Agent Architecture & Hierarchy

17 October 2020 16:05

Objectives:

- Learn what an agent is and agent functions
- Understand the different types of agent

Agents

By a hierarchic system, or hierarchy, we mean a system that is composed of interrelated subsystems, each of the latter being in turn hierarchic in structure until we reach some lowest level of elementary subsystem.

An **agent** is something that acts in an environment.

Agents interact with the environment with a body. An embodied agent has a physical body. A robot is an artificial purposive embodies agent.

Agents act in the world through their actuators, also called effectors.

Agent Systems

An agent system is made up of an agent and the environment in which it acts.
Agents in an agent system receive stimuli from their environment and carries out actions on the environment.

An agent is made up of a body and a controller. The controller receives percepts from the body and sends commands to the body.

The Agent Function

Agents are situated in time T that can be thought of as discrete, broken into sub-sections. T+1 is one of these intervals after T.

If time is dense, there is always another moment in time between any two other given moments – e.g. time is continuous.

Assume that T has a starting point, which we arbitrarily call 0.

Suppose P is the set of all possible percepts. A **percept trace**, or **percept stream**, is a function from T into P. It specifies what is observed at each time.

Suppose C is the set of all commands. A **command trace** is a function from T into C. It specifies the command for each time point.

A percept trace for an agent is thus the sequence of all past, present, and future precepts received by the controller. A command trace is the sequence of all past, present, and future commands issued by the controller. The commands can be a function of the history of percepts. This gives rise to the concept of a **transduction**, a function from percept traces into command traces.

A transduction is **causal** if, for all times t, the command at time t depends only on percepts up to and including time t. The causality restriction is needed because agents are situated in time; their command at any time cannot depend on future percepts.

A **controller** is an implementation of a causal transduction.

The **history** of an agent at time t is the percept trace of the agent for all times before or at time t and the command trace of the agent before time t.

Thus, a **causal transduction** maps the agent's history at time t into the command at time t. It can be seen as the most general specification of a controller.

Belief State

Although a causal transduction is a function of an agent's history, it cannot be directly implemented due to the fact that an agent does not have access to its entire history. It only has access to its current percepts and those that it has remembered.

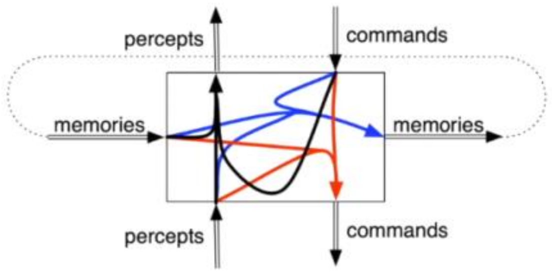
The memory or belief state of an agent at time t is all the information the agent has remembered from the previous times. An agent has access only to the history it has encoded in its belief state. Thus, the belief state encapsulates all the information about an agents history, that the agent can use for current and future commands. At any time, ag agent has access to its belief state and its current percepts.

The belief state can contain any information, subject to the agent's memory and processing limitations. This is a very general notion of belief.

The belief state function determines what we're going to remember - I.e. what will the next belief state look like
The command function uses the current belief state and the current percepts to decide on some action

If there are a finite number of possible belief states, the controller is called a **finite state controller** or a **finite state machine**. A **factored representation** is one in which the belief states, percepts, or commands are defined by features. If there are a finite number of features, and each feature can only have a finite number of possible values, the controller is a **factored finite state machine**. Richer controllers can be built using an unbounded number of values or an unbounded number of features. A controller that has an unbounded but countable number of states can compute anything that is computable by a Turing machine.

Functions Implemented in a Layer



- *memory function:* $remember(memory, percept, command)$
- *command function:* $do(memory, percept, command)$
- *percept function:* $higher\_percept(memory, percept, command)$

Hierarchical Control

There is much evidence that people have multiple qualitatively different levels. Kahneman [2011] presents evidence for two distinct levels: **System 1**, the lower level, is fast, automatic, parallel, intuitive, instinctive, emotional, and not open to introspection, and **System 2**, the higher level, is slow, deliberate, serial, open to introspection, and based on reasoning.

In a hierarchical controller there can be multiple channels – each representing a feature – between layers and between layers at different times.

There are three types of inputs to each layer at each time:

1. the features that come from the belief state, which are referred to as the remembered or previous values of these features
2. the features representing the precepts from the layer below in the hierarchy
3. the features representing the commands from the layer above in the hierarchy.

There are three types of outputs from each layer at each time:

- the higher-level percepts for the layer above
- the lower-level commands for the layer below
- the next values for the belief-state features.

The low-level controllers can run much faster, and react quickly
They also deliver a simple view of the world to the high level controllers.

Types of Agent

We can view agents as being specific by the agent function that maps a percept sequence to a sequence of actions

Ideal rational agents do whatever action is expected to maximise performance measure on basis of percept sequence and built-in knowledge.
So in principle there is an ideal mapping of percept sequences to actions
The simple approach to this is a lookup table, however this is doomed to failure be

The lookup table suggests a notion of an ideal mapping
This would be the rational agent function
We want to implement the rational agent function by implementing a function that approximates the ideal mapping as closely as possible

1. Simple reflex agent – condition action rules
2. Reflex agents with state – retains knowledge about the world
3. Goal-based agents – have a representation of desirable states
4. Utility based agents – ability to discern some useful measure between possible means of achieving state
5. Learning agents – able to modify their behaviour based on their performance, or in light of new information

Goal Based Agents

If we want to solve more sophisticated problems, we might need to look at a GBA
It maintains some representation of the state, which gets updated. It has some knowledge about how the world works, and a set of goals

To achieve goals, you need a sequence of actions. We need to keep track of how close we are to achieving our goal. GBA's are much more flexible.

Utility Based Agents

Similar to GBA, but rather than purely thinking about what action we should do in terms of the goal, we think about the benefit a particular action has.

Sometimes there are many ways of achieving a goal, and many actions you can take to get there.

Utility based agents have a utility function that allows us between which goals we should achieve, and alternative ways of achieving a goal

Learning Agents

Can be built upon any of the previous types – all 4 can fit into the performance element

The performance element takes some input, and chooses some action on the environment.
The learning agent has a critic, a problem generator and a learning element

The learning element uses information about how the agent operates and the particular algorithms, and changes how the performance element operates. It does this by getting information on how it's doing from the critic, and new things to try out from the problem generator.

- Useful when we don't know much about the environment and the agent has to be able to learn.
- Learning provides an agent with a degree of autonomy
- Learning results from interaction between the agent and the world.
- The critic uses a performance standard to tell the agent how it is doing. Performance standard should be a fixed measure, external to the agent.
- Problem generator – responsible for suggesting actions in pursuit of new and informative experiences