Artificial Intelligence Fundamentals

Federico Matteoni

A.A. 2021/22

Index

0.1 Introduction

Prof.s: Maria Simi, Vincenzo Lomonaco
AI is taking over the world. Formalizing common sense is a lot more difficult. We can formalize knowledge in very specific and small domains. But is deep learning the final solution to AI? "It will transform many industries, but it's not magic. Almost all of AI's recent progress is based on one type of AI, in which some input is used to quickly generate simple response." (*Andrew Ng*)
This AI can do supervised learning, but requires huge amount of data (tens of thousands of pictures to build a photo tagger, for example). The rule of thumb of Ng is: if a person can do a mental task with less than one second of thought, we can automate it using AI either now or in the near future.
The challenges are:

Software is not a problem, the community is open and the software can be replicated the software can be replicated

Data is exceedingly difficult to get access to. Data is the defensible barrier for many businesses

Talent, because downloading and applying open-source software to your data won't work. AI needs to be customized to context and data, that's why there's a war for the scarce AI talent that can do this work.

Computational resources are also very important.

Deep Learning Is only one approach inside the much wider field of ML and ML is only one approach in the wider field of AI. Book: *Thinking Fast and Slow*, Kahneman. Two systems: system 1 does perceptual tasks, simple computations, system 2 instead does complex computation, recalling from memory... this is a distinction in our brains.

Machine Learning Is AI all about machine learning? Possible arguments against ML are:

Explanation and accountability: ML systems are not (yet?) able to justify in human terms their results. For some applications this is essential: knowledge must be meaningful to humans to be able to generate explanations? Some regulations requires the right to an explanation in decision-making, and seek to prevent discrimination based on race, opinions, sex... (see GDPR)

ML systems learn what's in the data, **without understanding what's true or false, real or imaginary, fair or unfair**. It is possible to develop unfair, bad models. People are generally more critical about information.

Building AI systems is a goal far from being solved, still quite challenging. Complex AI systems requires the combination of several techniques and approaches, not only ML.

AI Fundamentals Is mostly about reasoning and *slow thinking.* Different approaches, "good old-fashioned artificial intelligence" or "symbolic AI": teaching about the foundations of the discipline, now 60 years old.

Symbolic AI High-level human readable representations of problems, the general paradigm of searching for a solution, knowledge representation and reasoning, planning. Dominant paradigm from the mid 1950s until late 1980s. Central to the building of AI systems is the physical symbol systems hypothesis (PSSH), formulated by Newell and Simon (*Computer Science as Empirical Inquiry: Symbols and Search*)
The approach is based on the assumption that many aspects of intelligence can be achieved by the manipulation of symbols (the PSSH): *a physical symbol system has the necessary and sufficient means for general intelligent action.*
Human thinking is a king of symbol manipulation system (so a symbol system is **necessary** for intelligence), and machine can be intelligent (a symbol system is **sufficient** for intelligence). This cannot be prove, we can only collect empirical evidence: observation and experiments on human behavior in tasks requiring intelligence, and solving tasks of increasing complexity.

Strong and Weak AI The Chinese room argument, by John Searle, introduced the following distinction: strong ai relies on the *strong* assumption that human intelligence can be reproduced in all its aspects (general AI), including adaptivity, learning, consciousness..., while weak AI is the simulation of human-like behavior, without effetctive thinking or understanding, no claim that it works like the human mind. Dominant approach today, fragmented AI.
One strong argument against strong AI is the lack of needs by the systems: biological need, safety, relationships, self esteem, self-actualization (Maslow's hierarchy of needs).
What stands in the way of all-powerful AI is not a lack of smarts: it's that computers can't have needs, cravings or desires.

AI is the enterprise of building intelligent computational agents

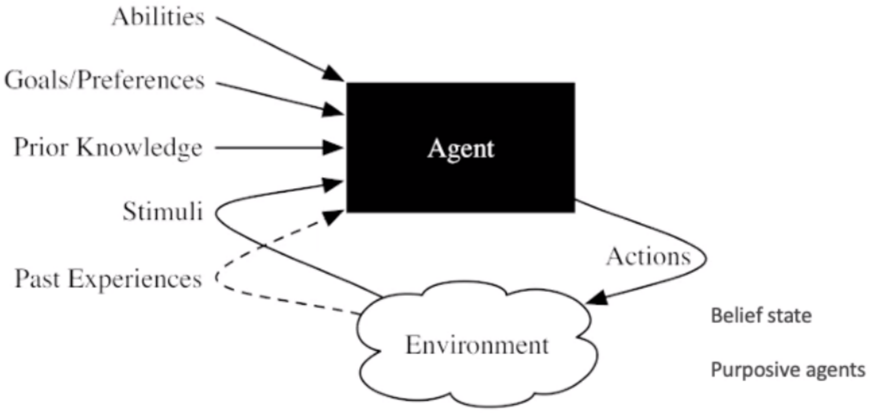
Agents An agent is something that acts in an environment. We are interested in what an agetn does, that is how it acts. We judge and agent by its action. An agent acts intelligently when: what it does is appropriate given the circumnstances and its goals, it is flexible to changing envoronments and changing goals, learns from experience, makes appropriate choices given its perceptual and computational limitations.
Computational agent is an agent whose decsiions about its actions can be explained in terms of computation and implkemented on a physical device.

Scientific Goal: undestrand the principles that make intelligent behavior possibile in natural or artificial systems

Engineering Goal: design and synthesis of userful, intelligente artifacts, agents that are useful in many applications

Artificial Intelligence Artificial intelligent is not the opposit of real intelligence. Intelligence cannot be *fake*: in an artificial agent behaves intelligently, it is intelligent. It is only the external behavior that defines intelligence, according to the **Turing Test** (weak AI). So **artificial intelligence is real intelligence created artificially**.
More updated test: Winograd schemas.
Human intelligence: biology (surviving various habitats), culture (language, tools, concepts, wisdom passed from parents and teachers to children) and life-long learning experience (learning throughout life). Another form is social intelligence, exhibited by communities and organizations.

So agents are situated in environments, inputs are abilities, goals, prior knowledge, stimuli and past experiences, and outputs actions which affect the environment.



Design process

design time computation, that goes into the design

offline computation, that the agent can do before acting in the world (ex: specializing the model)

online computation, done by the agent that is acting

Designing an intelligent agent that can adapt to complex environments and changing goals is a major challenge. Two strategies: simplify environments and build strong reasoning systems for these simple environments, or build simple agents for natural/complex environments simplifying the task.
Steps in the design process:

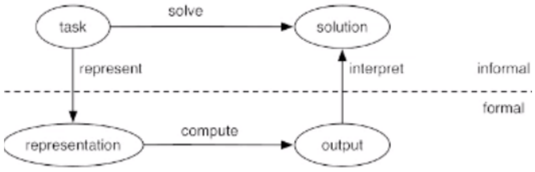
define the task in natural language, what need to be computed

define what is a solution and its quality: optimal, satisfying, aproximately optimal, probable...

formal representation for the task, choosing how to represent knowledge for the task, including representations suitable for learning.

compute an output

interpret output as solution



Levels of abstraction A model of the world is a syumbolic rtepresentation of the beliefs of the agents. In is necessarily an abstraction: more abstract representations are simpler and human-readable but they may not be effective enough. Low level descriptions are more detailed and accurate but more complex too. Multiple level of abstractions are possibile (hierchical design). Two levels always present in the design: knowledge level (what the agent knows and its goals, not in terms of how we represent) and the symbol level (internal representation and reasoning system). **Modularity** extent to which a system/task can be decomposed

flat: not modular

modular: interacting modules that can be understood on their own

hierarchical: modules are decomposed into simpler modules

Planning horizon how far ahead in time the agent plans

non planning agent

finite horizon planner: looks for a fixed amount of stages, greedy if only one step ahead

indefinite horizon planner: finite but not predetermined number of stages

infinite horizon planner: keeps planning forever (ex: stabilization module of a legged robot)

Representation concerns how the state of the world is described

Atomic states

feature-based representation: set of propositions that are true or false (PROP, CSP, most ML)

individuals and relations, or relational representation

Computational limits that determines whether an agent has

perfect rationality, reasons about the best actions without constraints

bounded rationality, decides on the best action that it can find given its limits

An anytime algorithm is an algorithm where the solution improves with time.
Learning dimensions determines whether

knowledge is given in advance, or

knowledge is learned (from data or past experience)

Learning typically means finding the best model th **Uncertainty**, which can be

in sensing (fully/partially observable states)

about the effects of the actions (deterministic/stochastic)

Preference dimension which considers thetehr the agent has

goal (achievemtn goal a proposition true in a final state, or mainentance goal, proposition true in all psobbile states)

complex preferences, involving trade-offs among the desiderability of various...

Number of agents

single agent reasoning

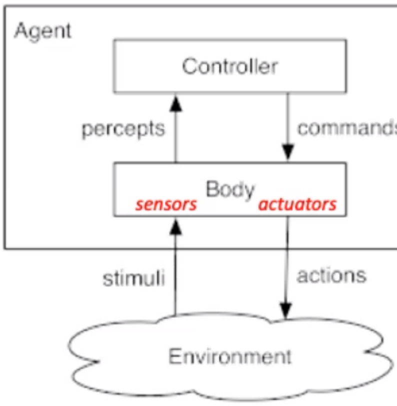
multi agent resoning

Interaction considers wheter the agent does:

offline reasoning or

online reasoning

Agent Architectures Agent interacts with an environment, receives informations with sensors and acts in the world with actuators. Robot: physical body. Program: software agent, digital environment.
Agent is made of body and controller, which receives percepts from the body and sends command to the body. A body includes sensors that converts stimuli into percepts and actautors that convert commands into actions.



Bot sensors and Agents act in time. T is a set of time points, with start at 0, totally ordered, discrete and each t has a next time $t+1$.
Percept trace/stream: function of time into percepts (past, present, future)
Command trace: a function of time into commands (past, present, future)
History at time t: percepts up to t and commands up to $t-1$

Causal Trasduction Function from history to commands. Transduction comes from *finite state transducers*, where both new states and commands are emitted. "Causal" because only previous and current percepts and previous commands can be considered. A controller ideally implements a causal transduction.
But complete history is usually unavailable, only the memory of it. The belief state of an agent at time t is all the information that the agent remembers from the previous times. The behavior of an agent can be described by two functions:

Belief State function $remember : S \times P \to S$ with S being the set of belief states and P the set of percepts

Command function $command : S \times P \to C$ with C being the set of commands.

The controller implements both, an approximation of the causal transduction.

Problem Solving as search The dominant approach to AI is formulating a task as a search in a state space. The paradigm is as follows:

Define a goal (a set of states, a boolean test function...)

Formulate the task as a search problem: define a representation for states and define legal actions and transition functions

Find a solution (a sequence of actions) by means of a search process

Execute the plan

This is a basic technique in AI: search happens inside the agent, it's the planning stage before acting. It's different from searching the world, when an agent may have to act in the world and interleave an action with planning.
Search is a general paradigm, underlying much of the artificial intelligence field. An agent is usually given only a description of what it should achieve, not an algorithm to solve it. The only possibility is to search for a solution. Searching can be computationally very hard (NP-Complete).
Humans are able to solve specific instances by using their knowledge about the problem. This extra knowledge is called **heuristic knowledge**.

Assumptions in classic problem solving Problem solving agents are goal driven agents, that work under simplified assumptions made in the design process.

States are treated as black boxes: we only need to know the heuristic value and whether they are a goal by applying the boolean goal function. The internal structure doesn't matter from the point of view of search algorithms. **Atomic representations**.

The agent has **perfect knowledge** of the state (full accessibility), no uncertainty in sensors.

Actions are **deterministic**, so that the agent know the consequences of its actions.

The state space is generated incrementally: can be infinite, so may not fit in memory.

Problem formulation A problem is defined formally by five components:

Initial state

Possible actions in state s, $\mathit{Actions}(s)$

Transition model: a function $\mathit{Result} : \mathit{State} \times \mathit{Action} \rightarrow \mathit{State}$
$\mathit{Result}(s, a) = s'$, a **successor state**

Goal states are defined by a boolean function
$\mathit{Goal\text{-}Test}(s) \rightarrow \{\mathit{true}, \mathit{false}\}$

Path-cost function, that assigns a numeric cost to each path. The sum of the cost of the actions on the path $c(s, a, s')$

Graphs for searching A (directed) graph consists of a set N of nodes and a set A of arcs, which are ordered pairs of nodes. Node n_2 is a neighbor/successor of n_1 if $\exists\,(n_1, n_2) \in A$, and a path is a sequence of nodes $(n_0, \ldots, n_k)$ such that $(n_{i-1}, n_i) \in A$ with length k.
The cost of the path is the sum of the costs of its arcs $\mathit{cost}((n_0, \ldots, n_k)) = \sum_{i=1}^{k} \mathit{cost}((n_{i-1}, n_i))$.
A **solution** is a path from a start node to a goal node and an **optimal solution** is one with minimum cost.

Search algorithms A problem is given as input to a search algorithm. A solution to a problem is a path (actions sequence) that leads from the initial state to a goal state.
Solution quality is measured by the path cost function and an optimal solution has the lowest path cost among all solutions. Different strategies (algorithms) for searching the state space may be characterized by:

Their time and space complexity, completeness, optimality…

Uninformed search methods vs informed/heuristic search methods, which use an heuristic evaluation function of the nodes

Direction of search (forward or backwards)

Global vs local search methods

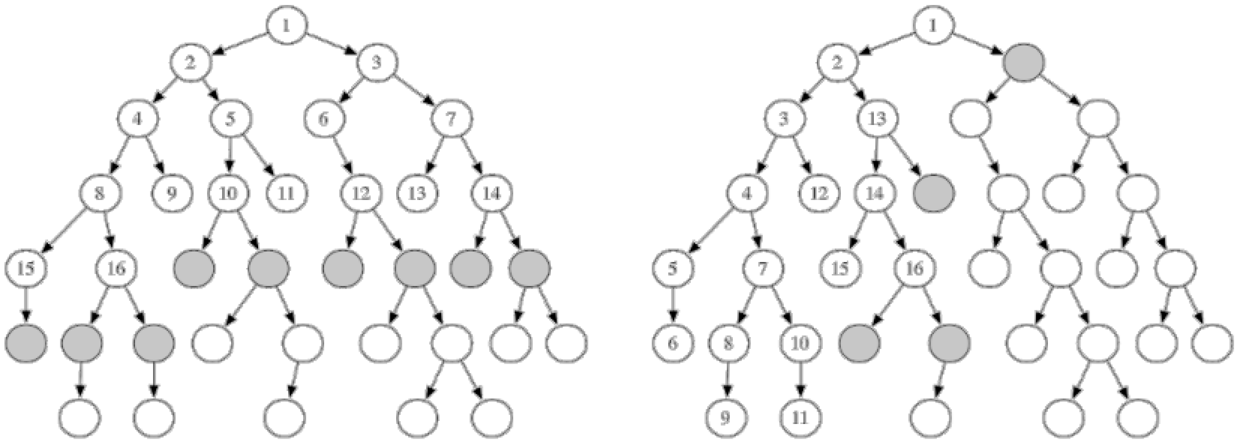
Generic search algorithm

```
input:
a graph
a set of nodes
boolean function goal(n) that tests if n is a goal node

frontier := {s | s is a start node}
while frontier is not empty:
select and remove path (n0, ..., nk) from frontier
if goal(nk):
return (n0, ..., nk)
for each neighbor n of nk
add (n0, ..., nk, n) to frontier
end while
return fail
```

Other algorithms With b max number of successors, d depth of solution and m max distance of solution.

Breadth and Depth search Respectively:



Breadth search: complete, optimal, time $O(b^d)$, space $O(b^d)$
Depth search: not complete, time $O(b^m)$, space $O(bm)$

Depth bounded search: supposes to know the distance of the solution, performs depth-first up to a limit without giving up completeness

Iterative deepening: tries depth limit 1, then 2, then 3 and so on, freeing memory from one iteration to the next.

Uniform cost search: at each stage, selects a path on the frontier with lowest cost.
The frontier is priority queue ordered by path cost, so the first path to goal is the least-cost path. When arc costs are equal is equivalent to breadth-first search.
This strategy is complete, provided that the branching factor is finite and there is some $\epsilon > 0$ such that all the costs are $> \epsilon$. It's also optimal, since it guarantees that the paths with lower costs are found first.

Heuristic search The idea is to not ignoring the goal when selecting the paths. Often there's extra knowledge that can be used to guide the search: **heuristics**, provided by an heuristic function $h : N \rightarrow R \Rightarrow h(n)$ is the estimate of the cost of the shortest path from node n to the goal node.
h needs to be efficiently to compute. An **admissible heuristic** $h^*(n)$ is a non-negative heuristic function that **underestimates** the minimum cost of a path from a node n to a goal: $\forall\, n \;\; h(n) \leq h^*(n)$

Best first search selects the most promising node on the frontier according to the heuristic function.

A^* **search** With an heuristic function in the form $f(n) = g(n) + h(n)$ with:

$g(n)$ being the cost of path leading to n (so the previous path up until n, *cost*(n))

$h(n)$ is an admissible heuristic (so, $h(n) \geq 0$)

Then $f(n)$ estimates the total path cost of going from a start node to a goal via n. The special cases are $h = 0$ (lowest cost search) and $g = 0$ (greedy best first).
Properties of A^*:

Complete

Always finds an optimal solution, if the branching factor is finite and arc costs are bounded above 0 (which means that $\exists\, \epsilon > 0 \mid$ arc costs are $> \epsilon$)

Optimiziations are possibile when searching graphs

The operate some sort of graph pruning:

Cycle pruning: doesn't add nodes to the frontier with states already encountered along the path (easy)

Multiple-path pruning: maintains an explored set of nodes that are at the end of paths that have been expanded. When an n is selected, if its state is already in the explored set, it's discarded.

Memory requirement is exponential ($O(b^d)$). Can be mitigated in some ways:

IDA^*: performs repeated depth-bounded searches with value of $f(n)$ used as bound

Recursive best-first, similar to branch & bound

SMA^* (simplified memory-bounded A^*)

Beam search, keeps in frontier only the best k paths, with k being the beam width (gives up optimality)

Consistent heuristics An heuristic that statisfies the monotone restriction guarantees consistency $h(n) \leq cost(n, n') + h(n')$
Consistency ⇒ admissibility. With the monotone restriction, the f-values of the paths selected from the frontier are monotonically non-decreasing.

Features Often better to describe states in terms of features: **factored representation**, more natural and efficient than explicitly enumerating states. Often, features are not independent and there are constraints that specify legal combinations of assignments. We can exploits these constraints to solve tasks.
Constraint satisfaction is about generating assignments that satisfy a set of hard constraints and how to optimize a collection of soft constraints (preferences).

CSP **Constraint Satisfaction Problem**, formal definition. A Constraint Satisfaction Problem $CSP = \langle X, D, C \rangle$ consists of three components:

A finite set of **variables**, $X = \{x_1, \dots, x_n\}$

A **finite domain** for each variable, $D = \{D_1, \dots, D_n\}$ with each $D_i = \{v_1, \dots, v_k\}$ containing values assignable to x_i.
Dom is a function that maps every variable in X to a set of objects of arbitrary type. $Dom(x) = D_x$

A **set of constraints** that restrict the values the variables can simultaneously take, C

Task: assign a value from the associated domain to each variable satisfying all the constraints. **NP-hard** in worst cases, but general heuristics exist and structures can be exploited for efficiency.

A **(partial) assignment** of values to a set of variables (**compound label**) is a set of pairs $A = \{\langle x_i, v_i \rangle, \dots\}$ with $v_i \in D_{x_i}$. A **complete assignment** is an assignment to all the variables of the problem. Can be projected to a smaller partial assignment by restricting the variables to a subset (projection, with the following notation: $\pi_{x_1,\dots,x_k} A$, with π being the projection operator of relational algebra)
Each constant in C can be represented as a pair $\langle$ scope, rel$\rangle$: scope is a tuple of variables participating in the constraint, and rel is a relation that defines the allowable combinations of values for those variables, taken from the respective domains. The relation can be represented as: an explicit list of all tuples of values that satisfy the constraint (explicit relation), or an implicit relation (an object that supports two operations: testing if a tuple is a member of the relation and enumerating the members of the relation).
We also use $C_{x_1,\dots,x_k} =$ rel to denote a constraint with scope $= x_1, \dots, x_k$, so the constraint $C = \langle (x_1, \dots, x_k), \text{rel} \rangle$

CSP solution To solve a CSP problem seen as a search problem, we need to define a state space and the notion of a solution.

State: assignment of values to some or all the variables.
Partial if values are assigned only to some of the variables, complete if every variable is assigned.

Solution: a complete and consistent assignment.
An assignment is consistent if it satisfies all the constraints: $\mathit{Satisfies}(\{\langle x_1, v_1 \rangle, \dots, \langle x_k, v_k \rangle\}, C_{x_1,\dots,x_k})$ for any constraint in C

Problem characteristics

Number of solutions required (one or all)

Problem size (number of variables and constraints)

Type of variables and constraints

Structure of the constraint graph

Thightness of the problems (measured in terms of the solution tuples over the number of all distinct compund labels of all variables)

CSP solving techniques Problem reduction techniques/inference/constraint propagation: techninques fro transofrming CSP into a equivalento problem easier to solve or recognizable as insoluble.
Searching efficiently: heuristicks, intelligent backtracking. . .
Exploiting the structure of the problem: independent sub-problems, tree structured constraint, tere decomp, exploiting symmetry.

Constraint hypergraphs

Related concepts Problem reduction techniques, enforcing local consistency and constraint propagation/inference.

Problem reduction P_1 reduced to P_2 when P_1 is equivalent to P_2, domains of variables in P_2 are subsets of those in P_1 and the constraints in P_2 are at least as restrective as those in P_1.
These conditions guarantees that a solution in P_2 is also a solution in P_1.
Strategies

Local Consistency properties

Node/domain consistency

Arc Consistency Algorithm (AC-3) Maintains a queue of arcs to consider, initially all the arcs in CSP. An edge produces two arcs. AC-3 pops off an arc (x_i, x_j) from the queue and makes x_i arc-consistent with respect to x_j. If this step leaves D_i unchanged, the algorithm just moves on to the next arc. If D_i is made smaller, then we need to add to teh queue all arcs (x_k, x_j) where x_k is a neighbor of $x_i \neq x_j$. If D_i becomes empty, then we conclude that the CSP has no solution.
When there are no more arcs to consider, we have finished.

Generalized Arc Consistency GAC, extension of AC-3.

CSP as search: simplifications

Backtracking search algorithm

Heuristics

Choosing the next variable

MRV (minimnum remaining values): variable with fewest "legal" remaining values

Degree heuristic: variable involved in the largest number of constraints

Choosing value

Least constaint variable: rules out the fewest schoices

Interleaving search and inference One of the simplest form of inference progpagation is forward checking: efficient constraint propagation, weaker than other forms. Whenever X is assigned, FC process establishes arc consinstency of X for the arcs conencting neighbor nodes. For each unassigned Y connected to X, delete form Y domain any value inconsistent with the value assigned to X.

Constraint learning When the search is at a contradiction, we know that some subset of the conflict set is responsible. Constaint learning is the idea of finding a minimum set of variables from teh conflct set that causes the problem: the no-good set. We record the no-good set either by adding a new costraint to teh CSP or by keeping a seaorate cache of no-goods. This way we do not repeat the no-good state.

Local search Require a complete state formulation, keep in memory only current state to improve it iteratively and do not guarantee to find solution even if it exists (not complete).
Used when space too large and need to be very efficient, provide a solution but not importanto to produce solution path and know in advance that a solution exists.

Heuristic Local Search Inject heuristics in the selection of the variable and the value by the means of an evaluation function. **Iterative best improvement**: choose the successor that most improves the current state according to an evaluation function f

Stochastic Local Search Adds randomness, escapes local minima...

Variants

Most improving step

Two stage choice

Any conflict

CSP: Min-Conflict Heuristic