CS 696i, Spring 2005, part III

Computer Vision

(Making Machines See)

and other miscellaneous stuff

Kobus Barnard

Computer Science, University of Arizona

Administrivia

First assignment should now be available:

http://kobus.ca/teaching/cs696i/spring05/

Off campus use will require:

login: me

pw: vision4fun

Characterizing computational intelligence

Summary from last time:

If we want to understand / learn from some system which seems intelligent, we should try to identify what problem is being solved.

If we want to implement intelligence (or anything) we need to know what we are trying to do.

Often the most compact explanation for complex behavior.

Having understood the problem, the complexity, as an implementation of a solution, likely makes more sense.

Characterizing computational intelligence

Further clarifications:

From the discussion: If we apply this reasoning to arbitrary intelligent behavior, then that behavior gets reduced to a “problem” that we might be able to solve with a computer---like many others such as “sorting”.

Characterizing computational intelligence

This suggests several possibilities:

- Intelligence is largely the solution by computational means of “problems” that can be stated (perhaps with some difficulty).
- Really intelligent stuff is computational, but we cannot even conceive of specifying the “problem”.
- Regardless of whether the behavior can be matched by a computational system, unless it is done the “human” way, we won’t call it intelligence.
- Some really intelligent stuff cannot be computed (at least under any currently conceived notion of “computation”)

Characterizing computational intelligence

This suggests several possibilities:

- Intelligence is largely the solution by computational means of "problems" that can be stated (perhaps with some difficulty).
- Really intelligent stuff is computational, but we cannot even conceive of specifying the "problem".
- Regardless of whether the behavior can be matched by a computational system, unless it is done the "human" way, we won't call it intelligence.
- Really intelligent stuff cannot be computed (at least under any currently conceived notion of "computation")

==> the title of the course refers to something that may not exist.

Characterizing computational intelligence

Remarks from the machine vision world:

While no one knows whether “intelligence” is computational (and this is one thing that keeps things really interesting), the machine vision field generally lives inside the first bullet:

In particular, we specify our problems and we work with computers.

Characterizing computational intelligence

Remarks from the machine vision world:

The only real source of evidence that the first bullet might wrong (in the field of machine vision) is the lack of success.

However, it is at least conceivable that the hard problems (such as general recognition) in vision could be “solved”.

If this is the case, then we still might not be satisfied that this is “intelligence” because the solution might not address how people do similar things.

Characterizing computational $\overset{\text{sub}}{\vee}$ intelligence

Remarks from the machine vision world:

The only real source of evidence that the first bullet might wrong (in the field of machine vision) is the lack of success.

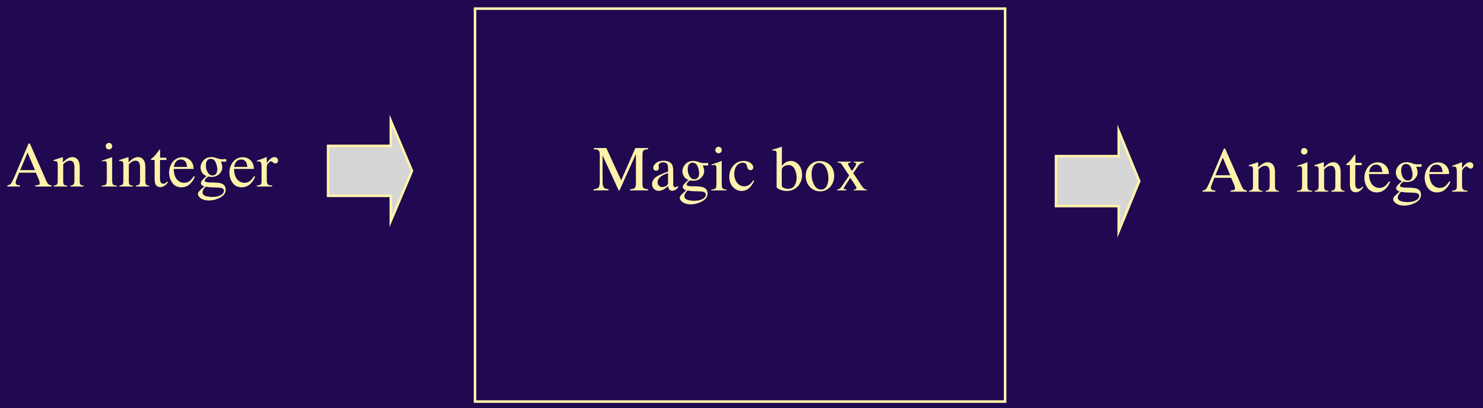
However, it is at least conceivable that the hard problems (such as general recognition) in vision could be "solved".

If this is the case, then we still might not be satisfied that this is "intelligence"

Solution ==> Rename the course for this section.

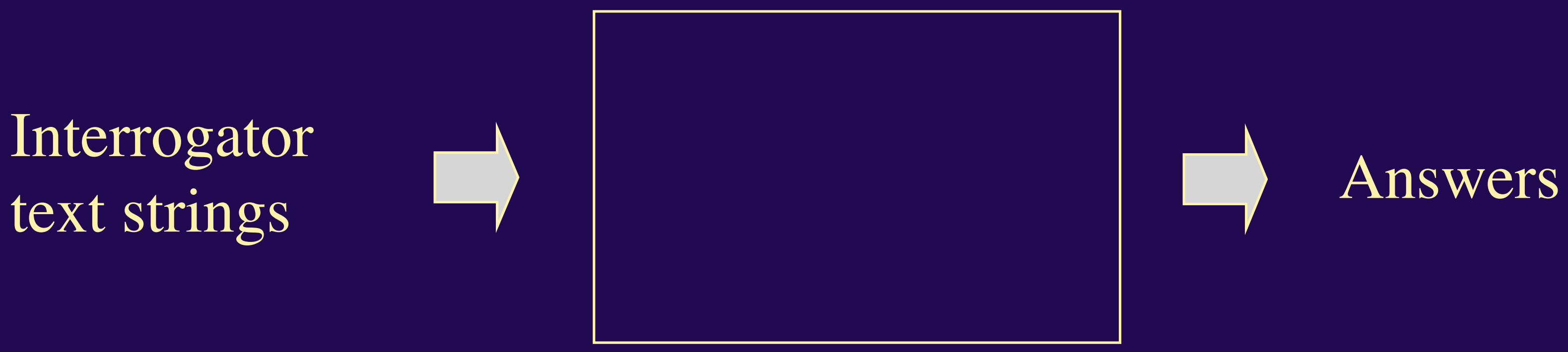
Characterizing computational intelligence

A computer (program) maps inputs to outputs. A digital computer maps discrete entities (representable by integers). So, a caricature of computational intelligence might look like:



Characterizing computational intelligence

Example: “The Turing Test”



“What is the meaning of life?”

087 104 097 116 ....

“A Monty Python movie”

063 032 077 111 ....

Stolen from: http://www.turing.org.uk/turing/

Who was Alan Turing?

Founder of computer science, mathematician, philosopher, codebreaker, strange visionary and a gay man before his time:

1912 (23 June): Birth, Paddington, London
1926-31: Sherborne School
1930: Death of friend Christopher Morcom
1931-34: Undergraduate at King's College, Cambridge University
1932-35: Quantum mechanics, probability, logic
1935: Elected fellow of King's College, Cambridge
1936: The Turing machine, computability, universal machine
1936-38: Princeton University. Ph.D. Logic, algebra, number theory
1938-39: Return to Cambridge. Introduced to German Enigma cipher machine
1939-40: The Bombe, machine for Enigma decryption
1939-42: Breaking of U-boat Enigma, saving battle of the Atlantic
1943-45: Chief Anglo-American crypto consultant. Electronic work.
1945: National Physical Laboratory, London
1946: Computer and software design leading the world.
1947-48: Programming, neural nets, and artificial intelligence
1948: Manchester University
1949: First serious mathematical use of a computer
1950: The Turing Test for machine intelligence
1951: Elected FRS. Non-linear theory of biological growth
1952: Arrested as a homosexual, loss of security clearance
1953-54: Unfinished work in biology and physics
1954 (7 June): Death (suicide) by cyanide poisoning, Wilmslow, Cheshire.

Alan Turing in 1946.

Characterizing computational intelligence

Example: Vision

Input:
Image

(50,123,24)

Entire image is a sequence of 256x384 triples of integers between 0 and 255. Obviously we can string them together to get one giant integer.

Characterizing computational intelligence

Example: Vision

Input:

Output:

(1) A text string

"A tiger is in the grass. Back off, real slow"

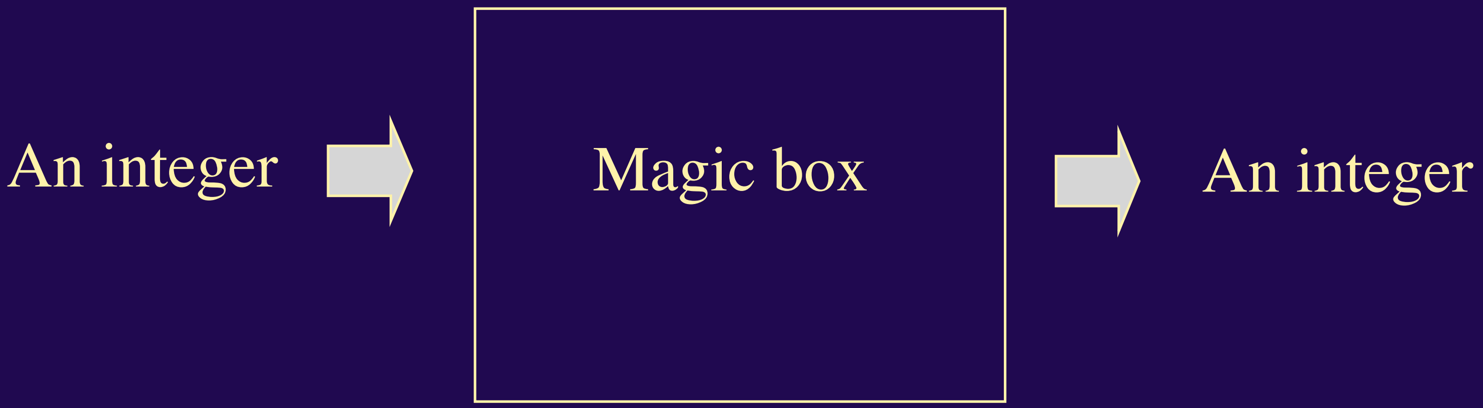
(2) A voltage to a Breitenberg vehicle wheel

Result: "Agent backs off, real slow"

Notice that there are many different output integers that might impress you, suggesting that the system is "intelligent"

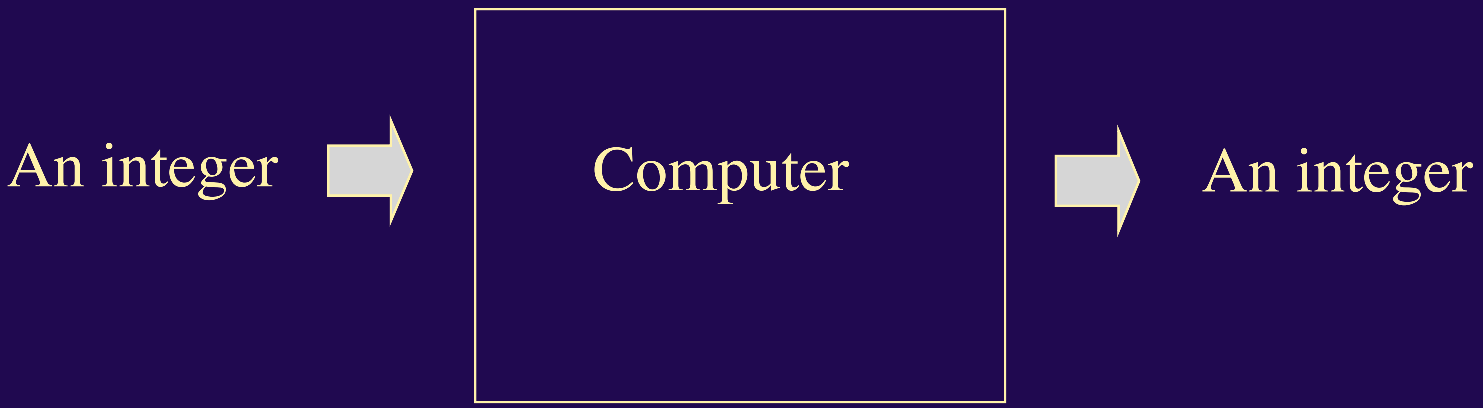
Characterizing computational intelligence

Discussion question: In the caricature below, is the representations of the input and output and output appropriate, or do we need something else?



Characterizing computational intelligence

We will now assume that we want to put a computer in the box. Thus, our system is a function, mapping integers to integers.



Technical comment: While we can argue that the size of integer that a system could ever see is bounded, it may not be fruitful to rely on this when arguing about what should be in the box---discussion point for later.

Characterizing computational intelligence

We will now assume that we want to put a regular computer in the box to implement a function mapping integers to integers.

We appeal to the theory of computation to ask questions like:

1) What constitutes a computer? Does a mac count?
2) Does there exist a computer (program) for an arbitrary input/output function. IE, given a set of input integers, and a set of output integers, can we be sure that there is a computer (program) linking them?
3) If one exists for a particular problem, what might be the requirements for the resources inside the box. (How long do have to wait / how much memory do you need?).

Lessons from the theory of computation

Theorem: There are input/output relations that do not have a program to link them.

Lessons from the theory of computation

Theorem: There are input/output relations that do not have a program to link them.

Rephrased: There are things that you might conceive that a computer could do, that there is no computer that can do it.

Lessons from the theory of computation

Theorem: There are input/output relations that do not have a program to link them.

Rephrased: There are things that you might conceive that a computer could do, that there is no computer that can do it.

Intuitive reason: There are more functions on integers than there are computers.

Lessons from the theory of computation

Theorem: There are input/output relations that cannot have a program to link them.

Rephrased: There are things that you might conceive that a computer could do, that there is no computer that can do it.

Intuitive reason: There are more functions on integers than there are computers.

Example: Halting problem---given an integer representing a program and its input, compute whether that program would terminate working on that input.

Lessons from the theory of computation

Aside: What is a computer?

In studying computation, we define various theoretical idealizations of machines, constructed for ease of reasoning about what they can do. All of immediate interest are essentially equivalent to a standard computer. Understanding this is part of a CS education.

The canonical example is a "Turing Machine" which is a model of effectively computable procedure (including all algorithms).*

* This phrasing, as with some other material on this topic was stolen from Pete Downey's CS573 slides.

Basic Turing machine specification

A Turing machine operates on an infinitely long “tape” which contains the input. The tape can be both read and written.

The machine has a finite control which, based on a current state and input, can write a symbol on the tape, move the tape left or right, and make a transition to a different state.

Important: We can associate a machine with an integer.

Exercise left to the student: This can do all that a mac can.

Unsolvability continued

Recall the claim: There are input/output relations that do not have a program to link them.

We can prove this using only the fact that computers maps onto integers.

Unsolvability continued

Proof (by contradiction): Suppose that every function mapping integers to integers was computable. Every Turing machine has an index, i. So, if the assumption was true, we can identify with each function, f(j), a machine, i. Denote this by $f_i(j)$.

Now consider a function, g(j) such that $g(j)=f_j(j)+1$. What machine computes g(j)?

By construction, g(j) is different from each of the Turing machine computable functions at the value j. Thus there is no machine computing g(j).