An Approximation of the Universal Intelligence Measure

Shane Legg and Joel Veness

November 2011

Informal Definition of Intelligence

After surveying 70 informal definitions of intelligence, Legg and Hutter (2007) argue that the essence of many of these definitions is the following:

> *"intelligence measures an agent's ability to achieve goals in a wide range of environments"*

This is then formalised within the reinforcement learning framework as follows...

Universal Intelligence Measure

$$\Upsilon(\pi) := \sum_{\mu \in E} 2^{-K(\mu)} V_{\mu}^{\pi}$$

Where:

- π is the agent, expressed as a conditional measure
- E set of all computable reward bounded environments
- μ is a specific environment
- $K(\cdot)$ is the Kolmogorov complexity function
- $V_{\mu}^{\pi} := \mathbb{E}(\sum_{i=1}^{\infty} R_i)$ is the expected sum of future rewards when agent π interacts with environment μ.

Strengths of the Universal Intelligence Measure

- A formally defined measure of intelligence!
- Based on many standard informal definitions.
- A continuous measure of intelligence which is thus more informative of progress than the pass/fail Turing test.
- The most intelligent agent, according to this definition, is Hutter's AIXI which has strong optimality properties.
- Non-anthropocentric as it is based on the fundamentals of mathematics and computation rather than human imitation.

From Theory ... to Practice

Universal Intelligence was designed to be a *theoretical definition*. Indeed, it's not even computable!

To convert it to a practical test of intelligence requires some work:

- How do we approximate environment complexity?
- What reference machine will we use?
- How will we deal with non-halting programs?
- How can we efficiently sample?

Our aim is to address these issues in a *practical way*, while keeping to the spirit of the Universal Intelligence Measure.

$\Rightarrow$ We call the resulting measure Algorithmic IQ, or AIQ.

Solomonoff Sampling

Solomonoff defined his *universal prior distribution* in two ways:

- In terms of the Kolmogorov complexity of each measure
- In terms of the output of arbitrary programs

In the latter, the prior probability of a sequence beginning with x is:

$$M_{\mathcal{U}}(x) := \sum_{p\,:\,\mathcal{U}(p)=x*} 2^{-\ell(p)}$$

where $\mathcal{U}(p) = x*$ means that the universal Turing machine $\mathcal{U}$ computes a sequence that begins with x when it runs program p.

$\Rightarrow$ No (incomputable) Kolmogorov complexity function
$\Rightarrow$ Suggests approximation by sampling random programs

A Monte Carlo Estimate of Universal Intelligence

We can then approximate the Universal Intelligence of an agent π with a simple Monte Carlo sample over random programs $p_1, p_2, \ldots$

$$\hat{\Upsilon}(\pi) := \frac{1}{N} \sum_{i=1}^{N} \hat{V}_{p_i}^{\pi}$$

where $\hat{V}_{p_i}^{\pi}$ is now the empirical total reward returned from a single trial of environment $\mathcal{U}(p_i)$ interacting with agent π.

$\Rightarrow$ Multiple *programs* can compute the same environment.

$\Rightarrow$ The weighting by $2^{-\ell(p_i)}$ is no longer needed as the probability of a program being sampled decreases with additional length.

To make this work we need...

- a reference machine (programming language) that is easy to run and easy to sample from.
- programs to compute reinforcement learning environments, not just sequences.
- to deal with non-hating programs.
- to be efficient enough to get useful results!

We will now deal with each of these issues in turn.

For the first issue, we chose the Brain F$%k language, known as BF, that has only 8 simple instructions and yet is Turing complete (assuming an infinite work tape, of course).

BF Reference Machine

- *input tape*: one-way read-only
- *work tape*: bidirectional read-write
- *output tape*: one-way write-only

BF symbolic instructions listed below, with their C equivalents:

>	move pointer right	`p++;`
<	move pointer left	`p--;`
+	increment cell	`*p++;`
-	decrement cell	`*p--;`
.	write output	`putchar(*p);`
,	read input	`*p = getchar();`
[	if cell is non-zero, start loop	`while(*p) {`
]	return to start of loop	`}`

BF Reference Machine

- We let the work tape have 10,000 cells.
- To deal with the work tape pointer going past the beginning or end we simply wrap it around.
- The cells themselves are simply integers from some finite range. In the results presented we use 5 symbols per cell.
- To deal with under/over flow in a cell we take a modulo.
- We add the % command that writes a random symbol to the work tape and thus allows stochastic behaviour.
- We add the # symbol to mark the end of a program.

Extending BF to compute RL environments

This is fine for normal Turing computations, but we need to compute reinforcement learning environments.

Firstly we uniformly sample program symbols until we hit a $\#$. Then in each cycle:

- On the input tape we place the agent's action history, with the most recent action first.
- We then run the sampled BF program to compute the environment's response.
- We interpret the output tape as containing a reward signal followed by a non-rewarding observation signal. Both are sent to the agent, with reward scaled to the range -100 to +100.

Dealing with non-halting programs

Programs need to halt with output in each cycle. We achieve this in a few ways:

- All programs must contain a "." command to write output.
- Trivial infinite loops "[]" are not allowed in programs.
- Any attempt to write excess output is interpreted as a halt.

As a result, 90% of randomly sampled programs terminate with output within 1000 computation steps in each cycle. We discard any program that does not.

Because we want programs to respond to the agent's actions, we require them to contain the "," symbol that reads the input tape.

Variance Reduction Techniques

If we take this extended BF reference machine and use simple Monte Carlo sampling it works. The problem is that it takes a long time to get enough samples to get statistically significant results.

To address this, we employ three variance reduction techniques:

- Adaptive stratified sampling
- Antithetic variates
- Common random numbers

These do not affect the results, they just get there faster.
We implemented simple Monte Carlo and used it to check this.

While this was the most complex part of the project, they are all standard statistical methods. See the paper for details. Typical speedup was by a factor of 4, sometimes much more.

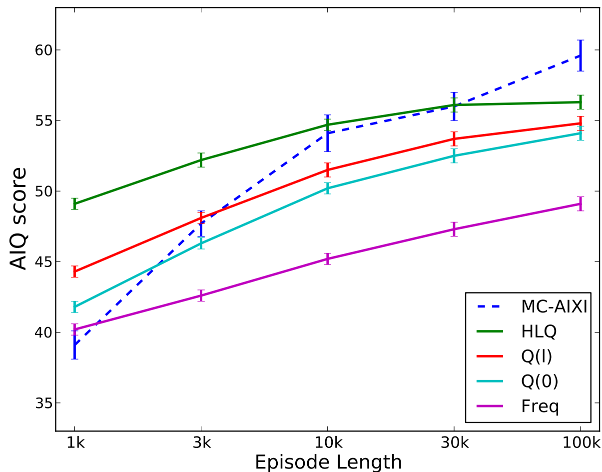
Empirical Tests

We tested the AIQ system on the following RL agents:

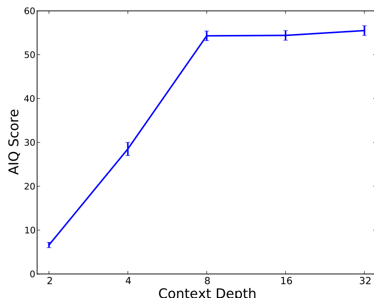
- MC-AIXI. This is a model based RL agent that can learn to play simple games such as TicTacToe, Kuhn poker and PacMan, but is rather slow to learn.
- HLQ(λ). This is a tabular RL agent similar to Q learning but with an automatic learning rate.
- Q(λ). A standard RL agent, Q(0) is a weaker special case.
- Freq. The agent simply does the more rewarding action most of the time, occasionally trying a random action.

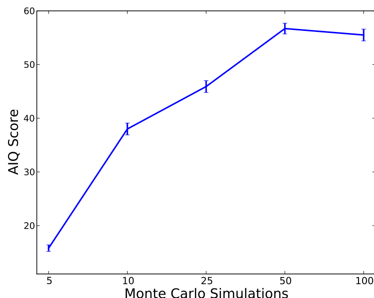
We tried a number of variants on the basic BF set up and they didn't have much impact. The main important parameter was the number of symbols on the tapes. With more symbols algorithms took longer to learn. Here we present the results with 5 symbols.

Estimated Agent AIQ Is Well Behaved!



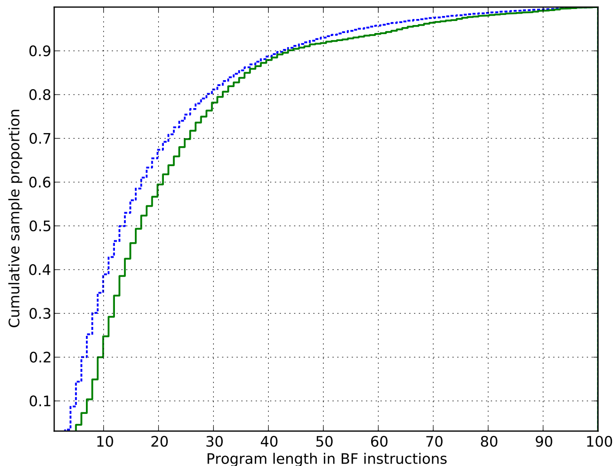
AIQ Scaling with MC-AIXI





A convenient feature of MC-AIXI is that we can easily scale the context length that the model uses, and the amount of search effort. In both cases we see the agent's AIQ behaving sensibly.

Program Lengths Without and With Adaptive Sampling



Conclusion

- Solomonoff sampling seems to be a reasonable way to approximate Universal Intelligence.
- Extending a simple reference machine to compute reinforcement learning environments was not difficult.
- Variance reduction can be used to speed up testing. Typically by a factor of 4, sometimes much more.
- As BF was the first reference machine we tried, and it seems to work well, this suggests that it isn't too difficult to find a reasonable reference machine. Further tests with alternate reference machines will be needed to explore this.
- Full Python source code available at www.vetta.org/aiq