Reinforcement Learning Beyond Small MDPs: Practical Generic Reinforcement Learning

Mayank Daswani, Peter Sunehag (presenting), Marcus Hutter

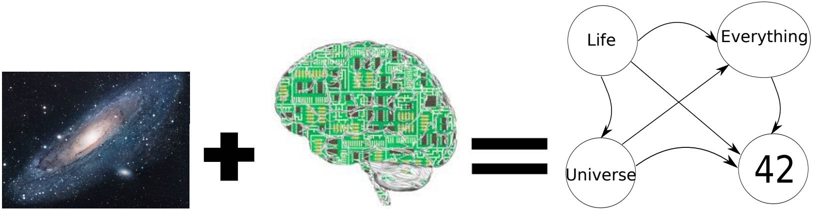
THE AUSTRALIAN NATIONAL UNIVERSITY

2014

Table of Contents

Table of Contents

Feature Reinforcement Learning



Feature RL aims to automatically reduce a complex real-world non-Markovian problem to a useful (computationally tractable) representation (MDP).

Formally we create a map ϕ from an agent's history to a state representation. ϕ is then a function that produces a relevant summary of the history.

$$\phi(h_t) = s_t$$

Feature Markov Decision Process (ΦMDP)

To select the best ϕ, one defines a cost function.

$$\phi_{best} = \arg\min_{\phi}(Cost(\phi)).$$

- Feature RL is a recent framework.
- Original cost from Hutter 2009 is a model-based criterion.

$$Cost(\phi|h) = CL(s_{1:n}|a_{1:n}) + CL(r_{1:n}|s_{1:n}, a_{1:n}) + CL(\phi)$$

A practically useful modification adds a parameter α to control the balance between reward coding and state coding,

$$Cost_{\alpha}(\phi|h_n) := \alpha CL(s_{1:n}|a_{1:n}) + (1-\alpha) CL(r_{1:n}|s_{1:n}, a_{1:n}) + CL(\phi).$$

- A global stochastic search (e.g. simulated annealing) is used to find the ϕ with minimal cost.
- For fixed ϕ, MDP methods can be used to find a good policy

Model-free cost criterion

Daswani&Sunehag&Hutter 2013 introduced a fitted-Q cost

$$Cost_{QL}(\phi) = \min_Q \frac{1}{2} \sum_{t=1}^{n} (r_{t+1} + \gamma \max_a Q(\phi(h_{t+1}), a) - Q(\phi(h_t), a_t))^2 + Reg(\phi)$$

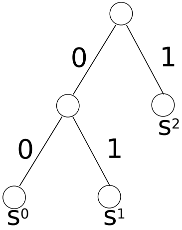
- $Cost_{QL}$ also extends easily to the linear function approximation setting by approximating $Q(h_t, a_t) \leftarrow \xi(h_t, a_t)^T w$ where $\xi : \mathcal{H} \times \mathcal{A} \to \mathbb{R}^k$ for some $k \in \mathbb{R}$.
- Connects feature rl to feature selection for TD methods, e.g. Lasso-TD or Dantzig-TD using ℓ_1 regularization while above *Reg* tends to be a more aggressive ℓ_0.
- For a fixed policy, a TD cost without $\max_a$ can be defined but one can also reduce the problem to feature selection for supervised learning using pairs (s_t, R_t) where R_t is the return achieved after state s_t.

Input : Environment $Env()$;
Initialise ϕ ;
Initialise history with observations and rewards from $t = init\_history$ random actions;
Initialise M to be the number of timesteps per epoch;
while *true* **do**
    $\phi = SimulAnneal(\phi, h_t)$;
    $s_{1:t} = (\phi(h_1), \phi(h_2), ..., \phi(h_t))$;
    $\pi = FindPolicy(s_{1:t}, r_{1:t}, a_{1:t-1})$;
    for $i = 1, 2, 3, ...M$ **do**
        $a_t \leftarrow \pi(s_t)$;
        $o_{t+1}, r_{t+1} \leftarrow Env(h_t, a_t)$;
        $h_{t+1} \leftarrow h_t a_t o_{t+1} r_{t+1}$;
        $t \leftarrow t + 1$;
    end
end

Algorithm 1: A high-level view of the generic ΦMDP algorithm.

Feature maps

- Tabular : use suffix trees to map histories to states (Nguyen&Sunehag&Hutter 2011,2012). Looping trees for long-term dependences (Daswani&Sunehag&Hutter 2012)
- Function approximation : define a new feature class of event selectors. A feature ξ_i checks the $n - m$ position in the history (h_n) for an observation-action pair (o, a).



If the history is $(0, 1), (0, 2), (3, 4), (1, 2)$ then a event-selector checking 3 steps in the past for the observation-action pair $(0, 2)$ will be turned on.

Table of Contents

Bayesian general reinforcement learning: MC-AIXI-CTW

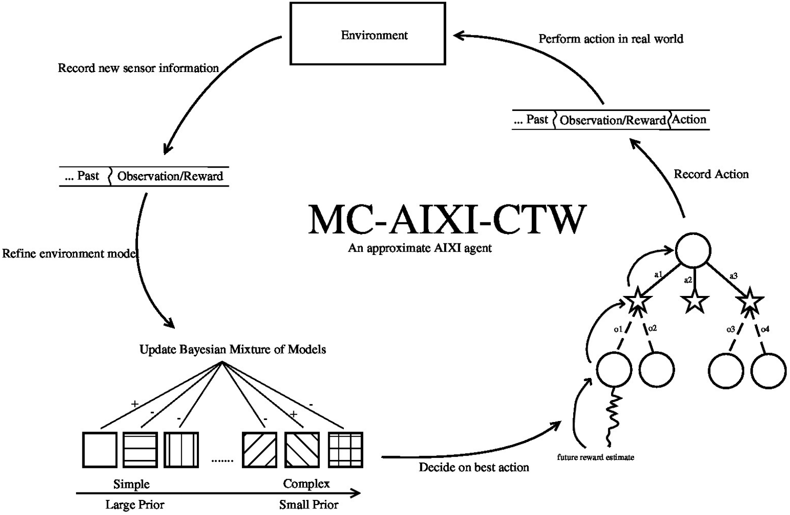
Unlike Feature RL, the Bayesian approach does not pick one map but uses a mixture of all instead. The problem is (again) split into two main areas:

- Learning - online sequence prediction / model building
- Planning/Control - search / sequential decision theory

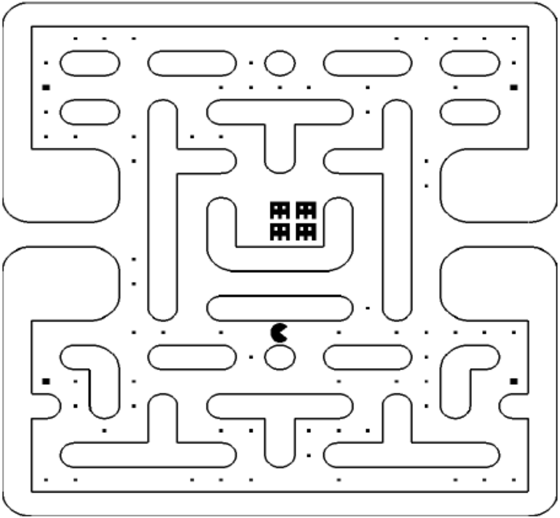
The hard parts:

- Large model class required for Bayesian mixture predictor to have *general* prediction capabilities.
- Fortunately, an efficient and general class exists: all Prediction Suffix Trees of maximum finite depth D. Class contains over $2^{2^{D-1}}$ models!
- The planning problem can be performed approximately with Monte-Carlo Tree Search (UCT)
- MC-AIXI-CTW (Veness et. al. 2010) combines the above

Overview of proposed agent architecture



Domain : POCMAN



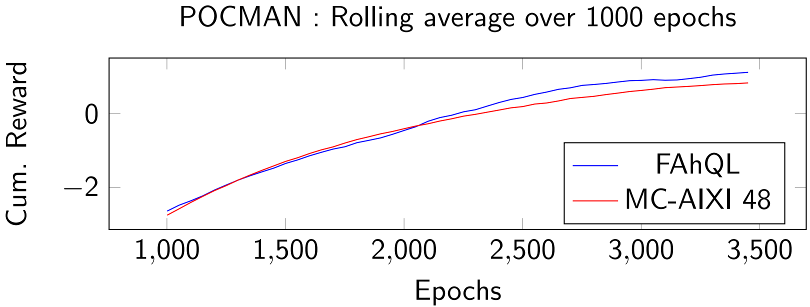


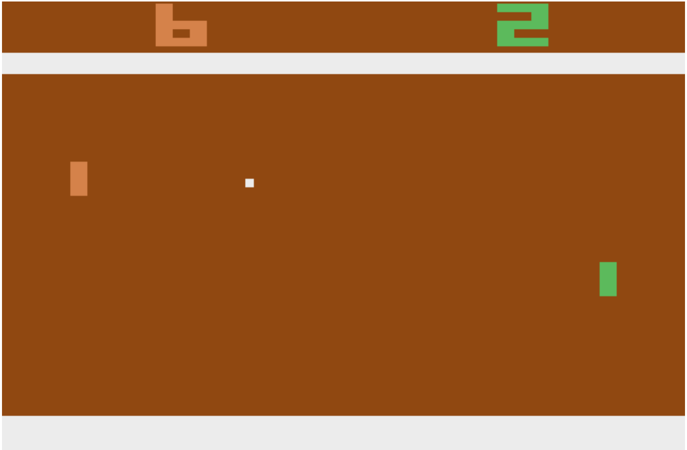
Figure : MC-AIXI vs hQL on Pocman

Agent	Cores	Memory(GB)	Time(hours)	Iterations
MC-AIXI 96 bits	8	32	60	$1 \cdot 10^5$
MC-AIXI 48 bits	8	14.5	49.5	$3.5 \cdot 10^5$
FAhQL	1	0.4	17.5	$3.5 \cdot 10^5$

Table of Contents

The Arcade Learning Environment (ALE)

ALE (Nadaf 2010, Bellamare et. al. 2012) is an interface built upon the open-source Atari 2600 emulator Stella. It provides a convenient interface to ATARI 2600 games.



Features for ALE

- Basic Abstraction of Screen Shots (BASS, from Nadaf 2010) first stores a background of the game it's playing. Then for every frame it subtracts away the background and divides the screen into 16x14 tiles. For each colour (8-bit SECAM) it creates a feature. It then takes the pairwise interaction of all these resulting features resulting in 1,606,528 features.
- Color provides object recognition.
- We study linear function approximation with BASS. We want to see how well one can do with that if one finds the right parameters
- Improved results has been achieved with non-linear neural/deep approaches.

The gap

Table : The gap between score (more is better) achieved by (linear) learning and (uct) planning

Game	UCT	BestLearner
Beam Rider	6,624.6	929.4
Seaquest	5,132.4	288
Space Invaders	2,718	250.1
Pong	21	-19

Out of 55 games, UCT has the best performance on 45 of them. The remaining games require a terribly long horizon.

Learning from an oracle

- Reinforcement learning is made much more difficult than supervised learning due to the need to explore.
- Therefore, many authors has in recent years been developing ways of teaching an rl agent through e.g. demonstration or advice with reduction to supervised learning.
- I will here discuss this idea in the context of Atari games through the Arcade Learning Environment (ALE) framework

Learning from UCT

A common scenario when applying reinforcement learning algorithms in real-world situations, learn in a simulator, apply in the real-world.

- UCT in the "real-world" still requires the simulator.
- UCT does not provide a policy representation, merely a trajectory.
- How do you extract a complete explicit policy from UCT?
- We will treat the value estimates from UCT as advice provided to the agent and we can then learn to play Pong with just a few episodes of data.
- Learning the value function is now a regression problem we solve using LibLinear (also exploring kernels, brings us back to feature selection/sparsification)
- Similar to the Dataset Aggregation algorithm for imitation learning (Ross and Bagnell 2010)

DAgger for reinforcement learning with advice Initialise $D \leftarrow \emptyset$
Initialise $\pi_1 (= \pi^*)$ t = 0 **for** *i = 1 to N* **do**

  while *not end of episode* **do**

    for *each action a* **do**

      Obtain feature $\phi(s_t, a)$ and oracle's $Q^*(s_t, a)$

      Add training sample $\{(\phi(s_t, a), Q^*(s_t, a))\}$ to D_a.

    end

    Act according to π_i

  end

  for *each action a* **do**

    Learn new model $\hat{Q}_i^a := {w_i^a}^\top \phi$ from D_a using regression

  end

  $\pi_i(\phi) = \arg\max_a \hat{Q}_i^a(\phi)$

end

Preliminary results

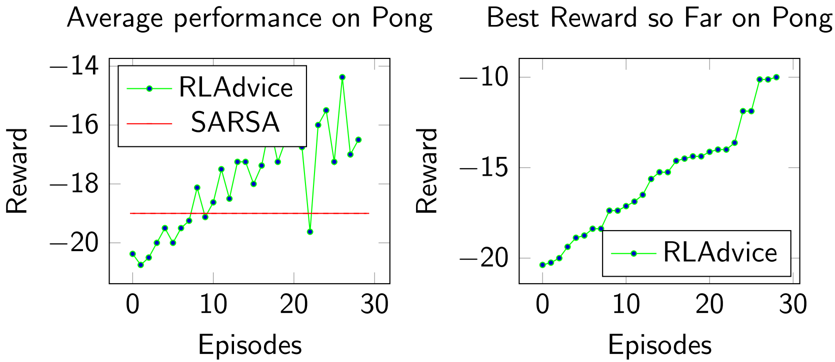


Figure : Pong Results: RLadvice with different amount of aggregated data (1-30 games) vs SARSA (linear function approximation) after 5000 games played. Results averaged over 8 runs

Conclusions/Outlook

- Reinforcement Learning is a powerful paradigm within which (basically) all AI problems can be formulated
- Many practical successes using MDPs by engineering problem reductions/reprentations
- Practically increasing the versatility of agents by learning reductions automatically.
- Recently introduced arcade gaming environment (from Alberta) for RL containing all ATARI games. Aim, have one generic RL agent solve all!
- Use data on how valuable states are from either simulations or experience to reduce complexity