

ANTICIPATIVE REINFORCEMENT LEARNING

¹Frederic Maire

Smart Devices Laboratory
School of Computing Science and Software Engineering
Queensland University of Technology
Box 2434, Brisbane Qld 4001, Australia
f.maire@qut.edu.au

ABSTRACT

This paper introduces Anticipative Reinforcement Learning (ARL), a method that addresses the problem of the breakdown of value based algorithms for problems with small time steps and continuous action and state spaces when the algorithms are implemented with neural networks. In ARL, an agent is made of three components; the actor, the critic and the model (the model is as in Dyna but we use it differently). The main originality of ARL lies in the action selection process; the agent builds a set of candidate actions that includes the action recommended by the actor plus some random actions. Once the set of candidate actions is built, the candidate actions are ranked by considering what would happen if these actions were taken and followed by a sequence of actions using only the current policy (anticipation using iteratively the model with a finite look-ahead). We demonstrate the benefits of looking ahead with experiments on a Khepera robot.

1. INTRODUCTION

What makes Reinforcement Learning (RL) so attractive is that it is a form of unsupervised learning where the desired behaviour of an agent is defined implicitly by the rewards given by the environment. No supervisor is needed to show the learner how to perform the task. The agent is expected to discover a good policy by simply interacting with its environment. The book by Sutton and Barto [1] is an excellent introduction to RL. For a more formal treatment, the reader is referred to the book by Bertsekas and Tsitsiklis [2].

In the Reinforcement Learning framework, an agent acts in an environment whose state s_t at time t it can sense (partial observability), and occasionally receives some penalty or reward r_t . The goal of the agent (its

learning task) is to find a policy for action selection that maximizes its *return*, that is the expectation of the discounted cumulative reward $E\left(\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k}\right)$. The *value* $V_{\pi}(s)$ of a state s with respect to a policy π is defined as the expected return when using policy π and starting in state s . Similarly the *action-value* $Q_{\pi}(s, a)$ is defined as the expected return when starting in state s and selecting action a for the first step then following policy π in the subsequent steps. Although there are many ways to attack the RL problem, including gradient descent on the policy without considering a value function [3], the paradigm we follow is to construct a value function that evaluates the “goodness” of different states.

Recent research has produced a flurry of algorithms for learning value functions, theoretical insights into their power and limitations, and a series of fielded applications. It is widely acknowledged that to be of use in complex domains, RL techniques must be combined with generalizing function approximators such as neural networks. In most real-world applications robots are situated in an environment that is not discrete. Moreover, the set of available actions may also not be discrete. Gaskett, Weetergreen and Zelinsky have compared in a survey paper [4] several recent attempts at extending Q-learning to continuous state and action spaces [5,6,7,8,9]. The main problem with the extension of RL algorithms to continuous spaces is the relative similarity of the Q-values for the same state and different actions compared to the values of different states. Another problem is that for small time steps, successive states have very close values, and require a more accurate estimation of their true values to correctly rank them. Moreover the smaller the time steps are, the smaller the learning rates must be (this entails a slow convergence if any).

By introducing a model of the environment and looking several steps ahead at the consequences of

¹ This work was partially supported by an Australian Technical Universities Network grant.

candidate actions, we should be able to better discriminate between candidate actions. The new RL method that we introduce here and call Anticipative Reinforcement Learning (ARL) is built on this idea and is suitable for control tasks which require continuous actions, in response to continuous states.

In this paper, we consider an episodic task to illustrate our RL algorithm. After recalling the difficulties associated with continuous spaces in section 2, we describe ARL in section 3. In section 4, we present experimental results.

2. THE CHALLENGES POSED BY CONTINUOUS SPACES

The number of training iterations necessary to sufficiently accurately represent the optimal value function when using function approximators scales poorly with the size of the time interval between states. The greater the number of actions per unit time (the smaller the increment in time between actions) the greater the number of training iterations required to adequately learn the value function. As the time interval between states decreases in size, the required precision in the approximation of the optimal value function increases exponentially [10]. *Advantage Learning* and *Residual Algorithms* address to a certain extent this problem for Q-learning [11,14,15], but have their own limitations.

3. ANTICIPATIVE REINFORCEMENT LEARNING

An ARL agent is made of three components; the *actor*, the *critic* and the (environment) *model*. The actor and the critic are the same as in *Actor-Critic* methods [12]. The model component has the same architecture as the model component of Dyna [13], but its use is different.

In Dyna, the environment model is used to simulate complete episodes. These simulated episodes are then used as training data for the agent. The agent used the same learning algorithms for these simulated episodes as for the non-simulated episodes. Whereas in ARL, the model is used to better discriminate between candidate actions.

The actor is the object implementing the policy of the agent; the mapping from state-space to action-space that defines the behaviour of the agent. The critic is the object implementing the value function; the real-valued mapping from state-space that estimates the expectation of the cumulative discounted reward. The model is the object that predicts the next (state,reward) pair given the current (state,action) pair. In our experiments, the actor, the critic and the model were all implemented with neural networks.

The high-level pseudo-code of Anticipative Reinforcement Learning is;

- Initialize the actor with any policy
- Perform policy iteration to initialize the model and the critic with respect to the initial policy
- **while** (numEpisodes <= maxNumEpisodes) {
 - run and record an episode
 - adapt the neural networks of the agent with the episode data
- }

First, the actor is initialized with a policy to be improved, and then a standard policy-iteration algorithm is run to initialize the critic and the model. Only once this initialisation phase is over, is the actor enabled to learn. Learning occurs at the end of each episode; data collected during the episode is used to adapt the neural networks. The main originality of ARL lies in the action selection process, which we describe next.

In order to decide which action to select at time t when in state s_t , the agent builds a set of n_c candidate actions $\{ca(1), ca(2), \dots, ca(n_c)\}$ that includes the action recommended by the actor plus some random actions (the parameter n_c defines the breadth of the exploration). The random actions are there for exploratory purpose. Once the set of n_c candidate actions is built, the candidate actions are ranked by considering the following n_c different scenarios. For each candidate action $ca(i)$, the agent predicts what would happen if it took action $ca(i)$ for the first step (when in state s_t) and then used only the current policy for selecting actions in the subsequent steps. A scenario is generated by iterating the model with a sequence of actions $(ca(i), pa_{t+1}(i), \dots, pa_{t+h_i-1}(i))$, where $pa_t(i)$ represents the action returned by the actor at time t in the i^{th} scenarios (according to the current policy). Figure 1 summarises the generation of the scenarios. The look-ahead sequence is limited to a finite horizon (predict only a few steps ahead). Each candidate action $ca(i)$ is evaluated by computing the cumulative discounted return for this finite horizon. That is the ranking of the candidate action $ca(i)$ is based on the following score;

$$\text{score}(ca(i)) = \sum_{k=1}^{h_i} \gamma^{k-1} r_{t+k} + \gamma^{h_i} V(s_{t+h_i})$$

where h_i is the distance of the horizon for $ca(i)$ (not necessarily the same for all $ca(i)$'s, as some actions might lead to a terminal state more rapidly than others).

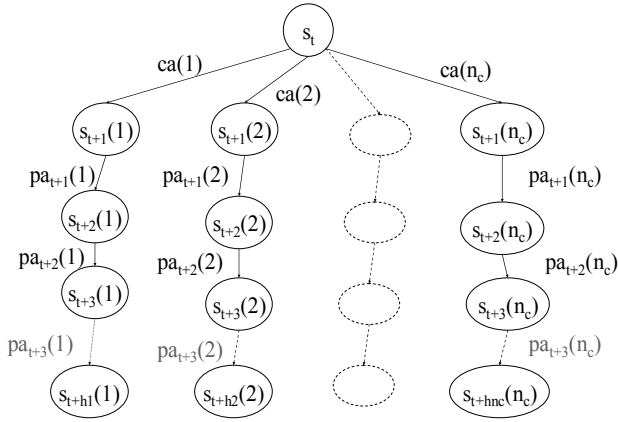


Figure 1 : Anticipative action selection process; the candidate actions are denoted by ca , actions recommended by the current policy are denoted by pa .

The highest ranking action is considered as the best action for state s_t . This fact is recorded and is used at the end of the episode to adapt the policy accordingly. The actual action performed can be selected with a softmax function using Gibbs distribution. That is the probability of selecting $ca(i)$ is

$$P(ca(i)) = \frac{e^{\text{score}(ca(i))/\tau}}{\sum_{j=1}^{n_c} e^{\text{score}(ca(j))/\tau}}$$

where τ is the temperature. Although the softmax action selection makes more unlikely the selection of a very badly ranked action, an ϵ -greedy action selection might be preferable (simpler to work with as it does not require the fine tuning of τ).

The benefit of looking ahead several steps in advance is that a less accurate estimation of the value of the states is needed to discriminate between the different candidate actions, because the states that are reached after a few steps differ more from each other than the states that are reached in just one step. The adaptation of the neural networks (actor, critic and model) at the end of each episode is done by batch; the actor learns for each state visited of the episode the highest scoring action, the critic learns the actual return, and the model learns the actual transitions.

4. EXPERIMENTS

The Matlab code used in these experiments is available at <http://www.fit.qut.edu.au/~maire/ARL/>. We

have used the *kiks* (<http://www.kiks.net>) package to control the robots; for preliminary experiments in simulation and for controlling a real Khepera robot (pictured in Figure 2). The environment of the robot is a playing field surrounded by four walls. Apart from the robot, the playing field contains a vertical pole. The main sensor for the robot is its linear camera (array of 64 pixels). The linear camera sensor returns a one dimensional array of grey intensity values. Figure 3 shows a typical reading of the linear camera.

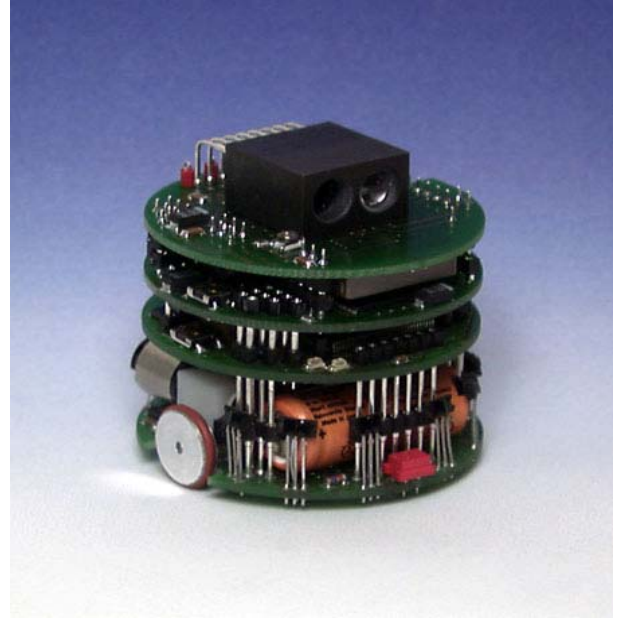


Figure 2 : A Khepera Robot with a Liner Camera

In our experiment the task of the robot is to learn to get to the pole from any position in the playing field. The robot is initialized with the simplistic policy of going straightforwardly (that is the output weights of the actor are initialized to zero). An action is a pair (left wheel speed, right wheel speed) that is sent to each wheel motor. To define the target state, the robot is put next to the pole manually. Once the robot has taken a picture of its target at close range (Figure 4), it is moved away from the pole.

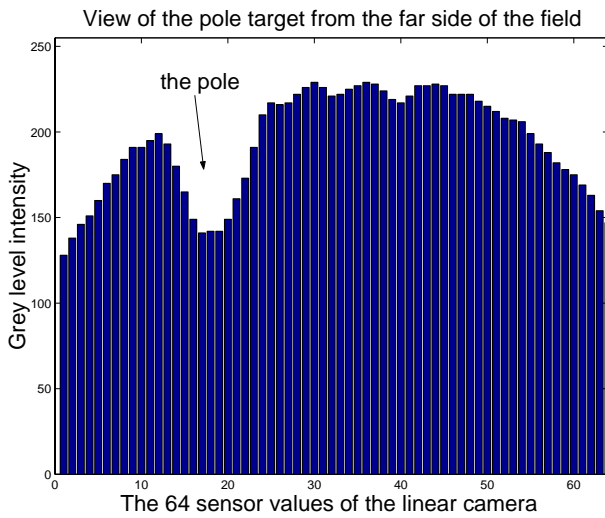


Figure 3 : Robot far from the target pole

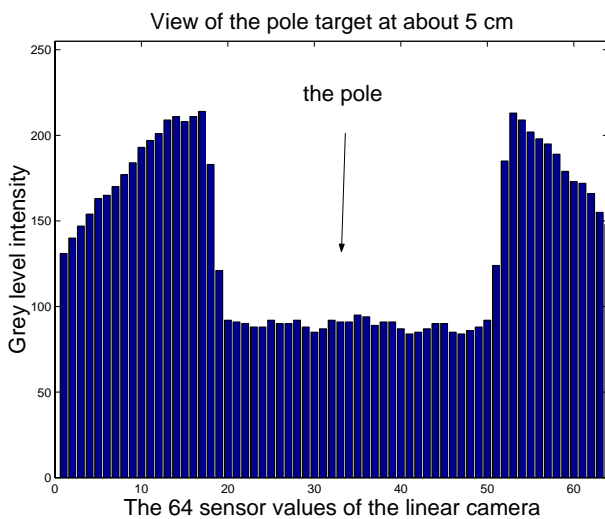


Figure 4 : Robot close to the target pole

At the beginning of each episode, the robot wanders for 20 seconds in order to place it to a random position on the playing field. An episode is stopped after 150 steps if the robot has not managed to get close to the pole. The control system of the robot has a subsumption architecture [16]. The highest priority behaviour is an obstacle avoidance neural network (with fixed weights). This behaviour becomes active when the proximity sensors return large values. If the robot does not see a linear image with a relative high variance (view of the pole), a spin behaviour is activated (robot turns on the spot). The neural network of the actor is active only when the linear image that the robot “sees” has a variance larger than a given threshold.

Figure 5 illustrates the influence of the look-ahead value on the performance (rate of success) of the robot. In all these experiments, all parameters were kept constant except the look-ahead. The learning rate was 0.001, the discount factor γ was set to 1, and the number of candidate actions was set to 10. The same number of episodes was run for different values of the lookahead. After computing the average success rate over 10 episodes, we computed the linear regression of each curve.

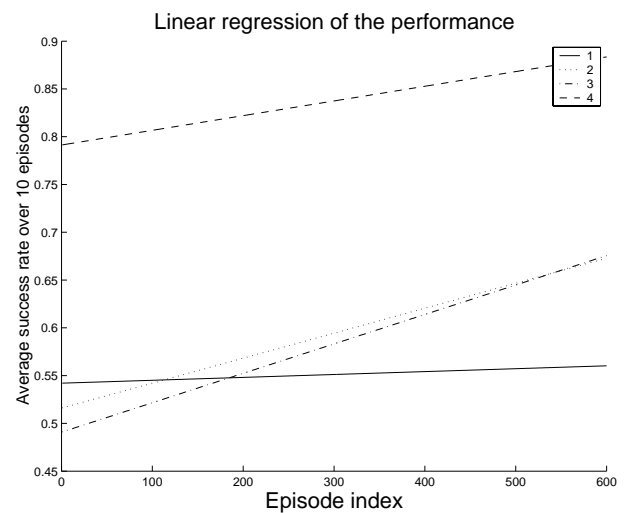


Figure 5: Learning speed and performance improve with the size of the horizon (how many steps the robot anticipates ahead).

What we observed is that a higher value for the lookahead speeds up learning and improves the overall performance. With a lookahead of 1 the robot did not improve its performance significantly, whereas with a lookahead of 4 the policy was dramatically improved. Between 5 and 12 the results are similar. As expected, when the lookahead becomes large (about 15), the performance starts to deteriorate.

Before having access to a Khepera robot, we tested our RL algorithm in a simulated environment. The task consisted in steering another virtual robot towards a target by using a pair of thrusters located on each side of the virtual robot. The virtual robot was capable of sensing the angle and distance of the target. The thrusters produced continuously variable thrust ranging from turning full leftward to turning full rightward. The controller had to learn to drive the vehicle onto the target as quickly as possible (note that the velocity vector modulus was constant, because of the way the robot was abstracted). This task was also episodic in the sense that we imposed a limit on the maximum number of time-steps the robot could use to reach the target. If the robot could manage to hit the target by the end of an episode, it would receive a

large positive reward. Moreover as an incentive to get to the target quickly, the robot received a small penalty (negative reward) at each time-step. We experimented a range of values for the speed of the robot. Unsurprisingly, we observed that if we keep all the other parameters fixed and decrease the speed of the robot, the robot has more and more difficulty discovering a good policy. At some stage, the robot had so much difficulty that its weights systematically blew-up (went to infinity) for a look-ahead of one; however it still managed to learn a successful policy with a look-ahead of two. This observation alone totally justifies the utilization and iteration of a model.

5. DISCUSSION AND CONCLUSION

Reinforcement learning techniques should be combined with function approximators in order to scale to more complex tasks. Non-generalising look-up table representations are not suited for large AI domains that cannot be searched exhaustively. Hence we need to develop RL methods for some kind of generalisation between states and actions.

We have introduced ARL, a simple and practical RL method for problems with continuous spaces.

The results reported here demonstrate that using a model of the environment and iterating this model to select an action improves the learning capabilities of the agent. In particular, the performance is increased by increasing the look-ahead.

6. REFERENCES

[1] Sutton, R., Barto, G.: Reinforcement Learning, An Introduction, MIT Press, (1998).

[2] Bertsekas, D.: Dynamic Programming and Optimal Control, Athena Scientific, Belmont, MA, (1995).

[3] Sutton, R., McAllester D., Singh S., Mansour Y.: Policy Gradient Methods for Reinforcement Learning with Function Approximation. NIPS 12, MIT Press, (2000), 1057--1063

[4] Gaskett, C., Wettergreen, D., and Zelinsky, A.: Q-Learning in Continuous State and Action Spaces, in Proceedings of 12th Australian Joint Conference on Artificial Intelligence, Springer-Verlag, Sydney, Australia, December (1999).

[5] Werbos, P.: Approximate dynamic programming for real-time control and neural modeling. In D. A. White and D. A. Sofge, editors, Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches. Van Nostrand Reinhold, (1992).

[6] Rummery, G.: Problem solving with reinforcement learning. PhD thesis, Cambridge University, (1995).

[7] Santamaria, J., Sutton, R., and Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces. Adaptive Behaviour, 6(2):163--218, (1998).

[8] Touzet, C.: Neural reinforcement learning for behaviour synthesis. Robotics and Autonomous Systems, 22(3-4):251--81, (1997).

[9] Gross, H., Stephan, V., and Krabbes, M.: A neural field approach to topological reinforcement learning in continuous action spaces. In Proc. 1998 IEEE World Congress on Computational Intelligence, WCCI'98 and International Joint Conference on Neural Networks, IJCNN'98, Anchorage, Alaska, (1998).

[10] Harmon, M.: An online reinforcement learning tutorial <http://www-anw.cs.umass.edu/mharmon/rltutorial/tut.html>

[11] Harmon, M., Baird, L.: Multi-player residual advantage learning with general function approximation. Tech Report WL-TR-96-1065. Wright-Patterson Air Force Base Ohio: Wright Laboratory. (1996).

[12] Witten, I.: An adaptive optimal controller for discrete-time Markov environment. Information Control, 34:286--295 (1977).

[13] Sutton, R. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Proceedings of the Seventh International Conference of Machine Learning, Morgan Kaufmann Publishers, San Francisco CA, 216--224 (1990).

[14] Baird, L.: Residual Algorithm: Reinforcement Learning with Function Approximation, In A. Prieditis and S. Russell, eds. Machine Learning: Proceedings of the Twelfth International Conference, Morgan Kaufman Publishers, San Francisco CA, (1995).

[15] Baird, L.: Advantage Updating, Tech Report WL-TR-93-1146. Wright-Patterson Air Force Base Ohio: Wright Laboratory, (1993).

[16] Brooks, R.: Cambrian Intelligence, MIT Press, (1999).