is the problem. If you allow a computer enough "initiative" so that it can "change its mind", it can do things that it cannot do if you limit it

to computing. One of those things is to act independently. Another is to behave intelligently. I want to suggest that they may be related.[1]

Turing seems to have recognized that a machine's intelligence might depend on its ability to "change its mind". Speaking to the London Mathematical Society in 1947, he said (Turing, 1947, p. 124):

> ...if a machine is expected to be infallible, it cannot also be intelligent. There are several mathematical theorems which say almost exactly that. But these theorems say nothing about how much intelligence may be displayed if a machine makes no pretence at infallibility.

He also seems to have realized that intelligence would require more than computing. And, in the paper that is so often cited to support the claim that he said that computing would be enough, he quite clearly said that it would not (Turing, 1950, p. 459)[2]. I quote:

> Intelligent behaviour presumably consists in a departure from the completely disciplined behaviour involved in computation, but a rather slight one, which does not give rise to random behaviour, or to pointless repetitive loops.

Turing did not develop this idea very far but, in Turing (1939), he seems to have given it some thought. He seems to have realized that merely saying that computers must do more than compute before they can become genuinely intelligent does not tell you much. It is a bit like saying that China is not in South America. It tells you where computer intelligence is not, but not where it is.

If you believe that it takes more than computing to produce intelligence, you might want a map of the uncomputable so that you could figure out where, in the land of the uncomputable, intelligence lies. Turing tried to develop such a map in his Ph.D. thesis (Turing, 1939), but he did so in terms of what he called "oracles" that he did not tell us how to build (because he didn't know).

What he managed to show was that the land of the uncomputable *has* a geography — that some uncomputable functions are more uncomputable than others. But he did not tell us where in this geography intelligence might lie, nor how we might implement machines that could take us there.

Subsequently, Turing (1946) turned his attention to building actual computing machines. Once such machines had been built, he asked (in Turing, 1950) whether they could become intelligent. Contrary to the conventional wisdom, I believe that he argued that they could not. What I think he said was that:

- It will probably be possible to program computing machines — computers limited to computing — so that they can *fake* intelligence well enough to fool human beings.
- It will probably not (repeat not) be possible to program such machines — limited to computations — to be *genuinely* intelligent.

Turing's paper is better known for its first conclusion (although the business about faking it is usually played down). Turing's second conclusion is largely ignored, perhaps because it seems to contradict the main claim of his paper —

that computing machinery suffices for intelligence. However it does not if you pay attention to a subtle, but important, distinction. It is possible to argue that *computing machinery* can become genuinely intelligent while, at the same time, the *computing machines*, that are based on that machinery, cannot. There is a difference between *machinery* and *machines*. *-ery* matters.

A computer's *machinery* consists of the components of which it is made. Those components operate in certain ways when the machinery is not broken. In contrast, the *machine* is that same machinery *plus* a specification of how that machinery may be used. As a piece of machin*ery*, a screwdriver can be used to drive screws, pry open paint cans or clean your fingernails. As a machin*e*, it is limited to driving screws.

What we call a *Turing machine* is *Turing's machinery* (originally a tape, a read–write head and a control unit) limited to computing. That same machinery can be used in other ways. If, for example, we fix the size of its memory at the start, it becomes a *finite automaton*. Such a machine is considerably less powerful than a Turing machine, the size of whose memory is potentially unlimited.

When we use Turing's machinery to compute (i.e. as a Turing machine), we take its *first* output as its result. When we use that machinery as a more powerful machine that I propose to call [3] a *Putnam-Gold machine*, we take its *last* output as its result. That makes its outputs tentative because we cannot always know when an output is the last.

Notice that such "machines" use only the machinery of the Turing machine, but they use it in a different way that (as I will try to show) allows them to do more than compute. They are my hypercomputers and (as I will also try to show) they can help us get both a better understanding of what Turing (1950) was driving at and a better model of what it means to be an intelligent machine.

The role that computations play in the cognitive sciences today is very much like the role that numbers play in the physical sciences. Just as people working in the physical sciences often ask that accounts of the physical world be expressed in numerical terms, so people working in the cognitive sciences today often ask that accounts of cognitive processes be expressed in computational terms. The belief of many cognitive scientists is roughly this: "If you can't give a computational account of a cognitive process, you haven't really explained it."

This assumption helps keep cognitive scientists "honest" in several ways. One thing it does is to prevent them from ignoring critical details. When you try to give a computational account of a process — say the process by which we recognize occurrences of the letter "A" — you have to fill in details that you might otherwise ignore.

Another thing it does is that it helps prevent them from adopting theories that could not work. Once you fill in the details of a computational account you can program it for a computer, run the program, and see whether it does what you think it does, before you look to see whether the way the program does it is the same as the way that people do.

And, if your program really does the job you think it is going to do, you get a third benefit. A computer, running your program, can do that job for you.

It is largely because they enjoy these benefits of computational accounts of psychological processes that cognitive scientists are so eager to protect the view that "thinking can be reduced to computing" against its critics. They worry that, if this claim turns out to be unfounded, their science may again be reduced to kind of arm-waving that it used to involve when it was based on introspection, or the kinds of relatively trivial results it was able to obtain when it was based on the weaker finite-automaton model of behaviorism.[4]

I understand this motivation, and I would like to suggest that, even if real intelligence is beyond the reach of computability, it may not be necessary to discard computers as models for intelligence. It may be possible to extend the idea of the computer as a model in such a way that you end up keeping most of its benefits while, at the same time, stretching it enough to get that little extra something that intelligence requires.

I want to suggest that the Putnam-Gold machines do just the right amount of stretching. And I want to try to convince you that you have to do this much stretching if you want to produce genuine machine intelligence and to find adequate models for the genuine natural intelligence with which you and I like to believe we are so lavishly endowed.

2. Why Computing Is Not Enough

To see what's wrong with the idea that the human mind is a computing machine, suppose that it were. Suppose that it had the abilities of a general-purpose computing machine with a few programs, or something like programs, built in at the start. (We might call these built-in programs *instincts*.) Since these programs would probably not be enough, let's assume that the mind somehow develops additional programs to help it deal with its particular environment. (We might call the process by which this comes about *learning*.) Now suppose that all this mind can do to develop these new programs is to compute. How might it go about it?

We seem to have two basic choices. We can let the mind chose from among a (probably infinite) set of totally computable programs — programs that compute a result for every possible input. Or we can allow it to pick from a set of programs that includes partially computable ones — ones that might not compute results for some inputs[5].

Let's call a machine that computes only total functions *total* and one that can compute any totally computable function *universal*. When Turing (1936) defined his universal Turing machine, he faced a dilemma. He could have either a total machine or a universal machine, but he could not have both.[6] He chose the universal machine and that machine is the basis of today's general-purpose digital computer.

We face a similar dilemma when we try to design our computable person. If we want to endow that person with the capabilities of a programmable computing

machine, we can allow the machine to be either total or universal, but not both. Both options have drawbacks.

Suppose that we allow our human mind to use a universal machine to come up with a program to deal with predators. Since a universal machine can come up with a partially computable program, the human who uses a program developed by such a machine might get eaten by a predator for which a program it developed computed no response.

That problem would be avoided if we limited our human being to totally computable procedures. But now our human being might find itself in an environment with a predator for which its mind (not being universal) could not come up with a program.

Both cases could be detrimental to the survival of the species. Fortunately there is a third alternative. Let the mind fill any potential gap in the possibly partial programs before it tries to compute a value.

Suppose the mind is allowed to develop programs for a universal machine — whose values may be undefined for some inputs. To handle the undefined cases, let the mind start each computation off by producing (internally) an automatic response to use in case this is one of those no-result cases. Let it then run its program. If the program computes a result, the human uses it. If it does not, it uses the automatic response. This fills the gaps in the values of the partially computable functions with something — not necessarily the best something — that provides the mind with a response.

That's fine, but here's the kicker. If we do that, we end up with a machine that is no longer a computing machine because it can do things that computing machines cannot. One such thing is to solve the halting problem

The *halting problem* is the problem of finding a single program (*Halt*) that can, given — as its inputs — both an arbitrary program (*Prog*) and an input (*Inp*) to that program, determine whether or not Prog applied to Inp (or *Prog(Inp)*) will or will not halt. In other words, Halt, applied to Prog and Inp (or *Halt(Prog,Inp)*) must:

- Compute the output YES if Prog(Inp) halts.
- Compute the output NO if Prog(Inp) does not halt.

Turing (1936) proved that no computing machine could solve this problem. But a gap-filling machine can do it as follows. Let it output NO when it starts and then let it simulate the behavior of Prog operating on Inp. If the simulation halts, let it output YES.

It is not hard to see that this process always produces the right result if we take its result to be the last output it produces. And that it produces it, as a computation does, in finite time. But, in contrast to a computation, which gets at most one shot at producing a result, our hole-filling procedure gets two, if it "wants" them. Following the terminology of Putnam (1965) we might call such a procedure a *two-trial* procedure. (In these terms, a computation is a *one-trial* procedure. It is allowed only one try at an answer.) A two-trial procedure is allowed to make a guess and then it may (but need not) "change its mind".[7]

Two-trial machines are only the first of a series of machines that can be implemented by computing machinery and can do more than compute. The more tries you allow, the more powerful the machine you get. Three-trial machines (allowed at most[8] three tries) can do things that the two-trial kind cannot. And, in general, an $(n+1)$-trial machines can do things that n-trial machines cannot (Kugel, 1977).

If you allow an unlimited (but at most finite) number of outputs, or trials, for each input and count the *last* output a machine produces as its result, saying that the result is undefined if either there is no first or no last, you get a Putnam-Gold machine.

The difference between a Turing machine and a Putnam-Gold machine does not seem like much of a difference. You cannot tell whether a machine is a Turing machine or a Putnam-Gold machine by looking at its machinery. Both computing machines and Putnam-Gold machines follow the instructions of their programs a step at a time in a strictly "mechanical" way. But their results have a very different quality. When we take a machine's first output as its result we can be sure we have the final result as soon as we see it. When we take its last output as its result, we cannot (unless the machine shuts itself off). Its outputs are tentative and we have to remember that it may "change its mind".

Such machines implement Turing's (1948, p. 124) condition of making "no pretence at infallibility". In a sense, this idea was also anticipated by Gödel who wrote (Wang, 1974, p. 325) that:

> ...the mind, in its use, is not static, but constantly developing... (A)lthough at each stage of the mind's development the number of its possible states is finite, there is no reason why this number should not converge to infinity in the course of its development.

Gold (1965) suggested that what a Putnam-Gold machine does might be thought of as computing "in the limit" and that what it does resembles the way we compute the values of irrational numbers like $\sqrt{2}$ and π in the limit — getting closer and closer to the exact result at each step, but never getting its decimal expansion "exactly right".

It would be nice if we could get along without such numbers whose values are defined "in the limit".[9] But we can show that such numbers are needed to correctly characterize the diagonal of the unit square (which by the Pythagorean theorem is exactly $\sqrt{2}$ units long) and the circumference of the unit-diameter circle (which is exactly π units long).

To show that the extra power of a Putnam-Gold machine is necessary for intelligence, we need something in the cognitive world that corresponds to the diagonal of the unit square or the circumference of the unit circle — something that intelligent systems can do that cannot be characterized by a computation. We need, in other words, some aspect of intelligence that can only be implemented by a Putnam-Gold machine. And here we face a mildly annoying problem. When we talk of intelligence, we don't really know what we are talking about. There seems to be no generally accepted definition of what "intelligence" is.

Let's go look for one.

3. Intelligence and Computing Machinery

Toward the end of the 19th century, the French Ministry of Public Education asked Alfred Binet to develop a test to measure the ability of schoolchildren to learn. That test has come to be known as an "intelligence test" so, to start us off, let's assume that intelligence is (at least in part) the ability to learn.

To learn what? We learn lots of different kinds of things, but many of them can be represented by computer programs.[10] And many of the things we learn, we learn from examples. So let's go further and assume, not only that intelligence is the ability to learn, but also that it at least includes the ability to learn general ideas that can be represented by computer programs from examples. That seems to be what the child does when it learns the grammar of its native language from the utterances it hears and what it does when it learns the concept of "dog" after seeing a few tail-wagging examples.

Finding a program is quite different from running one. When we run a program, we go from a fully specified function, f_n, to its values $f_n(1)$, $f_n(2)$, and so forth. When we try to go "the other way" — from the values $f_n(1)$, $f_n(2)$, ... to the function, f_n we seem to "go beyond the information given", to use Bruner's (1973) felicitous phrase. We go from a finite set of values to the infinite values of the function. When we do that, some of the values we "predict" may be wrong even if all the values we are given are correct. If, for example, you know that $f_n(1)$, $f_n(2)$, $f_n(3)$, and $f_n(4)$ are all 2, there is nothing that prevents $f_n(5)$ from being 73. When you hear that the plural of "house" is "houses" and of "grouse" is "grouses", there is nothing that prevents the plural of "mouse" from being "mice".

This process of going from the values of a function to a program for evaluating that function is, I believe, a good model for the process of going from evidence to theories. Is it also a good model for at least part of what is involved in intelligence? I think so.

Notice that Binet's test tried to measure this ability indirectly. Most IQ tests do not try to see how well you can learn, but how well you have already learned. And they do that by seeing what "programs" you have learned to use by the time you take the test. There are obvious practical reasons for this, and it is how we tend to determine the intelligence of the people we meet. We see what they know and can do. But I suspect that what we often have in mind when we think of our intelligence is, not the knowledge we have acquired, but our ability to acquire such knowledge.

Some questions on IQ tests seem to try to measure this ability somewhat more directly. One popular type of question asks you to extrapolate sequences of integers. You might, for example, be asked to continue the sequence **2,4,6,8,**... The "correct" continuation is, of course, 2,4,6,8,**10,12,14,16,**... but that is not the only possible one. The sequence might have been 2,4,6,8,**2,4,6,8,**.... or 2,4,6,8,**8,8,8,8,**... or even 2,4,6,8, **who,do,we,appreciate**. The first continuation may be the "right"

one, but a person who stuck to it after being told that the fifth item in the sequence was "2" or "8" or "who" would be a fool. Yet that is exactly what a computing (or one-trial) machine would have to do if it chose its theory of a sequence after seeing its first four elements.

That does not seem very smart and it is one reason why computations may not be a good model for intelligence. After they come up with a theory, they are forced — by the definition of "computation" — to be stupid. They are forced to stick with their results, after evidence has shown them to be wrong.[11] (We might say that they can "know", but not "think".)

Now consider the machine that is allowed to keep the equivalent of an "open mind". When it has seen enough evidence to develop a theory, let it output that theory. But let it also change its theory when that theory no longer fits the evidence. And, instead of saying that the theory this machine produces is the *first* one it comes up with, let's say that it's the *last*. Notice that we now no longer insist that it tell us when it has finished. Which turns what used to be a computing machine into a Putnam-Gold machine and will make it (I claim) at least potentially capable of genuine intelligence.

Of course merely avoiding pig-headedness does not guarantee intelligence. The ability to change your theory when the evidence goes against it is not enough. The quality of the theories you come up with matters too. The theory that classifies emeralds as "grue" — green up to now, but blue ever-after — (Goodman, 1954) changes to remain consistent with the evidence (by changing the meaning of "now") as it sees more green emeralds, without being particularly "intelligent".

There are other kinds of questions on IQ tests that suggest aspects of intelligence that might also be modeled by Putnam-Gold machines.[12] Consider, for example, questions that ask you to categorize — to distinguish squares from circles, birds from dogs, and the like. Such categories have enough open-ended-ness to suggest the need for the kind of flexibility that Putnam-Gold procedures allow. As Wittgenstein (1958) has pointed out, concepts can always be stretched to accommodate new examples. The way we extend concepts over time, or narrow them with experience, shares some of the flexibility of the way we develop theories and may involve some of the same kinds of uncomputations. So, I believe, do many other kinds of thinking that we consider to require intelligence.

4. Genuine Intelligence

The idea that genuine intelligence may require more than computation is not new. Turing (1948) and Lucas (1961), among others, have suggested that human minds might be more powerful than computing machines because they can "violate" Gödel's (1931) incompleteness theorem according to which no consistent axiomatic system, powerful enough for arithmetic, can prove its own consistency.

But a Putnam-Gold machine can easily prove the formal consistency of any formal system of arithmetic (including a system that represents itself), without

thereby demonstrating its inconsistency. That's because such systems are not axiomatic systems.

To see how this might come about, consider Rosser's (1936) extension of Gödel's theorem to formal consistency. Recall that a system is said to be *formally consistent* if it cannot prove both a theorem, T, and its negation, ¬ T. Consider a procedure that takes, as input, a program that represents an axiomatic system A. To determine whether it is formally consistent, the procedure begins by outputting "CONSISTENT". Then it generates the theorems of A one at a time. Each time it generates a new theorem, it checks it against all the (finitely many) theorems already generated for a formal inconsistency. If it finds an inconsistency, it outputs "INCONSISTENT". Clearly its last output correctly determines the consistency or inconsistency of A. But the procedure is not a computation. It is the kind of two-trial procedure of which Putnam-Gold machines are capable.

Thus, if the mind is a Putnam-Gold machine, it can (so to speak) "violate" Gödel's (1931) incompleteness theorem. But, of course, it does not (really) violate that theorem because, like the theorem that says you can't trisect an angle with straightedge and compass alone, Gödel's theorem only tells us what cannot be done by certain means. Putnam-Gold procedures are not among the means it deals with.

Violating Gödel's theorem in this way is not something intelligent people typically do. (Most of them don't know what it is.) But they do do things that seem to involve such trial-and-error processes. Developing theories from evidence is one. And there are (I claim) plenty of others. They include understanding ordinary English, having a sense of humor, creating art, solving problems, and many other things.

There are computable ways to do such things as come up with theories from evidence and there are (as I have suggested) uncomputable ways. Let us say the computable ways involve *fake intelligence* and that the less "pig-headed" and uncomputable Putnam-Gold ways involve *genuine* intelligence. Fake intelligence is the kind that most of today's research in Artificial Intelligence seeks to develop in computing machinery.

There are at least two ways this can be done. A system that displays *fake intelligence* can compute its theories from evidence or it can take the theories developed by other systems that use genuine intelligence to develop them.

There is a lot to be said for fake intelligence. It allows us (and our computers) to use ideas developed by others, which has some of the advantages of the supermarket over the orchard. Developing it in both people and machines can be worthwhile for its own sake. And I believe that Turing thought that developing it in machines might be a good first step toward developing genuine machine intelligence.

This is the point that, I believe, Turing (1950) was trying to make when he suggested the *imitation game* as a target for research into machine intelligence. Recall that the imitation game is played between a computer and a person each of which communicates with a human judge via a computer terminal. Both try to convince the judge that they are the human and the computer wins if it succeeds.

To program a computer to play the imitation game, the way it is typically understood, is to program a computing machine to exercise fake intelligence. The AI researcher provides the program, which is the part of the whole job that requires genuine intelligence. The rest of the job — applying the program — is the fake part that the computer does. In some sense, genuine intelligence seems to involve both parts of the job and, it seems to me that Turing (1950) may have believed that if we could figure out how to do this second part of the whole job by machine, it might make it easier for us to figure out how to do the first — to develop the programs — by machine.

For a while, it was believed, by many, that the ability to play the imitation game well would be a good test for machine intelligence. This view was even attributed to Turing himself, although it seems unlikely that he held it. (See Copeland, 2000, p. 522.) He certainly did not say that he thought the question "Can a computer win the imitation game?" is equivalent to the question "Can machines think?" What he said (Turing, 1950, p. 433) was that it "is closely related to it".

When Turing was asked, in a BBC broadcast in 1952, if he had a definition of "think", he said (Turing, 1952, p. 466) that he was "unable to say anything more about it than that it was a sort of buzzing that went on inside my head". And then he added:

> The important thing is to try to draw a line between the properties of a brain, or of a man, that we want to discuss, and those we don't. To take an extreme case, we are not interested in the fact that the brain has the consistency of cold porridge. We don't want to say 'This machine's quite hard....so it can't think.'

Drawing that line, rather than defining "thinking" or "intelligence", may have been Turing's main reason for introducing the imitation game. Certainly, the way Turing (1950) introduced the imitation game suggests that he did not think of it as a test for genuine intelligence. He described it as a variation of a game in which a man tries to pretend that he is a woman. But surely nobody would think that a man was really a woman if he won such a game. A man in drag is not, after all, a woman, no matter how well his lipstick is applied.

Let me suggest that Turing may have suggested the imitation game as a first step toward genuine machine intelligence. For fifty years many people assumed it could also be the last. I don't think that Turing would have agreed.

5. So What?

The fact that computing machinery can be used both as a computing machine and as a more powerful Putnam-Gold machine confuses some people. There is an old saying that "If it walks like a duck and talks like a duck, it is a duck." A Putnam-Gold machine looks like a computing machine and behaves like a computing machine. So why is it an uncomputer (or what most of the contributors to this issue call a

hypercomputer)? The simple answer is " For the same reason a screwdriver (as a piece of machinery) can be a paint-can opener. Because we decide to use it as one."

One way to see the point of the distinction between computing machines and the more powerful Putnam-Gold machines is to compare it to the distinction between computing machines and the less powerful finite automata. Although computing machinery can be used to implement either, Chomsky (1956) was able to argue against behaviorism (successfully, I believe) with arguments based on this distinction. Chomsky (1957) argued that behaviorism could, in effect, be characterized as assuming that the mind was a finite automaton and then showing that this assumption was untenable. He suggested that some kinds of language processing required an infinite (or at least unlimited) memory which is unavailable to a finite automaton.

Allowing a memory space without a fixed size limit rather than with it doesn't seem like much of a difference. And it seems even less important if you realize that machines with truly unlimited memories cannot be built. (There is a strict limit on the amount of memory available to even the largest computer in the real world.) And yet the distinction has proved useful in both Computer Science and Cognitive Science. Looking at the computer *as though* it had an unlimited memory (or space) changes the way you use it and that, in turn, changes what you can do with it.

Where Turing machines allow unlimited space, Putnam-Gold machines, in effect, allow for unlimited (not necessarily computably long) periods of time. Looking at information processing systems — theoretical, electronic and biological — as though they had uncomputably long to come to their conclusions changes a number of things. It seems, to me, to make them more flexible, more independent, more fluid, more adaptable, and if I dare say so, more human, than either finite automata or Turing machines.

Using them as models might change the way we study intelligence in several ways:

- *Mathematics*: Gold (1965) suggested that Putnam-Gold machines might be thought of as computing their results "in the limit". The mathematics that deals with such limiting computations might play a role in the cognitive sciences similar to that played by the calculus (which also allows passing to the limit) in the physical sciences.[13]
- *Artificial Intelligence*: If we take the distinction between fake and genuine intelligence seriously, and recognize the potential value of both kinds of intelligence in machines, Artificial Intelligence might divide into two subfields. One would try to develop fake machine intelligence (which, in spite of the pejorative name I have given it, clearly has merit) and the other would try to develop genuine machine intelligence.
- *Computer Science*: Although Putnam-Gold machines can be implemented on today's digital computing machines, it might be worth trying to develop hardware on which they could be implemented more efficiently[14]. Software based on the trial-and-error (or limiting) computations of which Putnam-Gold machines are capable might, when developed, facilitate our ability to develop computer programs.
- *Cognitive Science*: Scientists studying human intelligence might want to join those studying machine intelligence in splitting into two sub-fields, one of which studied fake biological intelligence and the other of which studied genuine biological intelligence.
- *Brain Sciences*: Putnam-Gold machines suggest more flexible and adaptable models of the brain than Turing machines do, and such models might help us develop better accounts of how the brain works.
- *Education*: Educators might want to consider the possibility that developing genuine human intelligence might call for different educational methods than are called for to develop the fake kind. It is my opinion that what is called "back to basics" tries to develop fake human intelligence, whereas what is called "constructivism" tries to develop the genuine kind. It is also my opinion that both are worth developing.
- *Philosophy*: Various philosophical matters might be easier to deal with in terms of Putnam-Gold-machine models of the mind. Understanding free will might be one. Understanding Searle's (1980) "Chinese Room" might be another.

Such models might also offer a way to answer Lady Lovelace's (1843) objection that (in effect) computers cannot be intelligent because they can only do what they are told how to do. I believe that Turing (1950, p. 462) got it right when he said:

> I agree with Lady Lovelaces's dictum as far as it goes. But I believe that its validity depends on how digital computers *are* used rather than on how they *could* be used. (His italics)

If we use computers only to compute we are, in effect, telling them to "Do what we tell you how to do." If, on the other hand, we use them as Putnam-Gold machines to develop their own programs, we are telling them to "Go figure out what you want to be told to do for yourselves. Don't just cook with our recipes. Try to develop your own."

In other words, we are giving them more initiative. And that may be just what they need to become genuinely intelligent.

6. Conclusion

I have suggested that the answer to the question "Can computers be genuinely intelligent?" is "Yes and no." Computers can be genuinely intelligent, but only if we let them do more than compute.

Why is that surprising? Why would people think that they, or their computers, could be intelligent without "changing their minds", which is what restricting yourself to computations ties you to?

I suspect that people are drawn to this bizarre idea because they worry (and not without reason) about keeping their mind too open. Clearly there are times when you have to close it and compute. If you're driving a car and you see a truck pulling out of a side street in front of you, you'd better come up with an answer to the question "What shall I do?" quickly and with the sense of finality that a computation gives you. In such a situation, it is not particularly intelligent to spend a lot of time developing theories of truck behavior.

Another reason to close your mind about some things is that you don't want to think about them. For example, you can't do a very good job of driving a car if you keep wondering whether your steering wheel might break at any moment. If you keep testing it to make sure it's still OK, you will not only weave back and forth across the road, but you probably won't be able to pay enough attention to the oncoming traffic to avoid an accident. If you try to keep an open mind about everything, you will so overload your attention-paying capability that you won't be able to think effectively about anything.

Unfortunately, some people seem to conclude that, since they can't keep an open mind about everything, they shouldn't keep an open mind about anything. That accounts for some of the appeal of the computational model of intelligence according to which an intelligent mind closes itself once a theory or concept has been "found" or simply uses the theories given to it by others.

It's true that, once you've arrived at a theory of how your steering wheel works, you can forget about it and focus your attention on the road. You don't have to worry that turning it to the right will suddenly make the car turn left. But that doesn't mean that you can't, or shouldn't, keep an open mind about some things. You might, for example, want to keep an open mind about whether or not you've missed your turn.

It seems, to me, that there are merits in both a closed mind that can act quickly (to change the world) and an open mind that can come up with new ideas (to change itself). I suspect that intelligence requires both.

One problem with an open mind is a lack of certainty. The other day, a friend asked me what I was up to. I told her that I was writing a paper (this one) in which I was going to suggest that we change a basic paradigm of Cognitive Science and Artificial Intelligence.

"That's huge," she said.

"It's only huge," I replied, "if I'm right. And I don't know whether I am."

If we are Putnam-Gold machines (and I believe that, in some sense, we are), we cannot be sure that we are Putnam-Gold machines. All we can do is to assume that we are and see where that gets us.[15]

Let's.

Acknowledgements

My thanks to Jack Copeland, Jim Gips, Judy Kugel, Margaret Newhouse and Daniel Osherson for helpful suggestions (not all of which I took).

Notes

[1] Marvin Minsky (quoted in Storck, 1997) reports that AI co-founder Nathaniel Rochester was nearly fired when he "referred to the IBM 701 computer as 'smart'. The highest officials at IBM...wanted to reassure their potential customers that IBM products would only do what they were programmed to do."

[2] It is not easy to say why this sentence is so studiously ignored by people who claim that Turing (1950) argued that intelligence *could* be reduced to computation, but I will try to suggest some possible explanations of this curious phenomenon later in this paper.

[3] In previous papers (Kugel 1977, 1986) I called such a machine a "trial-and-error" machine. Putnam (1965) and Gold (1965) came up with the basic idea behind such machines and proved that they could do the uncomputable. I thought that it might be appropriate to name the machines after them.

[4] Recall that Chomsky's (1957) critique of behaviorism was based on this assumption.

[5] The function that computes the sum of two numbers is *totally defined* because its value is defined for any two numbers. The function that computes the result of dividing one number by another is *partially defined* because the result of dividing (say) 12 by 0 is undefined. But this function is totally computable because a computation can tell us, by looking for a 0 divisor, when a result is undefined. The function that computes the halting problem (defined below) is both partial and partially computable because it is undefined for some arguments and no computation can tell us every place it is not defined. (For a sketch of a proof, see the next note.)

[6] If it were possible to develop a programmable machine that could compute all and only the totally computable functions, it would also be possible to write a program to list all its programs in order (P_1, P_2, P_3, ...). In terms of this list you could write a program that would run each of these programs and thereby compute all the functions (f_1, f_2, f_3, ...) the machine can compute. Using this program, you could write another program that would totally compute a diagonal function, $d(n) = f_n(n) + 1$, whose value differs from the value of the n'th function in this list by one. This function is totally computable, but not computed by this machine (because it differs in at least one value from any function that is).

[7] A two-trial machine may not always be able to announce when it has produced its final result. A one-trial (or computing) machine always can.

[8] But not at least.

[9] In 1897, the Indiana State Legislature considered — but did not pass — a bill legislating the equivalence of π to a rational approximation on the grounds that the approximation was accurate enough for all practical purposes.

[10] Turing seems to have believed that at least some of our knowledge could be represented by programs. Thus, when Wittgenstein (1976, p. 31) asked "...how many numerals have you learned to write down?" Turing replied "...I should say aleph null". The only way that I can think of that Turing's finite head might contain the ability to write down an infinite number of numbers is by containing a program.

[11] Another approach to such errors is to attribute them to flaws in the data. That is sound if one is talking about what Chomsky (1965) has called "performance". But I am concerned here with what he called "competence".

[12] Answering these questions on IQ tests does not require such machines, of course, because IQ tests are time-limited.

[13] For some indications as to how this might go, see Jain et al. (1999), Martin and Osherson (1998), Kugel (1986) and Martin and Osheron's website: http://www.ruf.rice.edu/%7Eosherson/IL/ILpage.htm

[14] Other papers in this issue contain some ideas in this direction.

[15] People who study the machinery of the mind (Descartes and Skinner come to mind) typically assume that models of the mind can be found among machines of a specific type. Thus behaviorism thought they could be found among the finite automata. Today's AI typically assumes that they can be found among the Turing machines. I am suggesting that we broaden the search to the Putnam-Gold machines. But this may not be enough. If it is not, the same kind of maneuver that produced the Putnam-Gold machines from the Turing machines can be repeated again and again to produce a sequence of ever-more-powerful machines types (Kugel, 1977, pp. 322–328) beyond the Putnam-Gold machines among which intelligent machines may have to be found.

References

Bruner, J.S. (1973), *Beyond the Information Given: Studies in the Psychology of Knowing*, New York: W. W. Norton & Company.

Carpenter, B.E. and Doran, R.N., eds. (1986), *A.M. Turing's ACE Report and Other Papers*, Cambridge, MA: MIT Press.

Chomsky, N. (1956), 'Three Models for the Description of Language', *IRE Transactions on Information Theory* IT2, pp. 113–124.

Chomsky, N. (1957), 'Review of Skinner', *Language* 35, pp. 26–58.

Chomsky, N. (1965), *Aspects of the Theory of Syntax*, Cambridge, MA: MIT Press.

Copeland, B.J., ed. (1999), 'A Lecture and Two Radio Broadcasts on Machine Intelligence by Alan Turing', *Machine Intelligence* 15, pp. 445–476.

Copeland, B.J. (2000), 'The Turing Test', *Minds and Machines* 10, pp. 519–539.

Gödel, K. (1931), 'Über Formal Unentscheidbare Sätze Der Principia Mathematica Und Verwandter System I', *Montaschefte für Mathematik und Physik* 38, pp. 173–198.

Gold, E.M. (1965), 'Limiting Recursion', *Journal of Symbolic Logic* 30, pp. 28–48.

Goodman, N. (1954), *Fact, Fiction, and Forecast*, Cambridge, MA: Harvard University Press.

Jain, S., Osherson, D., Royer, J.S., Sharma, A. (1999), *Systems That Learn: An Introduction to Learning Theory*, second edition, Cambridge, MA: MIT Press.

Kugel, P. (1977), 'Induction Pure and Simple', *Information and Control* 35, pp. 276–336.

Kugel, P. (1986), 'Thinking May Be More Than Computing', *Cognition* 22, pp. 137–198.

Lovelace, A.A. Countess of (1843), 'Sketch of the Analytical Engine Invented by Charles Babbage, Esq., by L.F. Menabrea, of Turin, Officer of the Military Engineers, translation with extensive notes' in *Taylor's Scientific Memoirs III*, ed. R. Taylor, London: R. & J.E. Taylor.

Lucas, J.R. (1961), 'Minds, Machines and Gödel', *Philosophy* 36, pp. 112–127.

Martin, E. and Osherson, D. (1998), *Elements of Scientific Inquiry*, Cambridge, MA: MIT Press.

Melzer, B. and Michie D., eds. (1969), *Machine Intelligence 5*, New York: American Elsevier.

Putnam, H. (1965), 'Trial and Error Predicates and the Solution of a Problem of Mostowski', *Journal of Symbolic Logic* 20, pp. 49–57.

Rosser, J.B. (1936), 'Some Extensions of Some Theorems of Gödel and Church', *Journal of Symbolic Logic* 1, pp. 87–91.

Searle, J. (1980), 'Minds, Brains, and Programs', *Behavioral and Brain Sciences* 3, pp. 417–424.

Storck, D.G. (1997), *Hal's Legacy: 2001's Computer as Dream and Reality*, Cambridge, MA: MIT Press.

Turing, A.M. (1936), 'On Computable Numbers, With an Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society, Series 2*, 42, pp. 232–265.

Turing, A.M. (1939), 'Systems of Logic Based on Ordinals', *Proceedings of the London Mathematical Society, Series 2*, 45, pp. 161–228.

Turing, A.M. (1946/1986), 'Proposal for Development in the Mathematics Division of an Automatic Computing Engine (ACE)', in B.E. Carpenter and R.N. Doran, eds., *A.M. Turing's ACE Report and Other Papers*, Cambridge, MA: MIT Press.

Turing, A.M. (1947/1986), 'Lecture to the London Mathematical Society on 20 February 1947', in B.E. Carpenter and R.N. Doran, eds., *A.M. Turing's ACE Report and Other Papers*, Cambridge, MA: MIT Press.

Turing, A.M. (1948/1969), 'Machine Intelligence', in B. Meltzer and D. Mitchie, eds., *Machine Intelligence* 5, New York: American Elsevier.

Turing, A.M. (1950), 'Computing Machinery and Intelligence', *Mind 59* (N.S. 236), pp. 433–460.

Turing, A.M. (1952), 'Can Automatic Calculating Machines Be Said to Think?', in Copeland (1999).

Wang, H. (1974), *From Mathematics to Philosophy*, London: Routledge & Kegan Paul.

Wittgenstein, L. (1958), *Philosophical Investigations*, New York: McMillan.

Wittgenstein, L. (1976), *Wittgenstein's Lectures on the Foundations of Mathematics Cambridge, 1939*, ed. C. Diamond, Hassocks, Sussex: The Harvester Press.