Contents

What is Computational Intelligence?

The study of the design of *intelligent agents*. An *agent* is something that acts in an environment. An *intelligent agent* is an agent that acts intelligently:

- Its actions are appropriate for its goals and circumstances
- It is flexible to changing environments and goals
- It learns from experience
- It makes appropriate choices given perceptual limitations and finite computation

Agents Acting in and Environment

Agent is a black box. Intelligence of the agent is judged by its actions. Starts off with prior knowledge, past experiences, goals/values and observations.

Central Hypotheses of CI

Symbol-system hypothesis:

- Reasoning is symbol manipulation

Church-Turing thesis:

- Any symbol manipulation can be carried out on a Turing machine

Challenges to Reasoning

Assults against

- Truth
  - Liar's PAradox: "I am lying"
- Sets/membership ϵ
  - Russell set $R = x \mid not\ x \epsilon x$
- Countability
  - Cantor: Power$(0, 1, 2, ...)$
- Change
  - Sorites : heap (minus one grain)
- Computability
  - Turing: Halting Problem

Artificial or Computaional Intelligence?

- The first is often called *Artificial Intelligence*
- *Scientific goal*: to understand the principles that make intelligent behaviour possible, in natural or artificial systems
- *Engineering goal*: to specify methods for the design of useful, intelligent artifacts
- Analogy between studying flying machines and thinking machines

Marr's Tri-Level Hypothesis

The three levels at which any machine carrying out an information-processing task must be understood.

1. **Computation theory**: What is the goal of the computation, why is it appropriate, and what is the logic of the strategy by which it can be carried out?
2. **Representation and algorithm**: How can this computational theory be implemented? In particular, what is the presentation for the input and output, and what is the algorithm for the transformation?
3. **Hardware implementation**: How can the representation and algorithm be realised physically?

Importance

Although algorithms and mechanisms are empirically more accessible, it is the top level, the level of computational theory, which is critically important from an information-processing point of view. The reason for this is that the nature of the computations that underlie perception depends more upon the computational problems that have to be solved than upon the particular hardware in which their solutions are implemented. To phrase the matter another way, an algorithm is likely to be understood more readily by understanding the nature of the problem being solved than by examining the mechanism (and the hardware) in which it is embodied.

In a similar vein, trying to understand perception by studying only neurons is like trying to understand bird flight by studying only feathers: It just cannot be done. In order to understand bird flight, we have to understand aerodynamics; only then do the structure of feathers and the different shapes of birds' wings make sense.

Abstractions over Turing Machines

1. **Computation theory**: Specify an input/output relation, e.g. given a string S, return its reverse S^{rev}
2. **Representation and algorithm**: Represent a Turing machine as four lists MR, ML, WL, HL and find a Turing machine $[mr, ml, wl, hl]$ satisfying the I/O spec (with respect to a suitable representation)
3. **Hardware implementation**: Run the turing machine $[mr, ml, wl, hl]$

There are many programming languages that we can use in place of Turing machines - choices can be arbitrary