# Introduction

Aditya Challa

BITS Pilani
Pilani Campus

August 4, 2024


# Course: Artificial and Computational Intelligence
# Lecture No. 2

# RECAP

- Four things they mean when people say AI
  - Acting Humanly - Turing Test - Can a machine exhibit intelligent behavior indistinguishable from a human?
  - Thinking Humanly - Cognitive Science - How do humans think and can machines think like humans?
  - Thinking Rationally - Logic - Can machines think logically?
  - Acting Rationally - Rational Agents - Can machines act rationally?
- There are issues with all of these ideas!
- We focus on the Acting Rationally part - Rational Agents

# RECAP

- Other important points:
  - Philosophy, Mathematics, Economics, Neuroscience, Psychology, Control Theory, Computer Science, Linguistics - All contribute to AI. So, its a very interdisciplinary field.
  - The history of AI is a series of overexpectations and underdeliveries. But the field has matured a lot in the last 10 years to a *point of no return*!.

# RECAP

- The broad lessons from history:
  - Economics trumps always. Anything which is not feasible and does not reduce costs might not last. See `https://www.goldmansachs.com/intelligence/pages/gs-research/gen-ai-too-much-spend-too-little-benefit/report.pdf`
  - Humans have an amazing ability to adapt and accept - ChapGPT was a phenomenon in 2021, but after 3 years there are open source models Llama3.1 which can do the same thing.
  - Efficiency is important - Bayesian methods can theoretically solve every problem out there, but it's highly inefficient.
  - What you measure is what you get - Evaluation metrics and designing them are probably the most important value addition humans still make.

# Abstracting the world of agents



- Where do we start? – Abstracting the world of agents
  - The concept of abstraction is a powerful tool which underlies all of mathematics.
  - Philosophically it means this –
    - If we see 1 orange + 1 orange = 2 oranges, 1 apples + 1 apple = 2 apples . . .
    - We abstract this into a equation $1 + 1 = 2$!
    - So basically remove all unnecessary information and build the framework.
  - we will try and apply this concept to modelling rational agents! – Start with how we interact with the world, remove all unnecessary details!

# Abstracting the world of agents



- This is how I will go about this – All our actions has the following steps:
  - We use our 5 senses to get some "signal" from the world around us,
  - We process the signal using some complex rules within our brain
  - We do the required actions which are outputs of the complex rules
  - The environment reacts to our actions. . .
  - The cycle repeats, at something like billion times a second..
- A specific example of this is driving!

# Abstracting the world of agents



- And as most mathematicians do, we will reduce the scope of each of these to the extent that we can actually do it on a computer!
  - A signal is nothing but some vector of real numbers.
  - The complex process is the *billion* dollar question which we want to solve and find
  - The actions are again a vector of real numbers.
  - The model for environment is also a difficult one - but technically not so different from the model of complex processes of our mind.
- An interesting thing to note – environments are basically billions of agents reacting to your action!

# Abstracting the world of agents

- So, we end up with the following diagram:

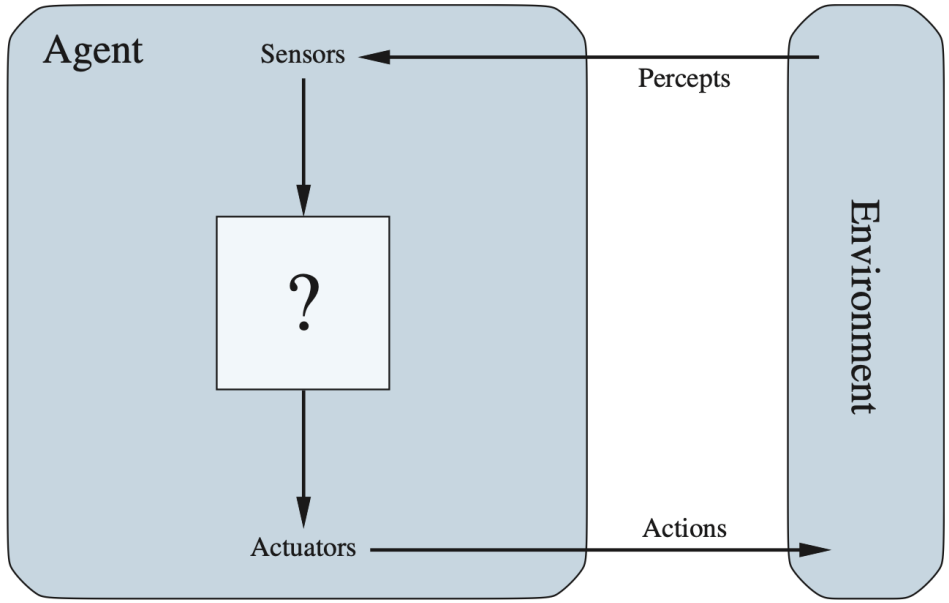


**Figure 2.1** Agents interact with environments through sensors and actuators.

# Abstracting the world of agents



- Some definitions.
  - An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
  - Percept refer to the content which the agent will receive
  - Percept sequence is the entire history of percept which the agent received
  - Agent function is the one which maps the percept sequence to a vector. Think of this as a BIG TABLE of inputs $\rightarrow$ outputs.

# Abstracting the world of agents

- Example – Vaccum World

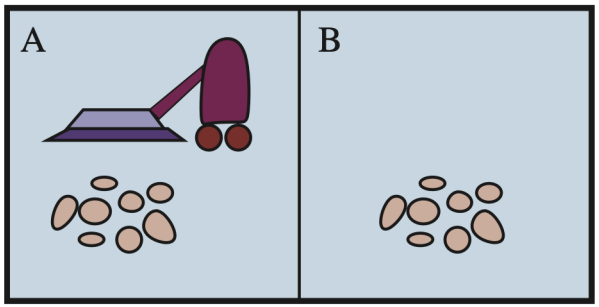


**Figure 2.2** A vacuum-cleaner world with just two locations. Each location can be clean or dirty, and the agent can move left or right and can clean the square that it occupies. Different versions of the vacuum world allow for different rules about what the agent can perceive, whether its actions always succeed, and so on.

# Abstracting the world of agents



- Example – Vaccum World

| Percept sequence | Action |
|---|---|
| $[A, Clean]$ | *Right* |
| $[A, Dirty]$ | *Suck* |
| $[B, Clean]$ | *Left* |
| $[B, Dirty]$ | *Suck* |
| $[A, Clean], [A, Clean]$ | *Right* |
| $[A, Clean], [A, Dirty]$ | *Suck* |
| ⋮ | ⋮ |
| $[A, Clean], [A, Clean], [A, Clean]$ | *Right* |
| $[A, Clean], [A, Clean], [A, Dirty]$ | *Suck* |
| ⋮ | ⋮ |

**Figure 2.3** Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2. The agent cleans the current square if it is dirty, otherwise it moves to the other square. Note that the table is of unbounded size unless there is a restriction on the length of possible percept sequences.

# Getting to Rationality?



- What is rationality and how do we measure it?
  - Consequentialism is an idea from philosophy which says – You measure the consequences of your actions, and that tells if an action is good/bad! (As opposed to intentions!)
  - There are obvious problems with this – But then this is easier to model and capture. So, we stick with this.
  - So, we basically design Performance Measures based on what we want to achieve in the environment. (Ref. thinking vs acting. Ref. place where human decisions still matter!)
  - Example - A vaccum cleaner in the vaccum world can be measured by amount of dirt it takes in.
  - **Important Note:** Designing a good performance measure is not obvious. One can get wildl different actions based on the performance measure (Ref. King Midas problem.)

# Getting to Rationality?

- Defining Rational Agent

  *For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

- So, for us whether something is rational or not depends on several things
  - Performance Measure
  - What the agent can perceive
  - And the ability of the agent to map a percept sequence to an action!
- We then to think *only* the last point as rationality, but that's not enough.

# Getting to Rationality?



- Rational agents maximize expected performance not actual performance!
  - This is a subtle but important difference.
  - The evidence for this comes from a long history of congnitive science experiments! - (Ref. Rat in a maze!, few others in the book)
  - Implicit here is *building an internal model of the world within the agent model* – Otherwise how would you know what the expected performance is?

# Creating the task environment



- As the famous saying goes – *All models are wrong but some models are useful.* So, we get to create the task environments and model rational agents accordingly. For this we need to specify:
  - Inputs : Signals from sensors
  - Outputs : Actions of actuators (think robots)
  - Performance Measure (discussed before)
  - Environment : A small closed system where the agent should do it's work. "small" because we cannot handle large systems yet!

# Creating the task environment

- Example - Autonomous Driving
  - Performance measure should be *some* combination of accuracy, fatalities, fuel efficiency, safety etc.
  - The environment can be taken to be highways (but this might not work with mountain regions)
  - Actuators would be - Brake/Accelerator (No need of clutch these days..)
  - Sensors – a 360 degree camera and maybe even LIDAR.

# Creating the task environment

| Agent Type | Performance Measure | Environment | Actuators | Sensors |
|---|---|---|---|---|
| Taxi driver | Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users | Roads, other traffic, police, pedestrians, customers, weather | Steering, accelerator, brake, signal, horn, display, speech | Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen |

**Figure 2.4** PEAS description of the task environment for an automated taxi driver.

# Creating the task environment

- So many decisions which the human should make –
  - Fully or Partially Observable? – Am I going to get traffic updates from google map or not??
  - Single vs Multi Agent – Should I consider the movement of other cars as following the same model as mine???
  - Competetive or Cooperative??
  - Deterministic or Non-Deterministic???
  - Episodic or Sequential??
  - Static vs Dynamic??
  - Discrete vs Continuous??
  - Known vs Unknown??

# Creating the task environment

- The hardest case in driving would be – *partially observable, multiagent, nondeterministic, sequential, dynamic, continuous, and unknown* or as is commonly known *driving on Bangalore roads!*

# Architectures of Agent Program



- What makes designing agents hard??
  - We know (thanks to mathematicians) that a simple table based system is sufficient to model *any* agent!

---

**function** TABLE-DRIVEN-AGENT(*percept*) **returns** an action
  **persistent**: *percepts*, a sequence, initially empty
    *table*, a table of actions, indexed by percept sequences, initially fully specified

  append *percept* to the end of *percepts*
  *action* ← LOOKUP(*percepts*, *table*)
  **return** *action*

**Figure 2.7** The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

# Architectures of Agent Program

- Let's do some back-of-envelope calculations on the Computational Requirement:
  - Each image has $10^6$ pixels, each pixel is 24 bits, we get 30 images per second – At the end of 1 hour the precept sequence can have

$$(24 * 10^6 * 30)^3 600 \approx 10^{32400}$$

    different possibilities. This would the length of the table!!!
  - For comparision, the number of atoms in the observable universe is $10^{80}$!
- So, the hardest part of modelling the agent is – *doing it efficiently!*

  *The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behavior from a smallish program rather than from a vast table.*

# Architectures of Agent Program

• So, we need better models than tables for modelling agents. There are different ways to approach this. The following are different categorization of agent models which are popular and widely used -

- Simple reflex agents
- Model based agents
- Goal based agents
- Utility based agents.
- Learning Based Agents.

# Architectures of Agent Program

- Simple reflex agents
  - Basically forget the history of percepts but focus on the "right now".
  - This is also called markov assumption.
  - While this looks too simple, these are actually quite powerful.
  - This also has a flexibility of using $T$ time steps in history as a state, where $T$ can be fixed based on compute power. (See DQN learning for atari games or even LLMs with their context windows!)

# Architectures of Agent Program

- Simple reflex agents

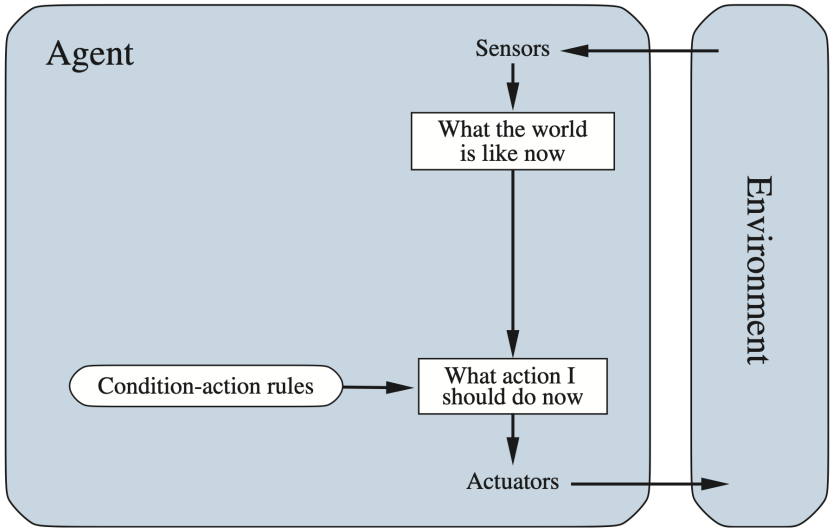


**Figure 2.9** Schematic diagram of a simple reflex agent. We use rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process.

**function** SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  **persistent**: *rules*, a set of condition–action rules

  *state* ← INTERPRET-INPUT(*percept*)
  *rule* ← RULE-MATCH(*state*, *rules*)
  *action* ← *rule*.ACTION
  **return** *action*

**Figure 2.10** A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

# Architectures of Agent Program

- Model based reflex agents
  - These agents build an “internal model” of the world.
  - They use the sensory inputs to update the state, and then accordingly act.
  - This is probably(???) what happens in our brains – You/Me can handle partial information exceptionally well.

# Architectures of Agent Program

- Model based reflex agents

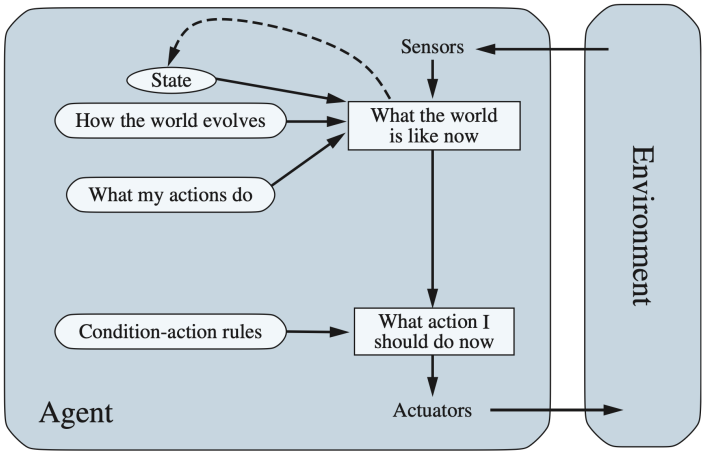


**Figure 2.11** A model-based reflex agent.

**function** MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
  **persistent**: *state*, the agent's current conception of the world state
    *transition_model*, a description of how the next state depends on the current state and action
    *sensor_model*, a description of how the current world state is reflected in the agent's percepts
    *rules*, a set of condition–action rules
    *action*, the most recent action, initially none

  *state* ← UPDATE-STATE(*state*, *action*, *percept*, *transition_model*, *sensor_model*)
  *rule* ← RULE-MATCH(*state*, *rules*)
  *action* ← *rule*.ACTION
  **return** *action*

**Figure 2.12** A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

# Architectures of Agent Program

- Goal Based Agents
  - Goal based agents essentially use “goals” as measure to dictate their actions.
  - This can be quite complex since there are usually short-vs-long term tradeoffs etc.
  - Search and Planning requires you to find a sequence of actions to maximize the performance (not just 1).

# Architectures of Agent Program

- Goal Based Agents

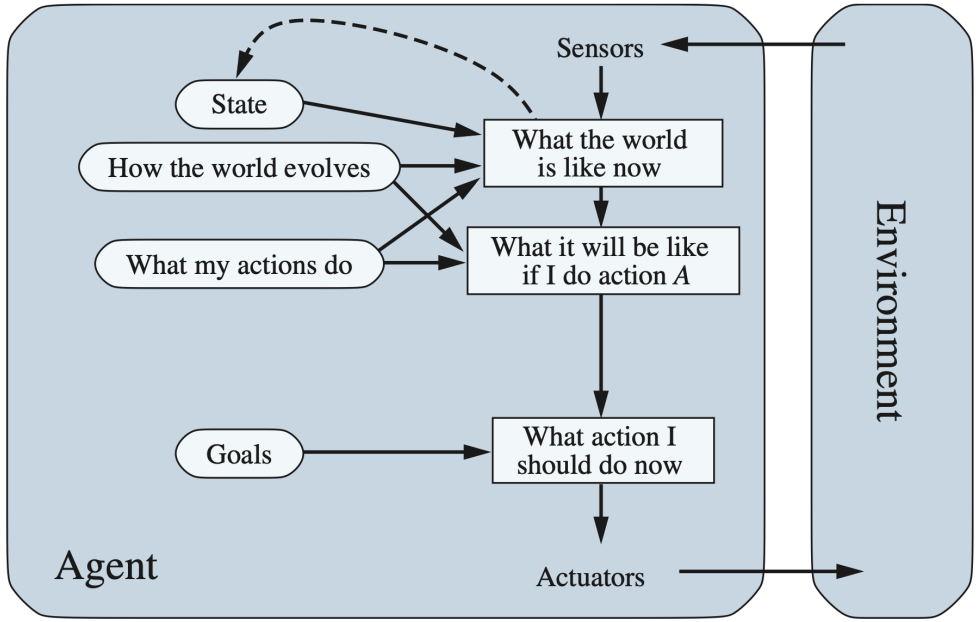


**Figure 2.13** A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

# Architectures of Agent Program

- Utility Based Agents
  - These agents try and have an internal model for the performance measure, and then take actions accordingly.
  - Distinguish this from an internal model of the world – Can you spot the difference??
  - These are called utility functions and are widely used in economics.
  - In cases where there is randomness, these agents try to maximize *expected utility*.

# Architectures of Agent Program

- Utility Based Agents

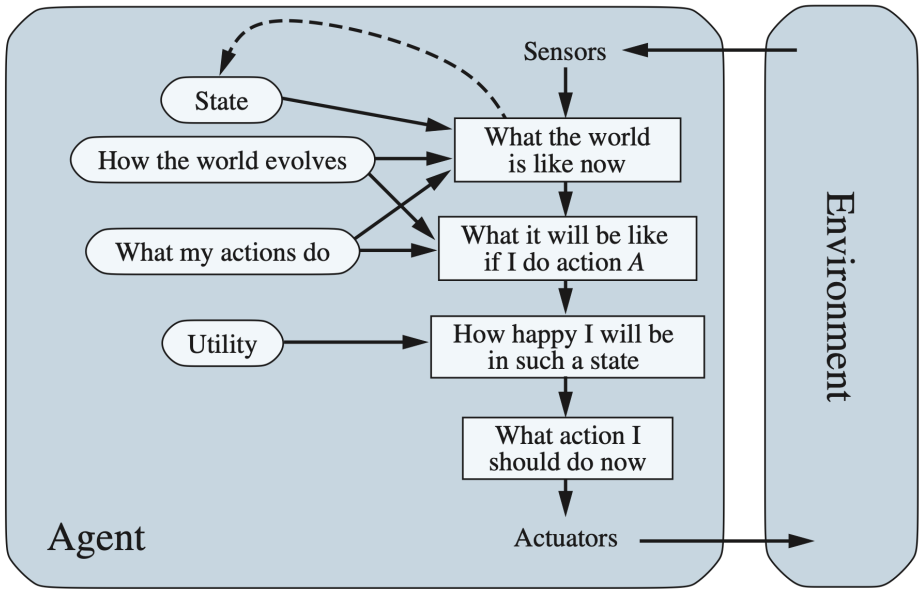


**Figure 2.14** A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

# Architectures of Agent Program

- Learning Agents
  - Another important aspect is whether the agent evolves or not? - in other words does it learn?
  - The learning module improves the performance module by using the critic module.
  - Th performance module decides the final actions by considering a list of possible actions.

# Architectures of Agent Program

- Utility Based Agents

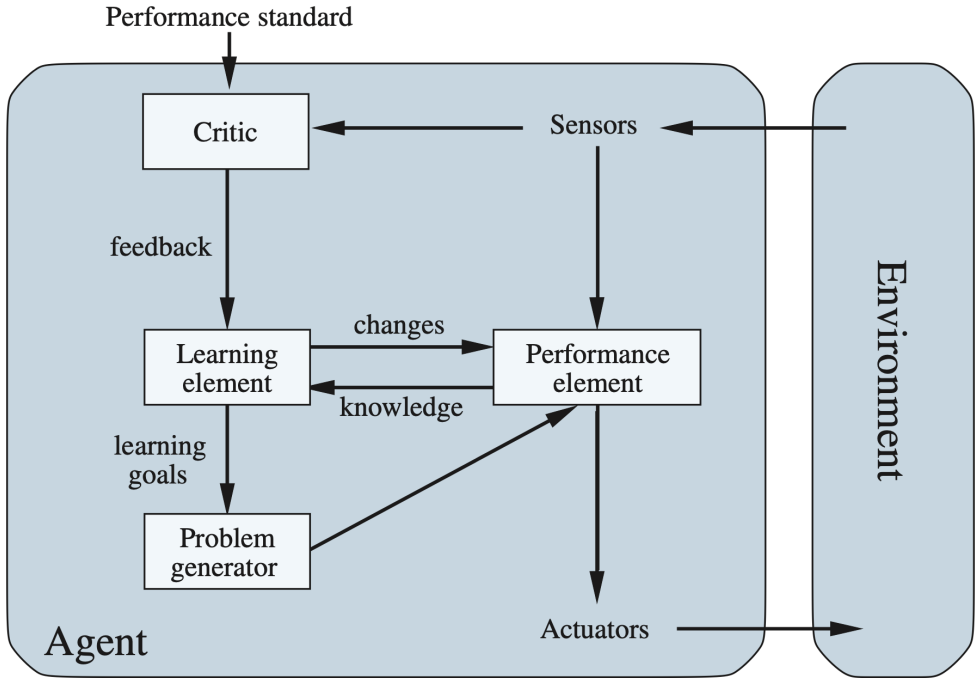


**Figure 2.15** A general learning agent. The "performance element" box represents what we have previously considered to be the whole agent program. Now, the "learning element" box gets to modify that program to improve its performance.

# Architectures of Agent Program



- Concept of Representations
  - Just as the way we abstract ideas, representations are ans abtraction of the states. There can be different ways we can treat representations –
  - **Atomic Representations** just treats each state as a vector and does not have any fine-grained information.
  - **Factored Representations** is like a (python) dictionary of variables/attributes mapped to a value – Think of indicators on a car dashboard.
  - **Structured Representations** are basically any kind of data structure which has a fixed meaning for it's entries. Think of a graph and how it is stored.

# Architectures of Agent Program



- Local vs Distributed Representations:
  - It's not always necessary that a concept is explicitly mapped to a vector.
  - **Local Representations** separate out the concepts
  - **Distributed Representations** do not use it.
  - The simple example is how words are represented? - Traditionally they use one-hot vectors (local), but Word2Vec uses distributed representations which captures the relationship across words.