Australian National University

# Reinforcement learning with value advice

Mayank Daswani, Peter Sunehag, Marcus Hutter

Research School of Computer Science,
CECS,
Australian National University.

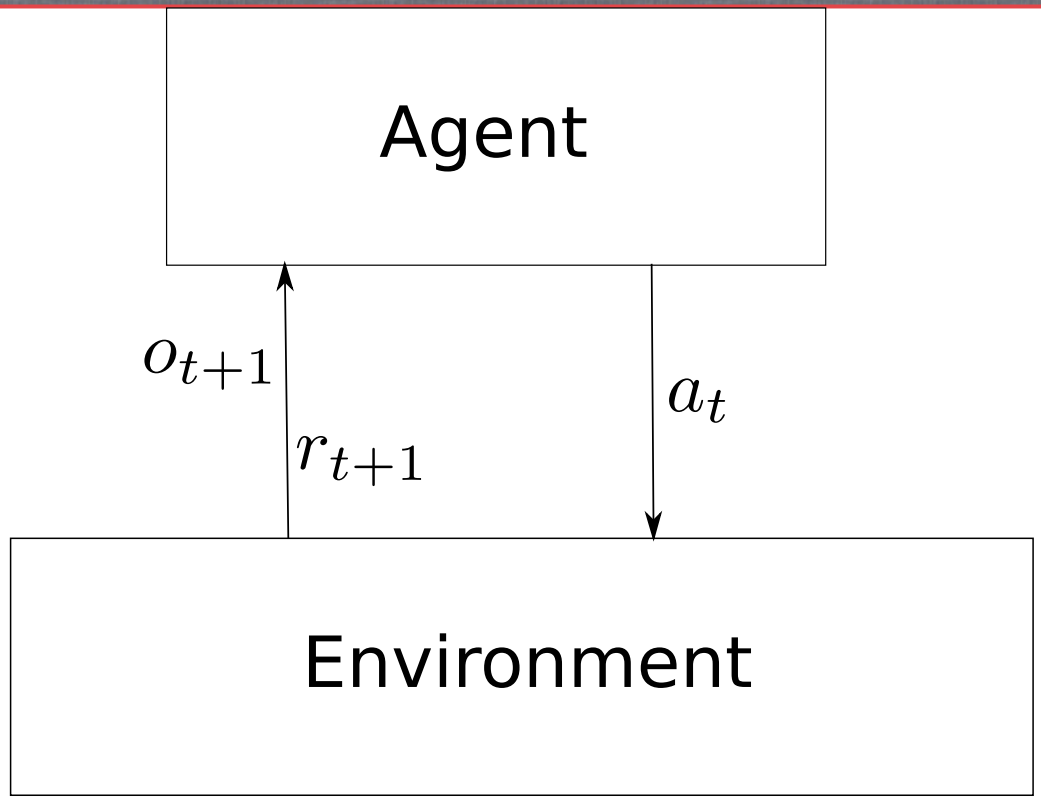
28th Nov 2014

# Abstract

The problem we consider in this paper is reinforcement learning with value advice. In this setting, the agent is given limited access to an oracle that can tell it the expected return (value) of any state-action pair with respect to the optimal policy. The agent must use this value to learn an explicit policy that performs well in the environment. We provide an algorithm called RLAdvice, based on the imitation learning algorithm DAgger. We illustrate the effectiveness of this method in the Arcade Learning Environment on three different games, using value estimates from UCT as advice.

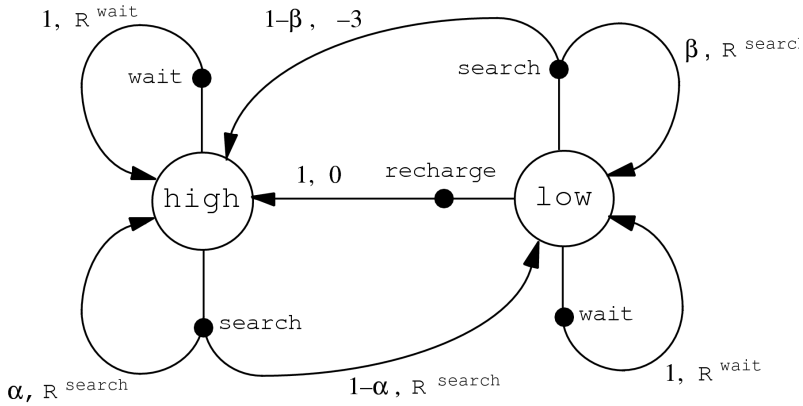
# Outline

# The RL problem




- An RL agent interacts with the environment by performing actions and receiving states and rewards.
- Construct an agent that learns good behaviour by trial-and-error interactions with the environment.

# Reinforcement Learning I




Source : Reinforcement Learning : [SB98].

- A Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, R \rangle$
- $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $R : \mathcal{S} \times \mathcal{A} \rightsquigarrow \mathcal{R}$ is the (possibly stochastic) reward function which gives the (real-valued) reward gained by the agent after taking action $a$ in state $s$.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the *state-transition function*.
- $\gamma \in [0, 1]$ is a discount factor.

# Reinforcement learning II

- A policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a function from states to actions.
- The future discounted return at time $t$ is $R_t = \sum_{k=t}^{T} \gamma^{k-t} r_{k+1}$ where $T$ is the end of the episode.
- The value functions for a particular policy $\pi$ are given by the Bellman equations as follows,

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}[R_t | s_t = s] \\
&= \sum_{a} \pi(s, a) Q^{\pi}(s, a) \\
Q^{\pi}(s, a) &= E_{\pi}[R_t | s_t = s, a_t = a] \\
&= R(s, a) + \gamma \sum_{s'} T(s, a, s') V^{\pi}(s')
\end{aligned}
$$

# Function approximation (FA)

- When the state space is very large or continuous, learning the exact value function is computationally infeasible.
- Instead we use a parameterised class of function approximators to compactly represent the value function.
- This also allows the agent to generalise to new states that it has not previously seen.

## Linear FAs

Let $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ be some real-valued $d$-dimensional features of the state. Then a linear function approximator approximates $Q$ using $\phi$ as follows,

$$Q(s, a) \approx w^\top \phi(s, a)$$

where $w \in \mathbb{R}^d$ is a learned weight vector.

# RL with value advice

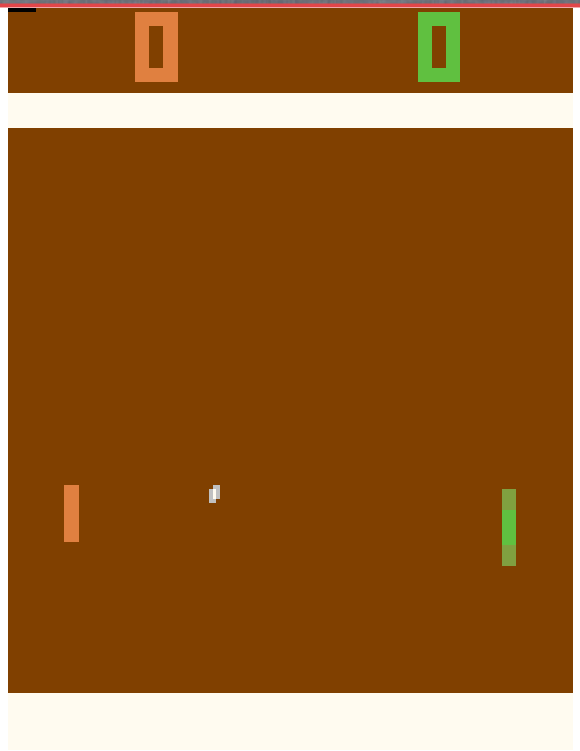
## The problem

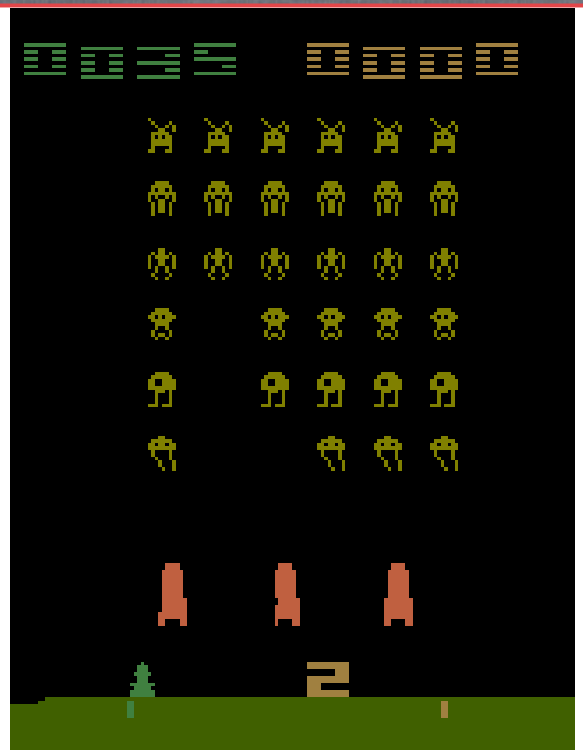
Given an oracle that provides good approximations of the optimal values for each state-action pair ($Q^*(s, a)$), can the agent learn a good value function representation in some function approximation class?

# Motivation

- Stochastic planning algorithms (such as UCT) have access to a generative model or an emulator of the environment.
- They can have excellent performance.
- They do not provide a complete explicit policy, simply an action at every iteration.
- They can be computationally expensive to use in real-time situations.
- Therefore there is a need to learn explicit, efficient and complete policies from such algorithms.
- *Training phase : access to the oracle. Testing phase: with learned policy.*

# Domain : Arcade Learning Environment



- A new benchmark domain for reinforcement learning agents.
- Provides an interface to the open-source ATARI2600 emulator STELLA.
- Access to hundreds of games in different styles.

# Gap between UCT and SARSA on the ALE

Table : The gap between learning and planning

| Game | UCT | SARSA on BASS |
|---|---|---|
| Beam Rider | 6,624.6 | 929.4 |
| Seaquest | 5,132.4 | 288 |
| Space Invaders | 2,718 | 250.1 |
| Pong | 21 | $-19$ |
| Atlantis | $1.94 \cdot 10^5$ | 25,375 |

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

Can we improve performance on the *same* class of function approximators used to generate the learning results?

# Algorithm : RLAdvice

- Collect training samples from the oracle for feature-action pairs following some policy $\pi_i$.
- We use features based on the screen so we collect samples for each action individually $D_a \leftarrow D_a \cup \{\phi(s_t), Q^*(s_t, a)\}$.
- Train a regressor $w_i^a$ based on the training data for each action, that learns a linear value function approximation to $Q^*(s, a)$.
- Use the trained regressor to give a policy $\pi_{i+1}(s) = \arg\max_a {w_i^a}^\top \phi(s)$.
- Repeat.

**How should we sample states?**

From the policy learned from the data so far. [RB10].

**Algorithm 1:** Reinforcement learning with value advice

Initialise $D \leftarrow \emptyset$.
Initialise $\pi_1 (= \pi^*)$.
t $= 0$
**for** *i = 1 to N* **do**
  **while** *not end of episode* **do**
    **foreach** *action a* **do**
      Obtain feature $\phi(s_t)$ and oracle expected return $Q^*(s_t, a)$.
      Add training sample $\{\phi(s_t), Q^*(s_t, a)\}$ to $D_a$.
    **end**
    Act according to $\pi_i$.
  **end**
  **foreach** *action a* **do**
    Learn new model $\hat{Q}_i^a := {w_i^a}^\top \phi$ from $D_a$ using regression.
  **end**
  $\pi_{i+1}(\cdot) = \arg\max_a \hat{Q}_i^a(\cdot)$ .
**end**

# Objective function

The regression optimises the following least squares objective, where $Q^*(s, a)$ is provided by the oracle.

$$w_i^a = \arg\min_v \left( \frac{1}{2} v^\top v + C \sum_{(\phi(s), Q^*(s,a)) \in D_a} (v^\top \phi(s) - Q^*(s, a))^2 \right) \quad (1)$$

# Relation to online learning

- Dataset Aggregation (DAgger) by [RB10] is an imitation learning algorithm that uses guarantees from online learning.
- RLAdvice is a modified DAgger for the (full information) value advice problem.
- Each epoch of the algorithm can be viewed as an iteration of an online learning problem where the agent picks a value function approximation and is given a corresponding loss by an adversary.
- The data collected by the agent provides an empirical estimate of the loss.
- The agent then follows a policy based on the representation with the best loss on the data so far.
- This is *Follow-the-leader*, a no-regret algorithm in online learning.

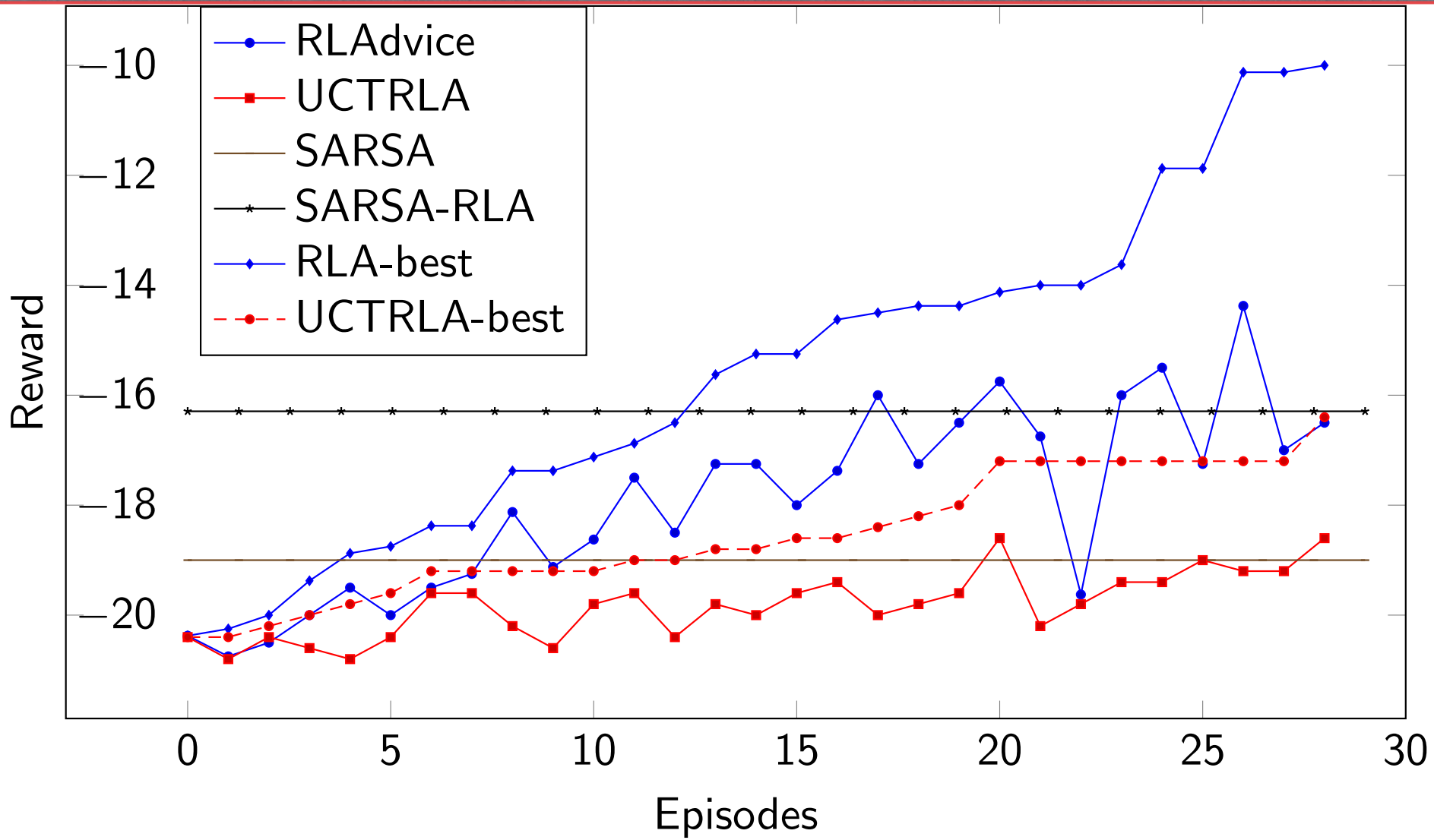
# BASS features

- Basic Abstraction of Screen Shots (BASS) features defined by [BNVB13].
- Tiles the screen into blocks, with each block containing indicator functions for each SECAM colour. A feature is *on* if a particular colour was present in the block.
- Also contains the pairwise AND of all such features. Thus the feature space is 1.6 million features.

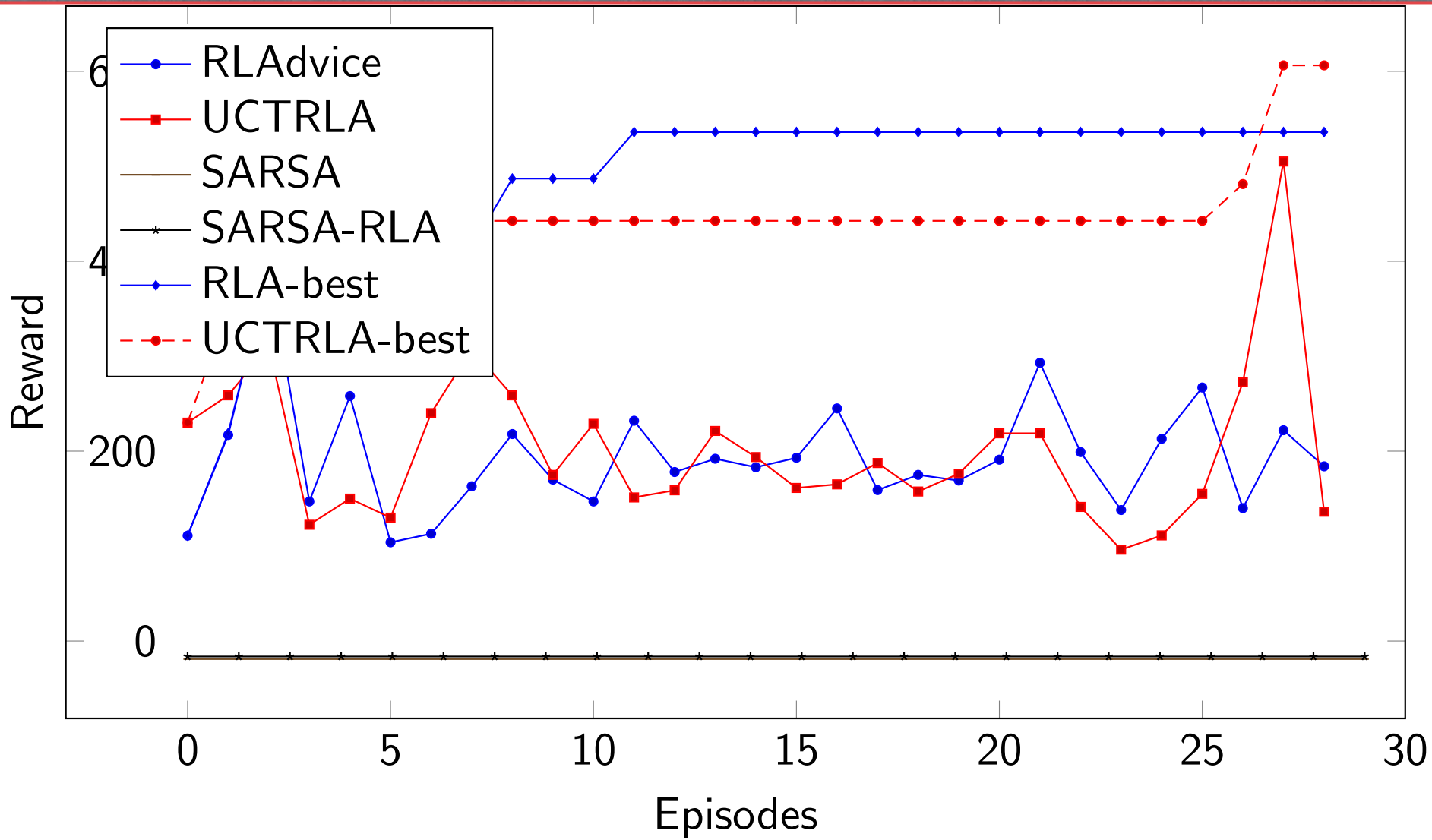
# Comparisons

- SARSA (traditional reinforcement learning) [SARSA].
- RLAdvice using regression. We have the following variations,
  - Training starting with a UCT policy and then iterating the model [RLAdvice].
  - Training only with the UCT policies [UCTRLA].
- SARSA trained using the regression weights obtained from RLAdvice [SARSA-RLA]. The weight vector from the RLAdvice algorithm can be further improved on (in theory) by using SARSA with these weights as the initialisation.

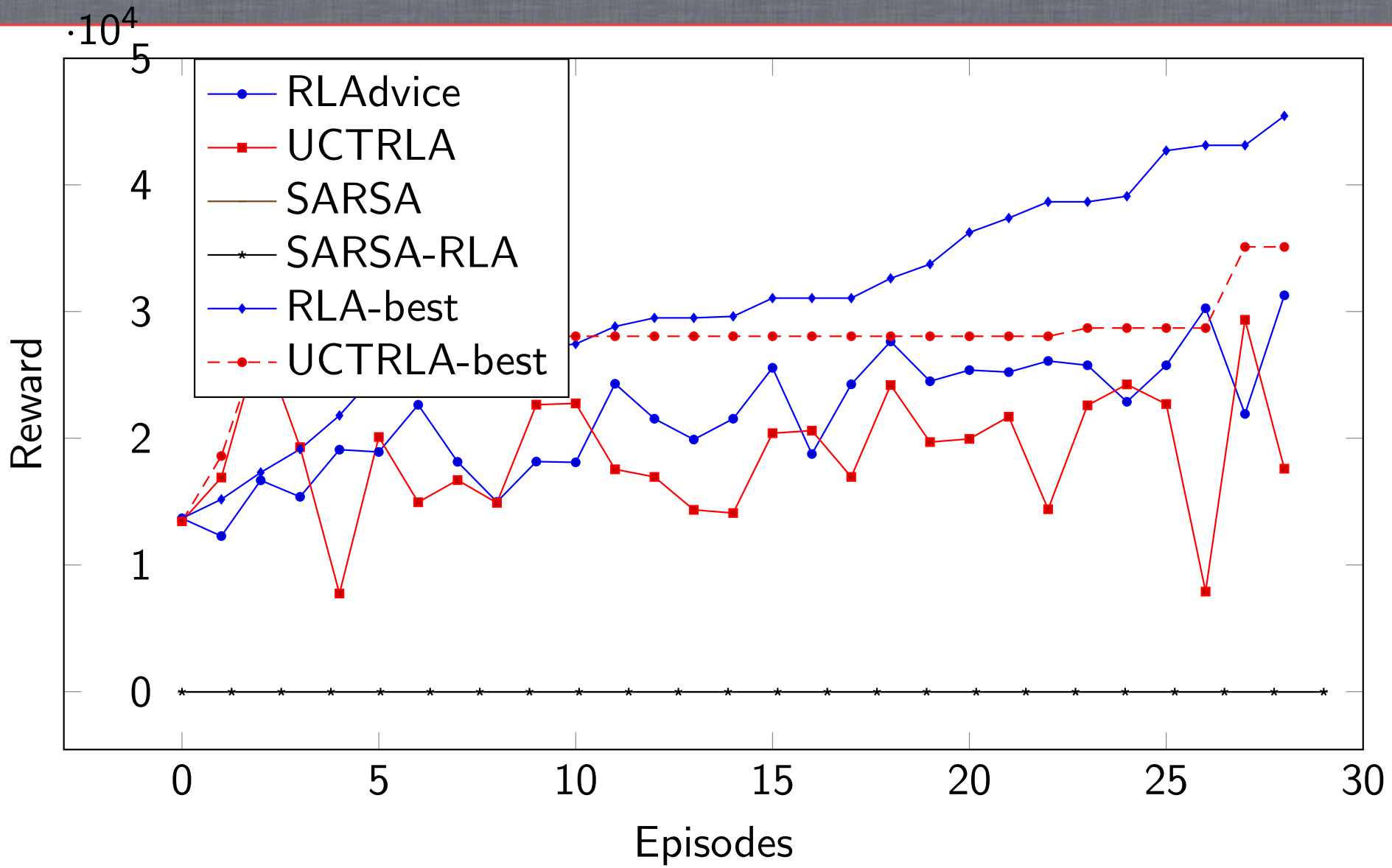
# Results using LIBLINEAR : Pong



# Results using LIBLINEAR : Space Invaders



# Results using LIBLINEAR : Atlantis



# Computation time

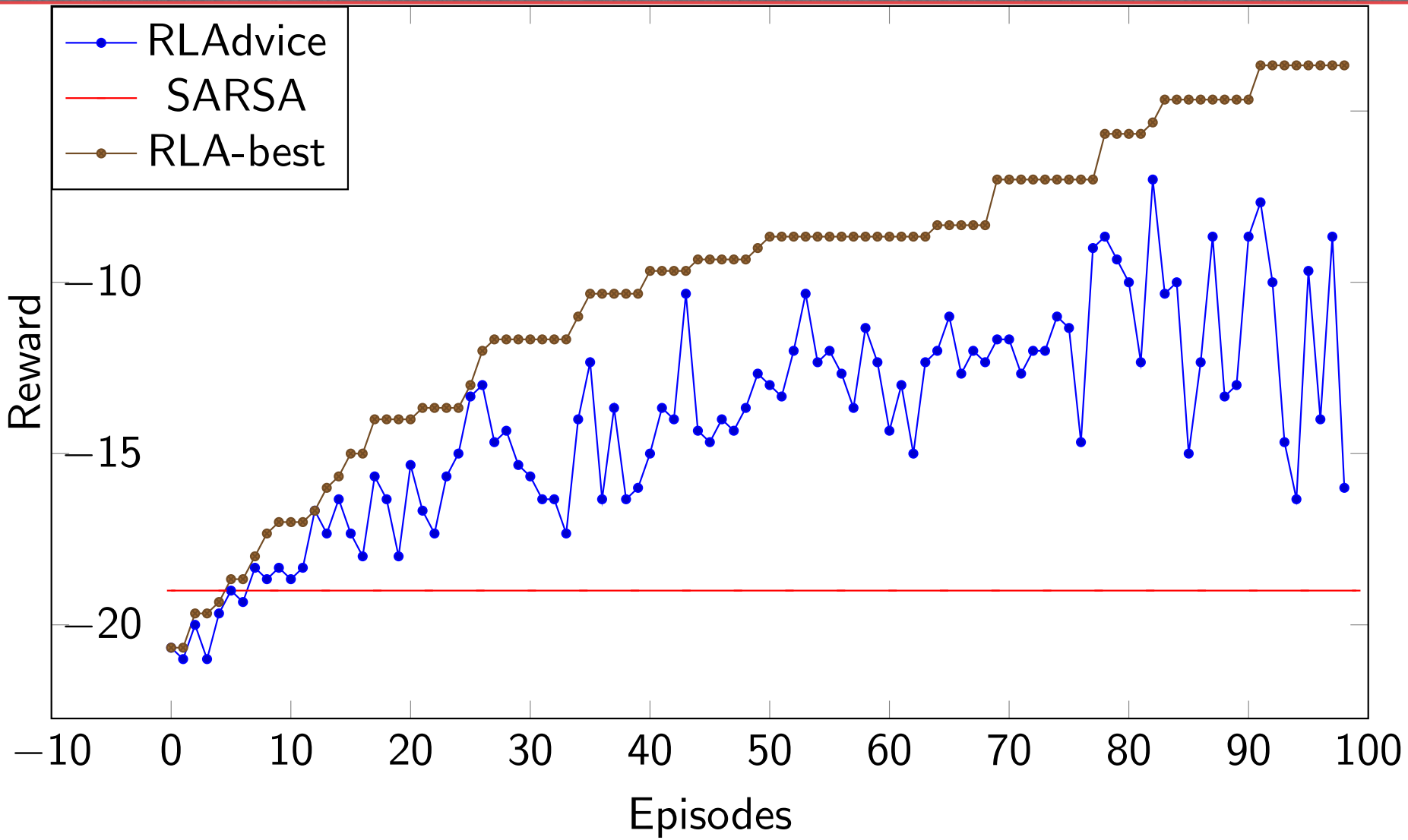
Table : Timing data on Atlantis, the most computationally expensive game, using LIBLINEAR for RLAdvice and UCTRLA. The time for RLAdvice includes generating the advice using UCT. The time for SARSA-RLA here is post-RLAdvice training and would be comparable to time taken by standard SARSA.

| Algorithm | Training episodes | Training time | Testing time (per episode) |
|---|---|---|---|
| SARSA-RLA | 5000 | 22.01 hours | 1 second |
| RLAdvice | 30 | 73.99 hours | 1 second |
| UCTRLA | 30 | 118.38 hours | 1 second |
| UCT | N/A | N/A | 3600 seconds |

# Warm starts using SDCA

- In order to speed-up the regression process by warm-starting each epoch we implemented Stochastic Dual Coordinate Ascent.
- We obtain a significant speedup on Atlantis and Space Invaders.
- Unfortunately we did not have the time to rerun all the experiments using this.

# Pong using SDCA



# Conclusion

- RLAdvice extends DAgger to the "full information" value advice setting.
- It allows us to extract complete, explicit policies from UCT.
- It can also tell us about the suitability of a value function approximation class as a result.

# Related Work

- Imitation learning : agent sees only the best action from the oracle (partial information). Additionally, learning task is to imitate the oracle (rather than find a good policy).
- RL with advice. Advice can be in the form of instructions in some language [MS96], input policies [ALB13], bounds on the value function [MST$^+$05], potential functions over the state-action pairs [WCE03] , recommended actions [TCF$^+$14].

# Future work

- Consider a teacher that is more aware of student interactions (teacher-student à la [TCF+14] rather than oracle-learner).
- Consider limiting the number of calls to the oracle (known as budgeted advice).
- Use a different class of function approximators, deep learning approaches show great results! [MKS+13]

# References I


[ALB13] M. G. Azar, A. Lazaric, and E. Brunskill. Regret bounds for reinforcement learning with policy advice. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Zelezný, editors, *ECML/PKDD (1)*, volume 8188 of *Lecture Notes in Computer Science*, pages 97–112. Springer, 2013.

[BNVB13] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

[MKS+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D., and M. Riedmiller. Playing Atari with deep reinforcement learning. 2013.

[MS96] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1-3):251–281, 1996.

# References II


[MST+05] R. Maclin, J. W. Shavlik, L. Torrey, T. Walker, and E. W. Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In M. M. Veloso and S. Kambhampati, editors, *AAAI*, pages 819–824. AAAI Press / The MIT Press, 2005.

[RB10] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and D. Mike Titterington, editors, *AISTATS*, volume 9 of *JMLR Proceedings*, pages 661–668. JMLR.org, 2010.

[SB98] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[TCF+14] M. E. Taylor, N. Carboni, A. Fachantidis, I. P. Vlahavas, and L. Torrey. Reinforcement learning agents providing advice in complex video games. *Connect. Sci.*, 26(1):45–63, 2014.

# References III

[WCE03] E. Wiewiora, G. W. Cottrell, and C. Elkan. Principled methods for advising reinforcement learning agents. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 792–799. AAAI Press, 2003.