The Fifteenth International Symposium on Artificial Life and Robotics 2010 (AROB 15th '10),
B-Con Plaza, Beppu,Oita, Japan, February 4-6, 2010

Efficient Robotic Memory Controller for long-term Planning

Hassab Elgaw Osman

Imaging Science and Engineering Lab, Tokyo Institute of Technology, Japan.
Email: osman@isl.titech.ac.jp

Abstract

This paper contributes on designing robotic memory controller for non-Markovian reinforcement tasks. Instead of holistic search for the whole memory contents, the controller adopts associated feature analysis to produce the most likely relevant action from previous experiences. Actor-Critic (AC) learning is used to adaptively tuning the control parameters, while on-line variant of Random Forest (RF) learner is used as memory-capable to approximate the policy of Actor and the value function of Critic. Learning capability is experimentally examined through non-Markovian cart-pole balancing task. The result shows that the proposed controller acquired complex behaviors such as balancing two poles simultaneously and displays long-term planning.

Keywords: non-Markovian, actor-critic, random forest, self-optimizing controller.

1 Introduction

As with the most real-world robot learning systems, the arising of perceptual aliasing, when the system has to scale up to a non-Markov setting or POMDP[1] renders conventional reinforcement learning (RL) methods impracticable, raising an interests in heuristic methods without world model (e.g., 'memory approach') that intrinsically and adaptively modifying the policy. In Memory-based systems the controller is unable to take optimal transitions unless it observed the past inputs, then the controller simultaneously solve the incomplete perception while maximizing discounted long-term reward. If a world model is available to the controller, it can easily calculate and update a *belief vector* $\vec{b_t} = \langle b_t(s_1), b_t(s_2), \cdots, b_t(s_N)\rangle$ over 'hidden states' at every time step t by taking into a account the history trace $h = o_1, o_2, \cdots, o_{t-1}, o_t$.

2 Our Approach

The process of memorizing and scaling up could be lengthy if traditional memory scheduling processes are to be used. In order to speed up learning process and improve the convergence rate, a RL-controller is modeled as scheduler for our proposed self-optimizing adaptive memory controller (Fig.1). Rather than holistic search for the whole memory contents the model adopt associated feature analysis to successively memorize a newly experience (state-action pair) as an action of past experience. Actor-Critic (AC) learning is used to adaptively tuning the control parameters, while on-line variant of random forests (RF) [1] learner is used as memory-capable function approximator coupled with Intrinsically Motivated Reinforcement Learning (IMRL) reward function to approximate the policy of *actor* and the value function of *critic*. At this point we would like to mention that M3 Computer Architecture Group at Cornell has proposed a similar work [2] to our current interest. They implement a RL-based memory controller with a different underlying RL implementation.

[1] basically, a POMDP is like an MDP but with observations (o) instead of direct state perception.

3 Controller Architecture

Fig.1 illustrates the general view of our memory controller based on heuristic memory approach for solving non-Markovian cart-pole balancing task and balancing two poles simultaneously. We briefly explain its components.

Past experiences. Sensory control inputs from environment would be stored at the next available empty memory location (chunk), or redundantly at several empty locations. In our memory implementation only the following parameters have to be specified by a designer: 1) The capacity of the memory, 2) A function which extracts features from its stored locations, 3) A predictor which pro-

vides relevant features of the current system state, and 4) A function which provides intrinsic rewards.

Feature predictor. Is utilized to produce associated feature for selective experience. This predictor was designed to predict multiple experiences in different situations. When the selective experience is predicted, the associated feature is converted to feature vector so the controller can handle it.

Features Map. The past experiences are mapped into multidimensional feature space using neighborhood component analysis, based on the Bellman error, or on the temporal difference (TD) error. In general this is done by choosing a set of features which approximate the states S of the system. A function approximator (FA) must map these features into a value function V^{π} for each state in the system. This generalizes learning over similar states and increases the speed of learning, but potentially introduces generalization error as the feature will not represent the state space exactly.

Memory access. The memory access scheduling is formulated as a RL agent whose goal is to learn automatically an optimal memory scheduling policy via interaction with the rest of the system. As can be seen in Fig.1 two scenarios are considered. In Fig.1a all the system parameters are *fully observable*, the agent can estimate V^{π} for each state and use its actions (past experiences). In Fig.1b the system is *partially observable* [3, 4], the agent does not know which state it is in due to sensor limitation, the agent updates its policy parameters directly. Since our system is modeled as non-Markovian (non-MDP) process decision depends on last state-action, and the state transitions $s_{t+1} = \delta(s_t, a_t)$ depend on arbitrary past state where the agent had visited. This transition is expressed by $Pr(s_t|s_{t-1}, a_{t-1}, s'_t, s''_t, \cdots)$, where s_{t-1}, a_{t-1} are the previous state and action, and t', t'' are arbitrary past time.

Learning behaviors from past experience. On each time step, a component of the TD learning algorithm, called the adaptive critic, is used to estimate the expected future reward of retaining various combinations of memory locations. The collection of memory locations show to have the highest accumulated rewards are more likely to be remembered. The amount of occasional intrinsic rewards received, a long with the estimates of the adaptive critic on this time step and on the previous time step, are used to compute the TD error– the change in expected future reward. This error signal also used to train the adaptive critic.

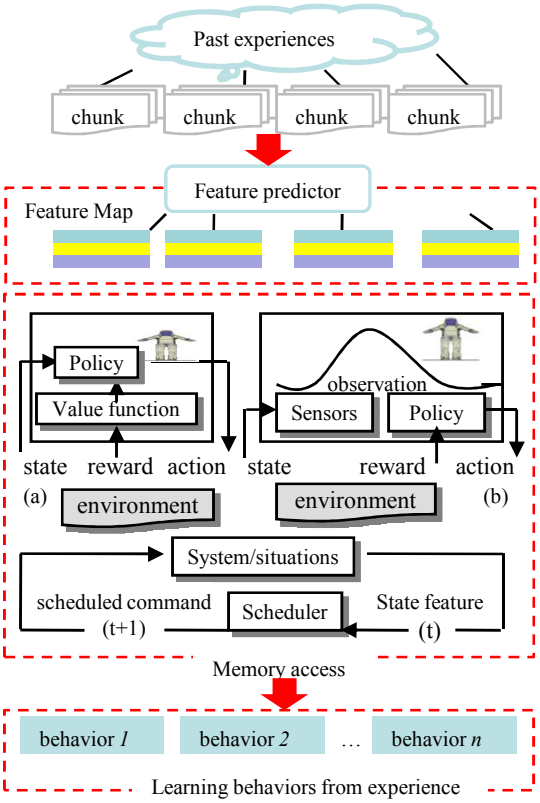

Figure 1: Architecture of self-optimizing memory controller. The controller utilizes associated feature analysis to memorize complete non-Markovian reinforcement task as an action of past experience.

4 Memory Capable FA

4.1 Actor-Critic Learning

Actor-critic (AC), a group of on-policy TD methods, separates the policy and the value function into independent memory structures. The policy structure, or *actor*, is used to decide which action to pick in each state. The estimate value function, or *adaptive critic*, determines whether the actions of the *actor* are to be rewarded or punished. The algorithms use these spare measures of performance to adopt an optimal behavior over time. The adaptive critic maps its current state event onto an estimate of whether it will be rewarded. The mapping is learned from the past experience. If $s+1$ is the situation that follows situation s in time, this expected future reward may be written as:

$$V(s) = \gamma^0 r(s) + \gamma^1 V(s+1) + \cdots + \gamma^n V(s+n) \quad (1)$$

The value of the current situation, $V(s)$, is the sum of all the rewards we will receive over the next n time steps. The rewards on each time step are "discounted" by factor, γ, in the range $[0, 1]$. Equation (1) can be rewritten in a recursive form:

$$V(s) = \gamma^0 r(s) + \gamma^1 V(s+1) = r(s) + \gamma V(s+1) \quad (2)$$

Obviously a value function estimates that fall far from this equality in considered inaccurate, and the error is estimated based on TD error:

$$\delta(s) = (r(s) + \gamma V(s+1) - V(s)) \tag{3}$$

Adopting these methods can save much computation for selecting optimal actions, due to utilizing separate memory for value function and policy.

4.2 AC in non-Markovian Domain

Due to non-Markovian characteristics, the controller infers the state of its environment from a sequence of observation it receives, learns an optimal action by detecting certain past events, that associated with its current perception.
In particular, at time t, the error of the critic is give by

$$E_c(t) = \frac{1}{2}([r(t) + \gamma J(t)] - J(t-1))^2 \tag{4}$$

while the error of the actor is

$$E_a(t) = \frac{1}{2}(J(t)] - R^*)^2 \tag{5}$$

where R^* is the optimal return, which is dependent on the problem definition. The expected return is expressed as the general cost function, $J(t)$, which is to be maximized by the controller. Specifically,

$$J(t) = r(t+1) + \gamma r(t+2) + \gamma^2 r(t+3) + \cdots \tag{6}$$

where $r(t)$ is the immediate reward and γ is the time-discounting factor $0 \leq \gamma \leq 1$.

4.3 RF Memory for Optimal Learning

On-line RF has the characteristics of a simple structure, strong global approximation ability and a quick and easy training [1]. It has been used with TD learning for building a hybrid function approximator [5, 6]. Here, in order to improve learning efficiency and to reduce the demand of storage space and to improve learning efficiency, the on-line RF approximator is structured in a way that both *actor* and *critic* can be embodied in one structure, subsequently, is used to approximate policy function of *actor* and the value function of *critic* simultaneously. That is, the actor and the critic can share the input and the basis functions structure of the RF. Let RF_{Appro} represent a hybrid approximator that combines actor and critic. Given a state $s(t)$ and action $a(t)$, RF_{Appro} is defined such that $RF_{Appro}(s(t), a(t)) = (J(t), a(t+1))$, where $J(t)$ is the estimated value of the given state-action pair, and $a(t+1)$ is the subsequent action to be taken by the controller. At the critic output the error is captured by TD error. However, at the action outputs the error is determined by the gradient of the estimated value $J(t+1)$ w.r.t the action $a(t+1)$ selected by the on-line RF at time t. Specifically,

$$e_a(t) = \alpha \nabla_{a(t+1)J(t+1)}$$
$$= \alpha \left(\frac{\partial J(t+1)}{\partial a_1(t+1)}, \cdots, \frac{\partial J(t+1)}{\partial a_d(t+1)} \right) \tag{7}$$

where α is a scaling constant and d is the choices availabilities at action a. Accumulating the error for each choice of the selected action, the overall actor error is given be:

$$E_a(t) = \frac{1}{2} \left[\sum_{i=1}^{d} e_{ai}^2(t) \right] \tag{8}$$

where $e_{ai}(t)$ is the choice of the action error gradient $e_a(t)$. In finding the gradient of the estimated value $J(t+1)$ w.r.t the previously selected action $a(t+1)$, the direction of change in action, which will improve the expected return at time step $t+1$, is obtained. Thus by incrementally improving actions in this manner, an optimal policy can be achieved. $E(t) = E_c(t) + E_a(t)$ define the reduced error for the entire on-line forest.

5 Experiment and Results

The proposed controller brings a number of preferable properties for learning different behaviors. In this section, we investigate its learning capability through a task of cart-pole balancing problem, designed with non-Markovian settings.

5.1 Related work

The pole balancing algorithm has not been extensively modeled for non-MDP. Although a variation of Value and Policy Search (VAPS) algorithm [7] has been applied to this problem for the POMDP case [8], they have assumed that x and θ are completely observable. NeuroEvolution of Augmenting Topologies [9] and evolutionary computation [10], are another promising approaches where recurrent neural networks are used to solve a harder balancing of two poles of different lengths, in both markovian and non-Markovian settings.

5.2 Non-Markovian Pole Balancing

As illustrated in Fig.2A, Cart-Pole balancing involves a vertical pole with a point-mass at its upper end installed on a cart, with the goal of balancing the pole when the cart moves by applying horizontal forces to the cart, which must not stray too far from its initial position. The state description for the controller consists of four continuous state variables, the angle θ (radial), and the speed of the pole $\acute{\phi} = \delta x / \delta t$ plus the position x and speed

of the cart $\dot{x} = \delta x/\delta t$. The two continuous actions set up for controller training and evaluation were RightForce (RF), (results in pushing the cart to the right), and LeftForce (LF), (results in pushing the cart left). At each time step t, the controller must only observe the θ and then takes appropriate action to balance the pole by learning from the past experience and the intrinsically rewards (Fig.2A). The optimal value function is shown in Fig.2B. A simulated sample run is shown in Fig.3. The controller could keep the pole balanced after about 4000 steps.

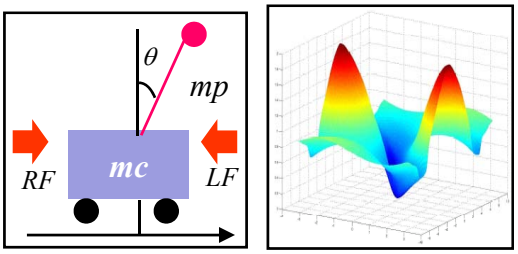

Figure 2: (A) Illustration of the non-Markov Cart-Pole balancing problem, where the angular velocity is not observing by the controller. (B) Optimal value function.

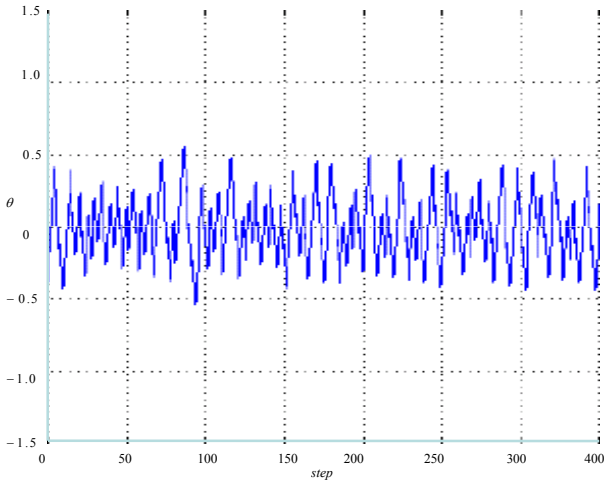

Figure 3: A sample learning for balancing the pole.

5.3 Non-Markovian 2-Pole Balancing

Then we moved to a harder setting of this problem, balancing two poles simultaneously. Each pole has its own position and angular velocity, θ_1 and $\dot{\theta}_1$ for the first pole and θ_2 and $\dot{\theta}_2$ for the second pole respectively. The controller must balance the two poles without velocity information. In order to assist the feasibility of our approach to balance two poles simultaneously we compared with state-of-the-art methods. Table 1 reports the performance of our controller compared with traditional value function-based methods (including SARSA-CMAC, SARSA-CABA, and VAPS) and policy search method (Q-MLP). Value function performance results are reported by [10], who used SARSA implementations by [11]. It shows that our controller takes the minimal evaluations to balance the poles. With regard to CPU time (reported in seconds) we slightly fall short to Q-MLP. However, it interesting to observe that none of the value function approaches could handle this task in within the set of steps due to the memory constraint. The result also indicates that our memory controller stand as a promising method in solving this benchmark more successful than the traditional RL techniques.

Table 1: Comparison of our result for balancing two carts simultaneously with state-of-the-art value function approaches and policy based methods.

	Method	Evaluation	time
V-function	SARSA-CMAC	Time Out	-
	SARSA-CABA	Time Out	-
	VAPS	Time Out	-
Policy	Q-MLP	10,582	153
Memory	Our	8,900	300

6 Conclusions

In this paper we provide evidences that the robot control system will benefit from the inclusion of a self-optimizing memory controller. Our model avoids manual 'hard coding' of behaviors, modeled the memory controller as a RL agent learning from past experience. Results based on non-Markov Cart-Pole balancing indicate that our model can memorize complete non-Markovian sequential tasks and is able to produce behaviors that make the controlled system to behave desirably in the future. One of our future plans is to overcome the limited capacity of memory. In our current design the number of "chunks" that can be used is quite limited. Another future plan will be in designing intelligent mechanism for memory updating, and to experiment with different applications such as visual and speech recognition, and robot navigation.

References

[1] Hassab Elgawi, O.: "Online Random Forests based on CorrFS and CorrBE," *In Proc.IEEE workshop on online classification*, CVPR, pp.1-7, 2008.

[2] Ipek, E., Mutlu, O., Martinez, J.F., and Caruana, R.: "Self-Optimizing Memory Control-

lers: A Reinforcement Learning Approach,". In *Intl. Symp. on Computer Architecture (ISCA)*, pp.39-50, 2008.

[3] Cassandra, A. R, Kaelbling, L. P, and Littman, M. L.: "Acting optimally in partially observable stochastic domains,". *Proc. Int'l. Conf on AAAI*, pp.1023-1028, 1994.

[4] Kaelbling, L, Littman, M and Cassandra, A.: "Planning and acting in partially observable stochastic domains,". *Artificial Intelligence*, 101:99-134, 1998.

[5] Hassab Elgawi, O.: "Architecture of behavior-based Function Approximator for Adaptive Control,". In *Proc. 15th Int'l. Conf on Neural Information Processing* ICONIP, LNCS 5507, pp.104-111, 2008.

[6] Hassab Elgawi, O.: "Random-TD Function Approximator," *Journal of Advanced Computational Intelligence and Intelligent Informatics* (JACIII), 13(2):155-161, 2009.

[7] Meuleau, N., Peshkin, L., Kim, K.-E., and Kaelbling, L. P.: "Learning finite-state controllers for partially observable environments,". In *Proc of the 15th Int'l Conf on Uncertainty in Artificial Intelligence*, pp.427-436, 1999.

[8] Peshkin, L., Meuleau, N., and Kaelbling, L. P.: "Learning policies with external memory,". In *Proc. of the 16th Int'l Conf on Machine Learning*, pp.307-314, I. Bratko and S. Dzeroski, Eds, 1999.

[9] Kenneth, O. S.: "Efficient evolution of neural networks through complexification,". Ph.D. Thesis; Department of Computer Sciences, The University of Texas at Austin. Technical Report AI-TR-04-314, 2004.

[10] Gomez. F.: "Robust non-linear control through neuroevolution,". Ph.D. Thesis; Department of Computer Sciences, The University of Texas at Austin. Technical Report AI-TR-03-303, 2004.

[11] Santamaria, J. C., Sutton, R. S., and Ram, A.: "Experiments with reinforcement learning in problems with continuous state and action spaces,". *Adaptive Behavior*, 6(2):163-218, 1998.

Appendix A: Pole-balancing learning parameters

Below are the equations and parameters used for cart-pole balancing experiments [10]

.1 Pole-balancing equations

The equations of motion for N unjoined poles balanced on a single cart are

$$\ddot{x} = \frac{F - \mu_c sgn(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i}$$

$$\ddot{\theta}_i = -\frac{3}{4l_i}(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}),$$

where $\tilde{F}_i$ is the effective force from the i^{th} pole on the cart,

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i (\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\sin\theta_i),$$

and $\tilde{m}_i$ is the effective mass of the i^{th} pole,

$$\tilde{m}_i = m_i(1 - \frac{3}{4}\cos^2\theta_i).$$

.2 Pole-balancing learning parameters

Table 2: Parameters for the single pole & double pole problem.

Parameters for the single pole		
Sym	Description	Value
x	Position of cart on track	$[-2.4, 2.4]$m
θ	Angle of pole from vertical	$[-12, 12]$deg
F	Force applied to cart	-10.10N
l	Half length of pole	0.5m
M	Mass of cart	1.0kg
m	Mass of pole	0.1kg
Parameters for double pole		Value
Sym	Description	Value
x	Position of cart on track	$[-2.4, 2.4]$m
θ	Angle of pole from vertical	$[-36, 36]$deg
F	Force applied to cart	-10.10N
l_i	Half length of i^{th} pole	$l_1 = 0.5$m
		$l_2 = 0.05$m
M	Mass of cart	1.0kg
m_i	Mass of i^{th} pole	$m_1 = 0.1$kg
		$m_2 = 0.01$kg
μ_c	friction coef on cart on track	0.0005
μ_p	friction coef if i^{th} pole's hinge	0.0005