Agents and Environments

CSCI 2400 “Part 2”
Instructor: David Byrd

Disclaimer: Lecture notes can’t and won’t cover everything I say in class. You should attend class each day and use these for review or reinforcement.

2 Intelligent Agents

2.1 What’s an intelligent agent?

An *intelligent agent* is anything that can perceive the environment through *sensors* and act on it through *effectors* (also called *actuators*). An entity that can’t act is not an intelligent agent; it is just a sensor. An entity without sensing can’t be intelligent; it can only behave in a random or fixed (predetermined) manner.

It is reasonable to think of it as a *function* that maps *states* to *actions*. The sensors help it determine the state of the world (and perhaps itself). The effectors allow it to take action (change the world).

2.2 Terminology

Percepts are the inputs from an agent’s sensors. For example, Pacman’s “wall sensor” might emit a percept of `<S,W>` if there are currently walls to the south and west. Pacman’s “ear sensor” might emit a percept of `4` if the nearest ghost is four steps away in the maze.

The **percept sequence**, then, is the time series of all percepts remembered by the agent. For example, if Pacman has the above two sensors, then the percept sequence after three time steps ($t = 3$) might be:

```
t, wall percept, ear percept
1, <S,W>, 4
2, <S>, 3
3, <N,E>, 3
```

Note that the percept sequence *does not* include actions the agent may have taken. The agent must remember those on its own, if needed. In more complex problems, this can actually be quite challenging, as the agent’s actions do not necessarily produce the desired or expected outcome in all cases. Then, the agent only knows what it *attempted* to do, but must use its sensors to infer what actually happened!

In some problem configurations, the agent may not know what its actions "do" at all! (Think of a person sitting inside an enclosed room full of lights and buttons, with no instructions. The green and blue lights just came on. What should the person do? There is no way to tell, so the person tries pressing the red button.)

The **agent function** is the exact, exhaustive, mathematical mapping of percept sequence to action. For example, one entry in Pacman's agent function for $t = 2$ could be: `[ (<S,W>,4), (<S>,3) ]` $\rightarrow$ `E`, indicating that if at $t = 1$ there were walls to the south and west (only) and the nearest ghost was four steps away *and* at $t = 2$ there is a wall to the south (only) and the nearest ghost is three steps away, Pacman will go east.

The **agent program** is the as-implemented code or logic that drives the agent's behavior. It typically combines *many* mappings from the agent function into a much smaller set of "rules". (In other words, it attempts to identify useful *patterns* in the exhaustive mapping to obtain similar behavior without having to enumerate all possible percept sequences.)

An example agent program related to the above Pacman example:

```
if ears(t) < ears(t-1) then
    action is the reverse of the previous action
else
    action is a random non-wall direction that does not reverse
           the previous action
```

That is, if the nearest ghost is closer than it was at the last time step, Pacman reverses the last action (if it went West, it goes back East). Otherwise, it explores a random direction that is not where it came from and doesn't run into a wall. (Notice the logical flaw in these rules? What if Pacman has entered a dead end?)

These rules are not particularly clever, but at least it gets Pacman moving around the maze, and it still works 100 time steps into the game ($t = 100$). With the actual agent function, which must map all possible *percept sequences* to actions, at $t = 100$ we will have a table containing $(2^4 \cdot 10)^{100}$ mappings. (The wall sensor emits four boolean variables each time step, which gives 16 possible different values. We have assumed the ear sensor returns values in the range $[0, 9]$, having a maximum hearing range of nine steps away and emitting zero if it hears nothing in range.) This is already 160^{100} or about 2.58×10^{220} unique percept sequences! In contrast, our agent program is a single decision leading to two simple action rules.

2.3 Rational Agents

We define a *rational agent* as one that selects the *expected optimal action* for every sequence of environmental states according to some performance metric. A *performance metric* is a function that assigns a numerical score to the outcome or result of an agent's behavior over time, typically representing its "goodness" (large positive numbers) or "badness" (small or negative numbers).

In Machine Learning, performance is usually thought of as a *loss function* (a penalty "score" to be minimized, a measurement of error) or an *objective function* (the evaluation of performance you wish to maximize or minimize depending on the specific problem).

A few examples of simple performance metrics: +100 points for not dying, +1 point per dot eaten by Pacman, -1 per move without winning. Often many rewards and penalties will be combined into an overall performance metric to encourage the agent towards a desirable behavior. (Clear the maze as quickly as possible, but without dying.)

It is important to realize that a poorly designed performance metric can thwart an otherwise capable agent! Think of trying to train a dog, but accidentally rewarding it for the wrong behavior. It may well correctly learn to do the wrong thing.

2.4 Example Problem: Annoying Cat World

Here's a different example (from what we did in class) of a simple problem for an intelligent agent to solve.

2.4.1 What do we know about cats?

(Disclaimer: I lived many years with, and generally like, cats.) They like to be alone. They like to annoy people. They are lazy. Can we design a useful agent that fits this description?

2.4.2 Problem Environment

Assume the cat lives in a house with two rooms, A and B. At any given time, the cat occupies exactly one room. Each room may, or may not, also be occupied by humans. Room B is to the right of Room A. A group of humans annoyed by a cat will leave the house and will not return (within the 1,000 time step duration of the problem).

2.4.3 Problem Definition / Agent Parameters

- **Sensors:** room sensor (is the cat in A or B?), human sensor (do any humans occupy the current room? `T/F`)
- **Actions:** `R` (move right), L (move left), A (annoy people), N (do nothing)
- **Prior Information:** map of house
- **Goal:** maximize points earned over 1,000 time steps
- **Performance Metric:** +1 point per empty room per time step (i.e. house's average "emptiness" over time)

2.4.4 Naive Implementation

A naive implementation may simply complete the *agent function* (i.e. table mapping all possible percept sequences to actions) up to the maximum time ($t = 1000$) specified in the problem. At $t = 1$ there will be four entries: $(A, empty) \rightarrow R, (A, humans) \rightarrow A, (B, empty) \rightarrow L, (B, humans) \rightarrow A$. For example, if the cat starts in room A and it is empty, the cat will move right to room B. If the cat starts in room A and there are humans, it will annoy them into leaving the house.

For $t = 1$, this seems perfectly reasonable. There are four possible percept sequences of length 1, and so four table entries. However, at $t = 2$ there are 16 possible percept sequences, at $t = 3$ there are 64, and in general at $t = n$ there are 4^n possible percept sequences (i.e. required number of table entries). Our simple problem therefore requires a lookup table of size 4^{1000} entries, which is absurdly large.

2.4.5 Simple Agent Program

We can, however, construct an incredibly simple agent program to solve the problem, without using any "artificial intelligence" techniques at all!

```
if room contains humans, annoy them
else
    if we have not visited the other room, move to it
    else do nothing
```

This seems suitably cat-like. The cat empties out the house as quickly as possible, then lies down for a long nap while collecting two points per turn. With a few lines of code, we have replaced a 4^{1000} entry table.

2.4.6 Adding Complexity

The problem can be made more interesting in many ways.

What if people (at random, from the cat's perspective) can re-enter the house after being annoyed? Now at any given time, it only *knows* if its current room is empty. It does not know if there are people in the other room. Thus it needs to move and check the other room to ensure it is receiving the full two points per turn. Right now, the optimal way to do this is to *never* rest and do nothing, but rather to *always* move if the current room is empty. This solution does not, however, represent our idea of cat-ness very well.

Let's address the above issue by altering the problem. Our performance metric will now also award -1 points (a penalty) every time the cat moves, to represent the cat's desire to be lazy. Now how often should it check the other room? Too often, and its score will be eaten up by movement penalties. Too rarely, and it may miss out on points when the other room becomes occupied by humans again. We may wish to identify some pattern of re-entry that would help determine how often to check the other room (*e.g.* how long do annoyed humans tend to stay out of the house?).

What if the cat does not have a map of the house? It must explore, at least in the beginning. This produces the exploration-exploitation problem (which we will return to much later). Exploring costs the cat points (movement penalty). *Not* exploring could cost the cat even more points (all rooms it does not know about might be occupied by humans).

What if the cat's sensors are not reliable? What if, sometimes, it believes its room is empty, but it is actually occupied, or it believes it is in room A but is really in room B? What if the cat's effectors are not reliable? What if it sometimes fails to move when it tries, or it fails to drive the humans out of a room? What if both sensors and effectors are not reliable? Now the cat has a real problem, as *not only* might an "annoy" action fail to remove the humans from the room, but *also* when it next looks around, it might falsely appear to have succeeded!

It is quite common for an agent's sensors and/or effectors to not be fully reliable. This is one of the reasons we define a *rational* agent to be one that maximizes *expected* reward, not *actual* reward. Frequently it is not possible for an agent to achieve the theoretical maximum reward (from an omniscient perspective) due to these limitations.

2.5 Types of Intelligent Agents

You may think of intelligent agents as falling into a rough hierarchy of complexity and capability. In increasing order of complexity:

- *Reflex agents* simply map current percepts directly to actions. They have no memory, no internal state, no ability to "think" or model the world in any way. Many bugs (e.g. cockroaches moving directly away from the direction of brightest light) appear to behave this way. In the right environment, such agents can still be quite successful.

- *Model-based agents* contain an internal model of the world (an estimated state of the world). When percepts arrive, the agent updates its internal representation based on the new information, then selects an action based on its internal model. An agent that explores a maze and remembers where it has been is one example of a model-based agent.

- *Goal-based agents* add a goal (desired state of the world) and some capability to anticipate the results of an action relative to reaching the goal state. Thus they add the capability to predict action consequences and create a "plan" to reach the goal state. An agent that tries to reach a specific location in a maze is a good example.

- *Utility agents* further add a "score" function. These agents try to not only reach a goal state, but with a maximum (or minimum) score while doing so. An agent that tries to reach a specific location in a maze by the shortest path, or while collecting the most treasure, could be a utility agent.

- *Learning agents* improve their own performance at some task by exposure to new data. This is where the entire sub-field of *machine learning* lives: agents that directly learn from experience. These are *inductive* agents that infer the "rules" of the world from (a lot of) data.

3 Environments

It is wasteful (time, materials, expense) to design an overly-complicated agent to solve a simple problem. Thus it is helpful to classify our problem environment in certain ways that may help us determine what type of agent is required to successfully navigate the environment.

Some common properties we use to think about agent environments:

- A *fully observable* environment allows the agent to see or know everything relevant to the problem at any time, while in a *partially observable* one the agent receives only some information due to noisy or limited sensors (relative to the world).

- A *deterministic* environment only changes due to the agent's own actions and it changes in exactly the way the agent intends/expects. In a *stochastic* environment, the world changes due to other causes or in unpredictable ways (randomness, uncertain sensors or effectors). Note that a partially observable environment is always stochastic *from the agent's perspective* because there are things it can not know or predict.

- A *static* environment updates only after the agent selects an action to perform (i.e. turn-based) or on a set schedule that is slow enough for the agent to adequately make decisions (i.e. timed turns). A *dynamic* environment changes constantly, faster than the agent can "think". In a dynamic environment, the agent is therefore always acting (by the time it does act) on outdated information.

- A *discrete* world is broken into a finite number of "chunks". That is, the world can be in a finite number of states, and they could be enumerated (in theory, if you had enough time). A *continuous* world contains infinite chunks (think Minecraft, where there are chunks of a fixed size, but as you walk in one direction it generates new chunks without limit) or contains infinite gradations (the world size is limited, but you can move at an infinitely fine resolution to $(x, y) = (4.10263, -2.0043)$, etc).

- A *single agent* environment is occupied by only a single source of intentional change (the environment can still be stochastic due to unintentional or random changes). A *multiple agent* environment contains multiple such sources, which may be competing or collaborating.

- An *episodic* environment does not contain relevant historical information. That is, for the problem being solved in the environment, all required information can be currently observed. A *sequential* environment requires a successful agent to possess memory of prior states and observations, because that prior history will affect its current and future actions.

- The agent may or may not *know* all of the above characteristics. It is especially difficult to design a useful agent for an *unknown* world, for which it is not known if the world is partially or fully observable, deterministic or stochastic, etc.

The simplest overall environment we might consider would be one that meets the first option in each bullet point above (fully observable, deterministic, static, discrete, single agent, episodic). A quite simple agent might perform very well in such a "toy" world. The most complex overall environment would be one that meets the second option of every bullet point (partially observable, stochastic, dynamic, continuous, multi-agent, sequential). We would typically need to devise a very sophisticated agent for such a problem, as our environment begins to approximate the "real world".

Most of the rest of the class is effectively climbing up this "ladder" of increasingly-challenging environments by developing increasingly-sophisticated agents.

Next up: Search.

Acknowledgements and thanks to Professors Mark Riedl and Jim Rehg of the Georgia Tech School of Interactive Computing.