

Optimal Agents

Nick Hay

27th September 2005

Motivation

- **Artificial Intelligence (AI)** is the field inspired by the successes of the human brain.
- The problem of AI has not yet been well defined. We lack a rigorous (i.e. mathematical) definition which only AIs satisfy. (Contrast with computability.)
- Well defining the AI problem is solving it with access to unbounded computational power. If we cannot solve AI without constraints, we cannot solve it at all.
- There have been efforts towards a rigorous definition of AI (e.g. Marcus Hutter's AIXI), but they are the first steps not a complete solution.

Overview

This talk will:

- 1 Describe our theoretical model, explaining why it is natural and making explicit the assumptions involved.
- 2 Describe the special case of reward-based agents (reinforcement learning; hedonism), including Marcus Hutter's AIXI. We argue reward-based agents are not what we want.
- 3 Outline future research directions. (Very much work in progress!)

Feel free to interrupt with questions or comments.

References

The ideas in this talk are particularly inspired by:

- Hutter, 2004. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin.
- Russell & Norvig, 2003. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey.
- Sutton & Barto, 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge.

Outline

- 1 The Optimal Agent
 - What We Want
 - Choosing Agents
 - Explicit Form
- 2 Application & Evaluation
 - Examples
 - Reward-Based Agents; AIXI
 - Further work

Outline

- 1 The Optimal Agent
 - What We Want
 - Choosing Agents
 - Explicit Form
- 2 Application & Evaluation
 - Examples
 - Reward-Based Agents; AIXI
 - Further work

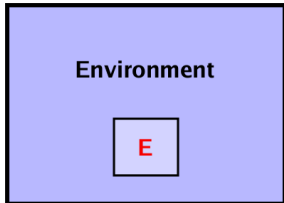
AI Is Making Things Achieve What We Want

- We derive our model from the intuitive idea that AI is “making things that achieve what we want”. For example, automatically running space craft, playing computer games, solving world hunger.
- The “Intelligence” in AI is useful only when it helps achieve what we want.
- We will informally present a formalisation of this definition in 4 parts.

What We Want

Influencing a variable

- We want reality to be a certain way. We formalise this as wanting **variables** in the environment to have particular **values**.



- Let E (for effect) be a variable having a value in the set E .
- Examples for E :
 - For an air conditioner: a room's temperature throughout time.
 - For a batch computation: the output.
 - In general: the state of the universe for the next T years.

What We Want

Utility functions

- A **utility function** evaluates the utility of each possible alternative value $e \in E$:

$$U: E \rightarrow \mathbb{R}$$

- Utility functions order certain effects, but also allow us to weigh up trade offs under uncertainty.
- Given a probability distribution $P(e)$ over E (an “uncertain effect”) define the **expected utility** as:

$$E[U] = \sum_e U(e)P(e)$$

- Probability distributions are functions $P: E \rightarrow [0, 1]$ such that $\sum_e P(e) = 1$.

What We Want

Toy example

- Toy example: utility function for a pet finding robot.

e	$U(e)$	$P(e a_1)$	$P(e a_2)$
Turtle	10	0.60	0
Cat	5	0	0.80
Nothing	0	0	0.15
Spider	-10	0.40	0.05

- The expected utilities of each alternative, a_1 and a_2 :

$$E[U|a_1] = \sum_e U(e)P(e|a_1) = 2$$

$$E[U|a_2] = \sum_e U(e)P(e|a_2) = 3.5$$

- a_2 has highest $E[U]$ even though $U(\text{Turtle}) > U(\text{Cat})$.

Achieving What We Want

Maximising expected utility

- Achieving what we want is maximising the expected utility of a variable E .
- Where we have a set of choices C , we select a choice $c \in C$ which maximises its expected utility:

$$E[U|c] = \sum_{e \in E} U(e)P(e|c)$$

where $P(e|c)$ is the probability that effect e occurs given a fixed choice c .

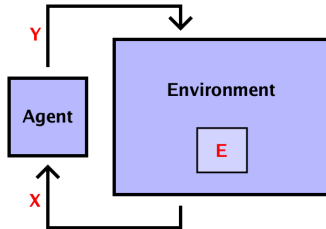
- Humans don't work like this, so what we want need not be maximising an expected utility. But it's a common simplification.

Outline

- 1 The Optimal Agent
 - What We Want
 - **Choosing Agents**
 - Explicit Form

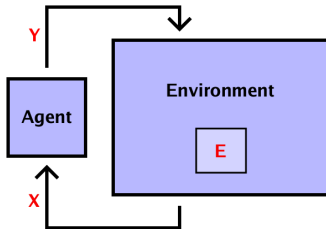
- 2 Application & Evaluation
 - Examples
 - Reward-Based Agents; AIXI
 - Further work

Things



- We follow a recent trend which focuses AI around the design of **agents**.
- For our purposes, an agent is a system that interacts with its environment, but is isolated apart from an input/output channel. Let X be the set of inputs, Y the set of outputs.

Things



- An agent a is a function mapping a history of inputs $x_{<i} \in X^{<N}$ to an output $y_i \in Y$:

$$a: X^{<N} \rightarrow Y$$

- Notation: if $x = x_1 x_2 \dots x_n$ is a sequence, $x_{<i} = x_1 \dots x_{i-1}$ and $x_{i:j} = x_i \dots x_j$. $X^{<N} = \bigcup_{i=0}^{N-1} X^i$.

Making Things That Achieve What We Want

Expected utility of an agent

- The expected utility of an agent a is given by:

$$E[U|a] = \sum_e U(e)P(e|a)$$

$$P(e|a) = \sum_{y^{x_{1:N}}} P(e|yx_{1:N})P(y^{x_{1:N}}|a)$$

$$P(y^{x_{1:i}}|a) = P(y^{x_{1:i-1}}|a)[y_i = a(x_{1:i-1})]P(x_i|y^{x_{1:i-1}}y_i)$$

$$\text{where } [X] = \begin{cases} 1 & \text{if } X \text{ is true} \\ 0 & \text{if } X \text{ is false} \end{cases}$$

- The important part is this depends on three things:

$U(e)$: What we want.

$P(x_i|y^{x_{1:i-1}}y_i)$: How we expect the environment to react.

$P(e|y^{x_{1:N}})$: Ability to infer the value of E from the complete IO history.

Making Things That Achieve What We Want

Choosing optimal agents

- Finally, an **optimal agent** a^* is one with maximal expected utility:

$$E[U|a^*] = \max_a E[U|a]$$

- Making things that achieve what we want is choosing agents with maximal expected utility.
- The equations for expected utility of an agent can be derived from the definition of “agent” (i.e. its isolation, and the existence of a fixed agent function).

Outline

- 1 The Optimal Agent
 - What We Want
 - Choosing Agents
 - **Explicit Form**

- 2 Application & Evaluation
 - Examples
 - Reward-Based Agents; AIXI
 - Further work

Finding An Explicit Solution

- The equation

$$E[U|a^*] = \max_a E[U|a]$$

implicitly defines the optimal agents a^* .

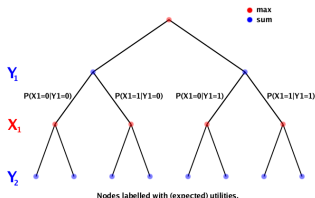
- It turns out there is an explicit characterisation, which explains how each action is taken.
- In effect, this agent plans its entire life before its first action, with the plan taking into account all possible input sequences.
- One can prove that every optimal agent is of this form. There are different optimal agents exactly when there are actions with equal expected utility.

The Optimal Agent

- The optimal agent selects actions by evaluating an expectimax tree over all possible futures:
 - Leaves labelled by $E[U|yx_{1:N}] = \sum_e U(e)P(e|yx_{1:N})$.
 - Nodes calculated by alternately maximising and taking expectation:

$$E[U|yx_{<i}] = \max_{y_i} E[U|yx_{<i}y_i]$$

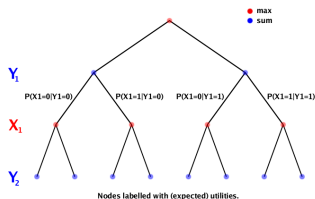
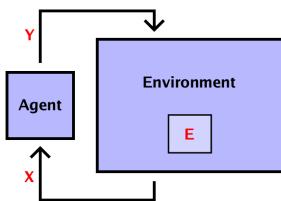
$$E[U|yx_{<i}y_i] = \sum_{x_j} P(x_j|yx_{<i}y_i)E[U|yx_{1:j}]$$



The Optimal Agent

Recap

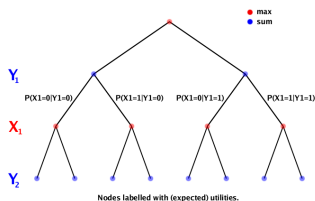
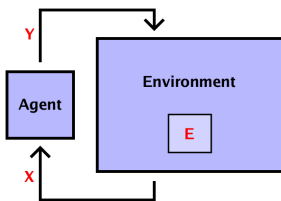
- We assume our goal in life is to maximise the expected utility of some variable within reality E .
- We achieve this by choosing the best possible agent a , i.e. one maximising $E[U|a]$.
- Using properties of agents, we derive the solution: an optimal agent is equivalent to one which evaluates a particular (huge) expectimax tree.



Free Variables Of The Model

We've described a family of models. To define particular instances we need to further specify:

- X, Y, E : the set of inputs, outputs, and effects.
- $U: E \rightarrow \mathbb{R}$: utility function describing what effects we want.
- $P(x_i | yx_{1:i-1}y_i)$: inferring which environmental input will follow a given past IO history.
- $P(e | yx_{1:N})$: inferring the effect of a complete IO history. This depends on the definition of E .



Outline

- 1 The Optimal Agent
 - What We Want
 - Choosing Agents
 - Explicit Form
- 2 Application & Evaluation
 - **Examples**
 - Reward-Based Agents; AIXI
 - Further work

Example: Thermostat

- Consider an agent controlling the temperature of the room. Here, $E = T^N$, where T is the set of temperatures of the room. With γ the set temperature, take the utility:

$$U(e) = U(t_{1:N}) = \sum_{i=1}^N -(t_i - \gamma)^2$$

- Let $X = T$, the temperature reading, and $Y = \{0, 1\}$ be the state of the heater (off or on).
- $P(e|yx_N) = P(t_{1:N}|yx_{1:N}) = [t_{1:N} = x_{1:N}]$, assuming perfect temperature reading.
- $P(x_i|yx_{<i}y_i)$ predicts the temperature of the room, given past temperature and heater state.

Stateful Environments

If we assume the environment has a state, and we care about something within it, we get a reformulation:

- $E = S$ the state of the environment over (at least) N time steps.

$$P(s|yx_{1:N}) = \frac{P(x_{1:N}|s)P(s|y_{1:N})}{\sum_s P(x_{1:N}|s)P(s|y_{1:N})}$$

$$P(x_i|yx_{<i}) = \frac{\sum_s P(x_{1:i}|s)P(s|y_{<i})}{\sum_s P(x_{<i}|s)P(s|y_{<i})}$$

- $P(s|y_{<i})$: partially computes the dynamics of the environment, given the action sequence.
- $P(x_{1:i}|s)$: extracts the input stream from a state history.
- $U(s)$: the utility of a state history.

Example: Game Of Life

Maximising gliders

- S is the set of all sequences of N board configurations. X and Y are the states of subregions of the board.
- $P(s|y_{<i})$: computes the probability of a sequence of board configurations.
- $P(x_{1:i}|s)$: extracts the input stream.
- $U(s)$: counts the number of gliders in s .



The optimal agent takes actions to maximise the total number of gliders.

Outline

- 1 The Optimal Agent
 - What We Want
 - Choosing Agents
 - Explicit Form
- 2 Application & Evaluation
 - Examples
 - **Reward-Based Agents; AIXI**
 - Further work

Reward-Based Agents

- Reinforcement learning makes a particular choice for the variable E and utility U . It describes “hedonistic” agents which try to maximise the total reward they receive.
- The input is divided into S , the state of the environment (more generally, an arbitrary input), and R , the reward signal.

$$X = S \times R$$

- The agent is trying to manipulate its sequence of rewards, maximising the expected sum. So, $E = R^N$ and:

$$U(e) = U(r_1 \dots r_N) = \sum_{i=1}^N r_i$$

Reward-Based Agents

- Since the optimised variable E is part of the IO history, inferring it is trivial:

$$P(e|yx_{1:N}) = P(r_{1:N}|yx_{1:N}) = [\forall i r_i = p_R(x_i)]$$

where p_R projects $X = S \times R$ onto its R component.

- Since we don't care about internal details of the environment, we can treat it as a blackbox: a function from agent outputs to agent inputs.
- Choosing $P(x_i|yx_{<i}y_i)$ is simpler than choosing $P(e|yx_{1:N})$: the former is a prediction that is tested at every time step, so one can "learn from error". In general the latter cannot be so tested as we do not have direct access to E . (Except in the special case when we do have direct access to it, as in reinforcement learning.)

Omniscient agents: AI_μ

- Now we have a single parameter for our model:

$$P(x_i | y_{x_{<i}y_i})$$

- If we knew the exact environment q , we could make a perfect predictor:

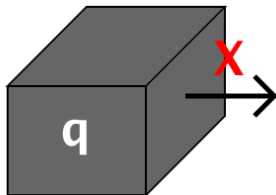
$$P(x_i | y_{x_{<i}y_i}) = [q(y_{1:i}) = x_i]$$

- This assumes determinism, but we can weaken this to a probability distribution μ if the environment is fundamentally probabilistic (e.g. quantum randomness?).
- Given complete knowledge of the environment, $P(e | y_{x_{1:N}})$ doesn't follow as easily. X and Y have fixed definitions in terms of the interface, but E can be *anything*.

Solomonoff Induction

- However, in general we won't know the exact distribution μ the environment follows. Here, the problem is predicting the output of a black box.
- Solomonoff induction solves this for a computable black box without input. Fixing a prefix-free universal Turing machine U , we have:

$$P(x_i|x_{<i}) = \frac{\sum_{q:q \text{ outputs } x_{<i}x_i\dots} 2^{-|q|}}{\sum_{q:q \text{ outputs } x_{<i}\dots} 2^{-|q|}}$$

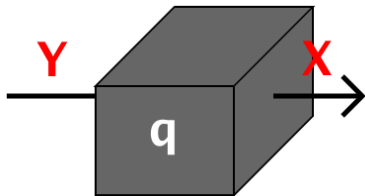


The environment q is modelled as a binary string w passed to U , so q outputs $U(w)$ and $|q| = |w|$. This formalises Occam's razor.

AIXI

- Marcus Hutter generalised Solomonoff induction to the case of agents, forming his **AIXI** model.
- Now the environment is a computable function q mapping agent outputs to agent inputs:

$$P(x_i | y_{<i} x_{<i} y_i) = \frac{\sum_{q: q(y_{<i} y_i) = x_{<i} x_i} 2^{-|q|}}{\sum_{q: q(y_{<i}) = x_{<i}} 2^{-|q|}}$$



The environment q is modelled as a binary prefix w passed to U , so $q(y) = U(wy)$ and $|q| = |w|$.

Problems With Reward-Based Agents

- Reward-based agents are a special case of this framework where reward signals are maximised. Although it is non-trivial, the general case introduces new research issues (e.g. choosing E , U , and $P(e|yx_{1:N})$).
- Our criterion for success isn't "maximal reward received by this agent". AI_{μ} isn't optimal by our standards.
- To achieve things we actually want the environment must have robust structure binding reward signals to things we care about e.g. a human controlled reward generation process. If this is imperfect, the agent will not in general achieve what we want (Genie problem: you get what you wish for).

Transferring Utility Function

- One problem with our framework is the complete utility function of humans (assuming we have one) is unknown. We don't know what we want well enough to write it into a program.
- Perhaps we can design agents which learn our utility function. It would be instructive to derive an optimal solution to this problem rigorously and without anthropomorphism.
- Reinforcement learning is seen as teaching what we want through reward and punishment, by analogy with human children. It is more like controlling an drug addicts' supply. (Really, it is like neither.)
- This problem isn't trivial to formalise, but could involve an environmental agent with a utility our agent has to learn and implement.

Outline

- 1 The Optimal Agent
 - What We Want
 - Choosing Agents
 - Explicit Form
- 2 Application & Evaluation
 - Examples
 - Reward-Based Agents; AIXI
 - **Further work**

Further work

Applying this framework:

- Understanding how it optimally solves various problems.
- Implementing other approaches to AI within this framework.
- Formalising and solving the problem of transferring a utility function.

Extending the framework:

- Replacing the decision theory (i.e. choosing the agent with maximal expected utility). The theory of what we want.
- Replacing the agent model. The model of what we create.
- Considering computable/tractable implementations.