6

Reinforcement-based Robotic Memory Controller

Hassab Elgawi Osman
Tokyo
Japan

1. Introduction

Neuroscientists believe that living beings solve the daily life activities, making decisions and hence adapt to newly situations by learning from past experiences. Learning from experience implies that each event is learnt through features (i.e. sensory control inputs) analysis that aimed to specify and then recall more important features for each event or situation.

In robot learning, several works seem to suggest that the transition to the current reinforcement learning (RL) (1), as a general formalism, does correspond to observable mammal brain functionality, where 'basal ganglia' can be modeled by an *actor-critic* (AC) version of *temporal difference* (TD) learning (2; 3; 4). However, as with the most real-world intelligent learning systems, the arising of 'perceptual aliasing' (also referred to as a problem of 'incomplete perception', or 'hidden state') (5), when the system has to scale up to deal with complex nonlinear search spaces in a non-Markov settings or *Partially Observation Markov Decision Process* (POMDP) domains (6) (see Fig. 1) renders to-date RL methods impracticable, and that they must learn to estimate *value function* v^{π} instead of learning the *policy* π, limiting them mostly for solving only simple learning tasks, raising an interest in heuristic methods that directly and adaptively modifying the learning policy $\pi: S \rightarrow A$ (which maps perceptual state/observation to action) via interaction with the rest of the system (7; 8).

Inclusion of a memory to a simulated robot control system is striking because a memory learning system has the advantage to deal with perceptual aliasing in POMDP, where memoryless policies are often fail to converge (9).

In this paper, a self-optimizing memory controller is designed particularly for solving non-Markovian tasks, which correspond to a great deal of real-life stochastic predictions and control problems (10) (Fig. 2). Rather than holistic search for the whole memory contents the controller adopts associated feature analysis to successively memorize a newly experience (state-action pair) as an action of past experience. e.g., If each past experience was a chunk, the controller finds the best chunk for the current situation for policy exploration. Our aim is not to mimic the neuroanatomical structure of the brain system but to catch its properties, avoids manual 'hard coding' of behaviors. AC learning is used to adaptively tune the control parameters, while an on-line variant of decision-tree ensemble learner (11; 12) is used as memory-capable function approximator coupled with Intrinsically Motivated Reinforcement Learning (IMRL) reward function (13; 14; 15; 16) to approximate the policy of

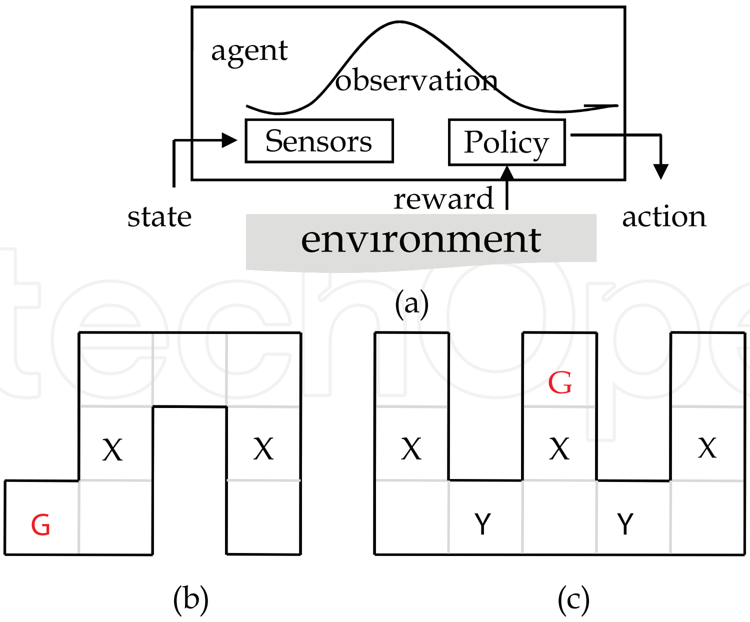


Fig. 1. POMDP and Perceptual aliasing. RL agent is connected to its world via perception state S and action A. In (a) a partially observable world, in which the agent does not know which state it is in due to sensor limitations; for the value function v^{π}, the agent updates its policy parameters directly. In (b) and (c) two maze domains. States indicated with the same letter (X or Y) are perceptually aliased because the agent is sensed only wall configuration.

the *actor* and the value function of the *critic*. Section 2 briefly highlights on POMDP settings. A description with comprehensive illustration of the proposed memory controller will be given in Section 3. Then Section 4 highlights a comparison of conventional memory controller and the self-optimizing memory controller. Section 5 shows the implementation of decision-tree ensemble as memory-capable function approximator for both critic and policy. Some experimental results are presented in Section 6 as promising examples. It includes the non-Markovian cart-pole balancing tasks. The results show that our controller is able to memorize complete non-Markovian sequential tasks and develop complex behaviors such as balancing two poles simultaneously.

2. A non-Markovian and perceptual aliasing

First we present the formal setting of POMDP and then highlight on related approaches tacking perceptual aliasing.

2.1 POMDP formal setting

The formal setting of POMDP is $\mathcal{P} = \langle \mathcal{M},\mathcal{O},\mathcal{Z} \rangle$ consist of:

1. An MDP of a tuple $\mathcal{M}=\langle S,A,T,R \rangle$ where S is the space of possible *states* of the environment, A is a set of *actions* available to the agent (or control input), $P : S \times A \times S \rightarrow [0,1]$ defines a conditional probability distribution over *state transitions* given an action, and $R : S \times A \rightarrow R$ is a *reward function* (payoff) assigning a reward for an action,
2. A set of possible observations $\mathcal{O}$, where $\mathcal{O}$ could constitute either a set of discrete observations or a set of real-value,

3. $\mathcal{Z}$, a probability density mapping state-observation combinations $S \times \mathcal{O}$ to a probability distribution, or in the case of discrete observations combinations $S \times \mathcal{O}$ to probabilities. In other words, $Z(s, o)$ yields the probability to observing o in state s. So basically, a POMDP is like an MDP but with observations instead of direct state perception.

If a world model is available to the controller, it can easily calculate and update a *belief vector* $\overrightarrow{b_t} = \langle b_t(s_1), b_t(s_2), \cdots, b_t(s_N) \rangle$ over 'hidden states' at every time step t by taking into a account the history trace $h = o_1, o_2, \ldots, o_{t-1}, o_t$.

2.2 Perceptual aliasing

It is important to note that in several literatures, perceptual aliasing is wrongly defined as the problem of having an uncomplete instance, whereas this paper defines it as a problem related to having different states that may look similar but are related to different responses. Uncomplete instances may provoke perceptual aliasing, but they are not the same. Although the solely work in this paper is focused on POMDP, we briefly highlight on related approaches, in order to decipher the ambiguities between POMDP and perceptual aliasing:

- *Hidden Markov Models* (HMMs): are indeed applied to the more general problem of perceptual aliasing. In HMM it is accepted that we do not have control over the state transitions, whereas POMDP assume that we do. Hence, POMDP are more related to incomplete perception than to perceptual aliasing. HMMs have been thoroughly applied to robotic behavior synthesis, see, for example (18).
- *Memory-based system*: 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.

  For an early practice attempts with other alternative POMDP approaches, e.g., the 'model-based approach or belief-based approach', and the 'heuristic method with a world model' within TD reinforcement learning domain, see (23; 24).
- There is a large body of work on behavior learning both supervisedly and unsupervisedly using fuzzy logic, Artificial Neural Networks (ANN) and/or Case Based Reasoning (CBR). Some of them do not establish rules and, specifically, CBR uses memory as its key learning tool. This, too, has been used in robotics in loosely defined navigation problems. See, for example (19)

3. Self-optimizing controller architecture

One departing approach from manual 'hard coding' of behaviors is to let the controller build its own internal 'behavior model'–'on-the-fly' by learning from past experience. Fig. 2 illustrates the general view of our memory controller based on heuristic memory approach. We briefly explain its components. It is worth noted that in our implementation only the the capacity of the memory and reward function have be specified by a designer, the controller is self-optimized in a sense that we do not analyzing a domain *a priori*, instead we add an initially suboptimal model, which is optimized through learning[1].

[1] At this point we would like to mention that M3 Computer Architecture Group at Cornell has proposed a similar work (17) to our current interest. They implement a RL-based memory controller with a different underlying RL implementation, we inspired by them in some parts.

Past experiences. Sensory control inputs from environment would be stored at the next available empty memory location (chunk), or randomly at several empty locations.

Feature predictor. Is utilized to produce associated features for each selective experience. This predictor was designed to predict multiple experiences in different situations. When the selective experience is predicted, the associated features are 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 (NCA) (20; 21), 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 V^{π} for each state in the system. This generalizes learning over similar states and more likely to increase learning speed, 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. A similar architecture that exploits heterogeneous learning modules simultaneously has been proposed (22). As can be seen in the middle of Fig. 2 two scenarios are considered. In (a) all the system parameters are *fully observable*, the agent can estimate v^{π} for each state and use its actions (e.g., past experiences). The agent's behavior, B, takes actions that tend to increase the long-run sum of values of the *reinforcement signal*, typically [0,1]. In (b) the system is *partially observable* as described in Fig. 1. Since our system is modeled as POMDP decision depends on last observation-action, and the observation transitions $s_{t+1} = \delta(s_t, a_t)$ depend on randomly past perceptual state. This transition is expressed by $Pr(s_t \mid s_{t-1}, a_{t-1}, s'_t, s''_t, \cdots)$, 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 t, an *adaptive critic* (that is a component of the TD learning), is used to estimate future values of the *reinforcement signal* of retaining different memory locations, which represents the agent's behavior, B in choosing actions. The combinations of memory locations show to have the highest accumulated signals are more likely to be remembered. TD error–the change in expected future signal is computed based on the amount of occasional intrinsic reinforcement signal received, a long with the estimates of the adaptive critic.

4. Non-Markovian memory controller

4.1 Conventional memory controller

Conventional manually designed memory controller suffers two major limitations in regard with scheduling process and generalization capacity. First, it can not anticipate the long-term planning of its scheduling decisions. Second, it lacks learning ability, as it can not generalize and use the experience obtained through scheduling decisions made in the past to act successfully in new system states. This rigidity and lack of adaptivity can lead to severe performance degradation in many applications, raising interest in self-optimizing memory controller with generalization capacity.

4.2 Self-optimizing memory controller

The proposed self-optimizing memory controller is a fully-parallel maximum-likelihood search engine for recalling the most relevant features in the memory of past. The memory

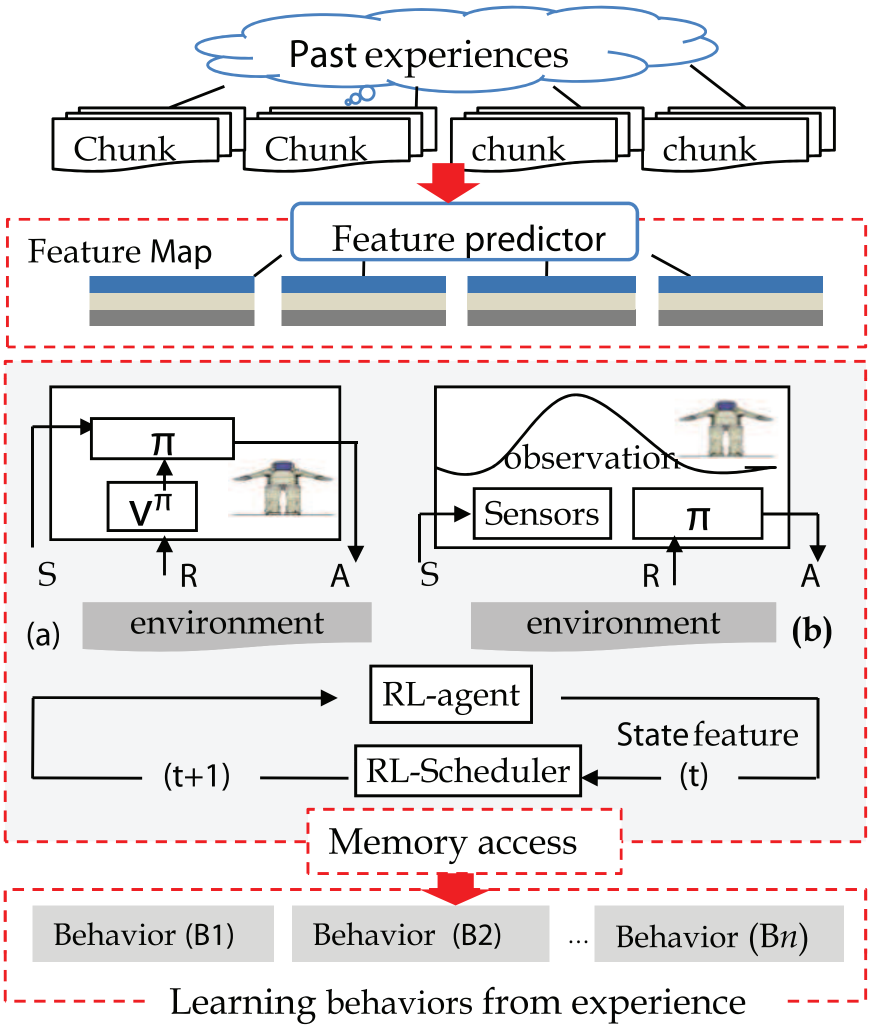


Fig. 2. Architecture of self-optimizing memory controller. The controller utilizes associated feature analysis to memorize complete non-Markovian reinforcement task as an action of past experience. The controller can acquired behaviors such as controlling objects, displays long-term planning and generalization capacity.

controller considers the long-term planning of each available action. Unlike conventional memory controllers, self-optimizing memory controller has the following capabilities: 1) Utilizes experience learnt in previous system states to make good scheduling decisions in new, previously unobserved states, 2) Adapts to the time-variant system in which the state transition function (or probability) is permitted to gradually change through time, and 3) Anticipates the long-term consequences of its scheduling decisions, and continuously optimizes its scheduling policy based on this anticipation.

No key words or pre-determined specified memory locations would be given for the stored experiences. Rather a parallel search for the memory contents would take place to recall the previously stored experience that correlates with the current newly experience. The controller handle the following tasks: (1) relate states and actions with the occasional reward for long planning, (2) take the action that is estimated to provide the highest reward value at a given state, and (3) continuously update long-term reward values associated with state-action pairs, based on IMRL.

5. Memory-capable function approximation

5.1 Actor-critic learning

Actor-critic (AC), a group of on-policy TD methods, separates the π and the v^{π} into independent memory structures. The π structure, or *actor*, is used to decide which action to pick in each state. The estimate of v^{π}, 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:

$$V(s) = \gamma^0 r(s) + \gamma^1 V(s+1) + \cdots + \gamma^n V(s+n) \tag{1}$$

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:

$$V(s) = \gamma^0 r(s) + \gamma^1 V(s+1) = r(s) + \gamma V(s+1) \tag{2}$$

It should be noted that the equality in Eq. 2 is valid only if n is infinite or the state at n time steps later, $s + n$, is always *a* so-called 'absorbing state.' 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.

5.2 AC in non-Markovian domain

Due to non-Markovian characteristics, the controller infers the state of its environment from a sequence of observations 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 given 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 utility 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 $0 \le \gamma \le 1$.

5.3 Decision-tree ensemble memory for optimal learning

On-line decision-tree ensemble learner has the characteristics of a simple structure, strong global approximation ability and a quick and easy training (11; 12). It has been used with TD learning for building a hybrid function approximator (26; 27). Here, in order to improve learning efficiency and to reduce the demand of storage space and to improve learning efficiency, the on-line ecision-tree ensemble approximator is structured in a way that both *actor* and *critic* can be embodied in one structure, subsequently, is used to approximate π of the *actor* and the v^{π} of the *critic* simultaneously. That is, the actor and the critic can share the input and the basis functions structure of the RF. Let DT_{Appro} represents a hybrid approximator that combines actor and critic. Given a state $s(t)$ and action $a(t)$, DT_{Appro} is defined such that $DT_{Appro}(s(t), a(t)) = (J(t), a(t+1))$, 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,

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

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. $E(t) = E_c(t) + E_a(t)$ defines the reduced error for the entire on-line appriximator.

6. Experiment and results

As discussed in previous sections, 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.

6.1 Related work

Modeling the pole balancing algorithm for POMDP has received much interest in the field on control and artificial intelligence. Although a variation of Value and Policy Search (VAPS) algorithm (28) has been applied to this problem for the POMDP case (29), they have assumed that the position of cart on track x and the angle of pole from vertical θ are completely observable. NeuroEvolution of Augmenting Topologies (30) and evolutionary computation (31), 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.

6.2 Non-Markovian Cart Pole balancing

As illustrated in Fig. 3A, 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 $\dot{\theta} = \delta x / \delta t$ plus the position x and speed of the cart $\acute{x} = \delta x/ \delta t,$ (see Appendix A.1 for the equations of motion and Appendix A.2 for parameters used as reported by (31)). 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 (that is, the controller is not observing the velocities $(\dot{x},\dot{\theta})$) and then takes appropriate action to balance the pole by learning from the past experience and the intrinsically rewards. The optimal value function is shown in Fig. 3B. A simulated sample run is shown in Fig. 4. The controller could keep the pole balanced after about 4000 steps.

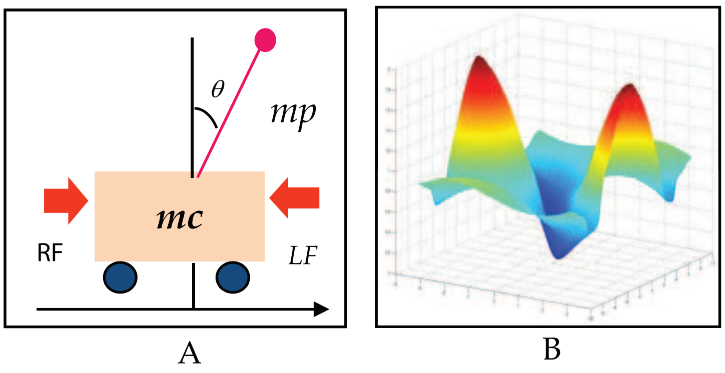


Fig. 3. (A) Illustration of the non-Markov Cart-Pole balancing problem, where the angular velocity is not observing by the controller. (B) Optimal value function.

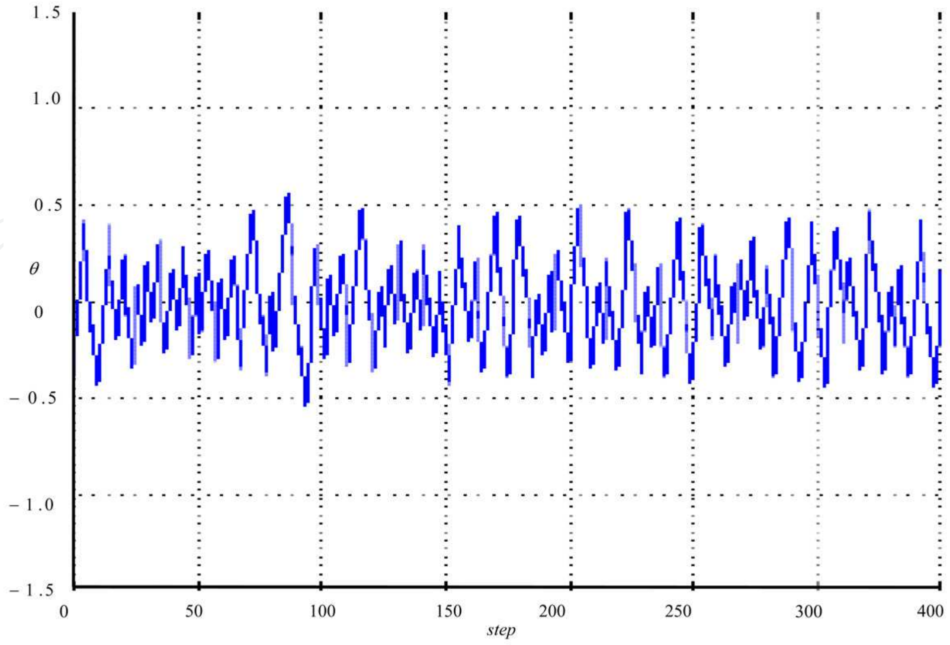


Fig. 4. A sample learning for balancing the pole. It suggests that the method could keep the pole near the top for a long time.

6.3 Non-Markovian Two-Pole balancing

Then we moved to a harder setting of this problem, balancing two poles simultaneously (see Fig. 5). 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. See Appendix A.2 for parameters used as reported by (31).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 other methods.

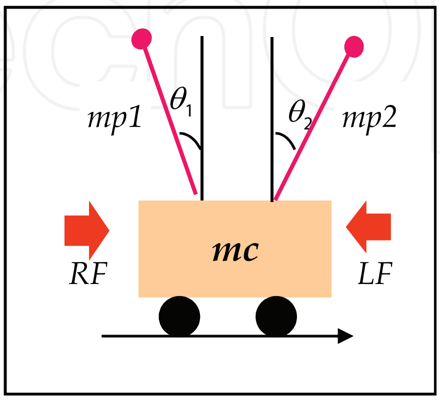


Fig. 5. Illustration of the non-Markov 2-Pole balancing problem. Parameters known are θ_1 and θ_2. The controller must balance the two poles without observing $\dot{\theta}_1$ and $\dot{\theta}_2$.

Table 1 reports the performance of our controller compared with traditional value function based methods (See Appendix B.1 for parameters used) (including SARSA-CABA (See Appendix B.2, SARSA-CMAC (See Appendix B.3, which are reported by (31), who used SARSA implementations by (32) and VAPS (See Appendix B.4) and policy search method (including Q-MLP (See Appendix B.5, as implementation of (31)). Table 1 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 (e.g., 100,000 time steps, which is equal to over 30 minutes in simulated time) 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.

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

Table 1. Comparison of our result for balancing two-pole simultaneously with other value function approaches and policy based methods. 'Evaluation' indicates the total time steps for the method to be able to keep the poles near the top for a long time.

7. Conclusions

This paper proposes an architecture which avoids manual 'hard coding' of behaviors, where an RL agent uses an adaptive memory process to create its own memory and thereby perform better in partially observable domains. The algorithm uses neighborhood component analysis (NCA) to determine feature vectors for system states. Decision-trees ensemble is used to create features which are useful in predicting the state of the system (i.e. building some sort of forward model). Chunks are used with a feature predictor to get features. These features are then used as the input features to learn a policy. 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 automate the capacity of memory in order to accommodate more complex tasks. In our current design the number of chunks that can be used is fixed. Another future plan will be in designing intelligent mechanism for memory updating, and to experiment with real world applications.

8. References

[1] Sutton, R., Barto, A. (1998) "Reinforcement Learning: An introduction,". *Cambring, MA: MIT Press*.

[2] Barto A. (1995) "Adaptive critics and the basal ganglia,". In *Models of Information Processing in the Basal Ganglia*, pp.215-232. Cambridge, MA: MIT Press.

[3] Suri, R.E., Schultz, W. (1999) "A neural network model with dopamine-like reinforcement signal that learns a spatial delayed response task,". *Neuroscience* 91(3):871-890.

[4] Suri, R., Schultz, W. (2001) "Temporal difference model reproduces anticipatory neural activity,". *Neural Computation* 13:841-862.

[5] Chrisman,L. (1992) "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach,". *Proc. Int'l. Conf on AAAI*, pp.183-188.

[6] Cassandra, A., Kaelbling, L., Littman, M. (1994) "Acting optimally in partially observable stochastic domains,". *Proc. Int'l. Conf on AAAI*, pp.1023-1028.

[7] Sutton, R., McAllester, D., Singh, S., Mansour, Y. (2000) "Policy gradient methods for reinforcement learning with function approximation,". *Advances in Neural Information Processing Systems* 12, pp. 1057-1063. MIT Press.

[8] Aberdeen, D., Baxter, J. (2002) "Scalable Internal-State Policy-Gradient Methods for POMDPs,". In *Proc. of the 19th Int'l Conf. on Machine Learning* 12, pp.3-10. Morgan Kaufmann Publishers Inc.

[9] Tsitsiklis, J., Van Roy, B. (1996) "Featured-based methods for large scale dynamic programming,". *Machine Learning* 22:59-94.

[10] Hassab Elgawi, O. (2009) "RL-Based Memory Controller for Scalable Autonomous Systems," In Proc. of *16th Int'l. Conf on Neural Information Processing*, ICONIP, Part II, LNCS 5864, pp.83-92, 2009.

[11] Basak, J. (2004) "Online adaptive decision trees: Pattern classification and function approximation,". *Neural Comput* 18:2062-2101.

[12] Hassab Elgawi, O. (2008) "Online Random Forests based on CorrFS and CorrBE," In Proc. of *Conf on Computer Vision and Pattern Recognition Workshop*, CVPR, pp.1-7.

[13] Singh, S.; Barto, A., Chentanez, N. (2005) "Intrinsically motivated reinforcement learning," In Proc. of *Advances in Neural Information Processing Systems*, NIPS, 17, MIT Press, 2005, pp.1281-1288.

[14] Singh, S., Lewis, R., Barto, A., Chentanez, N. (2009) "Where do rewards come from?" In Proc. of the *Annual Conf. of the Cognitive Science Society*, pp.2601-2606.

[15] Oudeyer, P.-Y., Kaplan, F., Hafner, V. (2007) “Intrinsic Motivation Systems for Autonomous Mental Development,” *IEEE Transactions on Evolutionary Computation*, 11 pp.265-286.

[16] Uchibe, E., Doya, K. (2008) “Finding intrinsic rewards by embodied evolution and constrained reinforcement learning,” *Neural Networks*, 21, pp.1447-1455.

[17] Ipek, E., Mutlu, O., Martinez, J., Caruana, R. (2008) “Self-Optimizing Memory Controllers: A Reinforcement Learning Approach,”. In *Intl. Symp. on Computer Architecture (ISCA)*, pp.39-50.

[18] Maria F., Malik G., Guillaume I., Derek L. (2006) “Robot introspection through learned hidden Markov models,”. *Artificial Intelligence*, 170(2):59-113.

[19] Urdiales, C., Perez, E., V´azquez-Salceda, J., S`anchez-Marr`e. (2006) “A purely reactive navigation scheme for dynamic environments using Case-Based Reasoning,”. *Auton. Robots*, 21(1):65-78.

[20] Goldberger, J., Roweis, S., Hinton, G., Salakhutdinov, R. (2005) “Neighbourhood Components Analysis,”. In *Advances in Neural Information Processing Systems* 17, MIT Press, pp.513-520.

[21] Keller, P. W., Mannor, S., Precup, D. (2006) “Automatic basis function construction for approximate dynamic programming and reinforcement learning,”. In *23rd International Conference on Machine Learning*, pp.449-456.

[22] Uchibe, E., Doya, K. (2006) “Competitive-Cooperative-Concurrent Reinforcement Learning with Importance Sampling,”. In Proc. of the *Eighth Int'l Conf. on Simulation of Adaptive Behavior: From Animals to Animats*, 8, MIT Press,Cambridge, MA, 2004, pp.287- 296.

[23] Jaakkola, T., Singh, S., Jordan, M. (1995) “Reinforcement learning algorithms for partially observable Markov decision,”. In *Advances in Neural Information Processing Systems* 7, pp.345-352, Morgan Kaufmann.

[24] Long-Ji L., Mitchell, T. (1992) “Memory approaches to reinforcement learning in non-Markovian domains,”. Technical Report CMU-CS-92-138, School of Computer Science, Carnegie Mellon University.

[25] Leslie P., Michael L., Anthony R. (1995) “Planning and acting in partially observable stochastic domains,”. *Artificial Intelligence*, 101:99-134.

[26] Hassab Elgawi, O. (2008) “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.

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

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

[29] Peshkin, L., Meuleau, N., Kaelbling, L. (1999) “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.).

[30] Kenneth, O. (2004) “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.

[31] Gomez. F. (2003) "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.

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

[33] Sutton, R. (1996) "Generalization in reinforcement learning: Successful examples using sparse coarse coding,". In *Touretzky et al*, 6(2):163-218.

[34] Albus, J. (1975) "A new approach to manipulator control: The cerebellar model articulation controller (CMAC),". *Journal of Dynamic Systems, Measurement, and Control*, 97(3):220-227.

[35] Baird, L., and Moore, A. (1999) "Gradient descent reinforcement learning,". *Advances in Neural Information Processing Systems*, 12.

[36] Watkins, C., Dayan, P. (1992) "Q-learning,". *Journal of Machine Learning*, 8(3):279-292.

[37] Lin, L.-J., Mitchell, T. (1992) "Memory approaches to reinforcement learning in non-Markovian domains,". Technical Report CMU-CS-92-138, Carnegie Mellon University, School of Computer Science.

[38] Tesauro, G. (1992) "Practical issues in temporal difference learning,". *Journal of Machine Learning*, 8:257-277.

APPENDIX

A. Pole-balancing learning parameters

Below are the equations and parameters used for cart-pole balancing experiments (31)

A.1 Pole-balancing equations

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

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

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

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

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

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

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

A.2 Pole-balancing learning parameters

	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
mc	Mass of cart	1.0kg
mp	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	$l_1 =$ 0.5m $l_2 =$ 0.05m
mc	Mass of cart	1.0kg
mp_i	Mass of i^{th} pole	$mp_1 =$ 0.1kg $mp_2 =$ 0.01kg
μ_c	friction coef on cart on track	0.0005
μ_p	friction coef if i^{th} pole's hinge	0.0005

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

B. Parameters for comparisons in cart pole balancing

Below are the parameters used to obtain the comparison result for SARSA-CABA, SARSA-CMAC, Q-MLP (31), and VAPS (28) in Section 6.3.

B.1 Parameters for value function methods

Parameter	Description
ϵ	greediness of policy
α	learning rate
γ	discount rate
λ	eligibility

Table 3. Parameters for value function methods.

B.2 Parameters used for SARSA-CABA

Sarsa(λ) with Case-Based function approximator (SARSA-CABA (32)): Is a Sarsa method with λ that uses a case-based memory to approximate the Q-function. A newly added state-action pair is calculated by combining the values of the k-nearest neighbors.

B.3 Parameters used for SARSA-CMAC

Sarsa(λ) with CMAC function approximator (SARSA-CMAC (32)): Almost similar to SARSA-CABA except that it uses a Cerebellar Model Articulation Controller (CMAC)(34) instead of a case-based memory to approximate the Q-function. Using this method the state-action space is divided into a set of tilings, each tiling constitutes a set of discrete features. Q-value is calculated as the sum of the value in each tiling.

Parameter	Task	
	$1a$	$1b$
Γd	0.03	0.03
Γ_k^x	0.05	0.05
Γ_k^x	0.1	0.1
ϵ	0.05	0.05
α	0.4	0.1
γ	0.99	0.99
λ	0.4	0.4

Table 4. Parameters used for SARSA-CABA.

B.4 Value and policy search

(VAPS (28)): Is an extension of a method proposed by (35) to policies graph, where stochastic gradient descent is used to search the space. The graph is made of 'nodes' indicating actions and 'arcs' representing observation. Transitions between nodes are initially based on the action associated with node that the agent previously visited, while the environment continue to produce arcs labeled with observations.

Parameter	Task	
	$1a$	$1b$
ϵ	0.05	0.05
α	0.4	0.1
γ	0.9	0.9
λ	0.5	0.3
No. of tilings	45 : 10 based on $x, \dot{x}, \theta_1$ 5 based on x, θ 5 based on $x, \dot{\theta}$ 5 based on $\dot{x}, \dot{\theta}$ 5 based on x 5 based on $\dot{x}$ 5 based on θ 5 based on $\dot{\theta}$	50 : 10 based on x_t, x_{t-1}, θ_t 10 based on $x, \theta_t, \theta_{t-1}$ 5 based on x_t, θ_t 5 based on x_{t-1}, θ_{t-1} 5 based on x_t 5 based on x_{t-1} 5 based on θ_t 5 based on θ_{t-1}

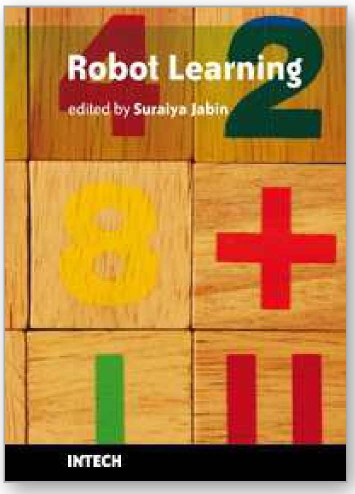
Table 5. Parameters used for SARSA-CMAC.

B.5 Parameters used for Q-MLP

Q-learning with MLP (Q-MLP): This method uses a Multi-Layer Perceptron to map state-action pairs to $Q(s, a)$ that makes it different from standard Q-learning (36). Backpropagation algorithm is used to learn the network values through gradient descent, produces a single Q-value as the output layer. This approach has been thoroughly applied to pole-balancing (37), and backgammon (38).

Parameter	Task		
	$1a$	$1b$	$2a$
ϵ	0.1	0.1	0.05
α	0.4	0.4	0.2
γ	0.9	0.9	0.9
λ	0	0	0

Table 6. Parameters used for Q-LMP.



Robot Learning

Edited by Suraiya Jabin

ISBN 978-953-307-104-6
Hard cover, 150 pages
Publisher Sciyo
Published online 12, August, 2010
Published in print edition August, 2010

Robot Learning is intended for one term advanced Machine Learning courses taken by students from different computer science research disciplines. This text has all the features of a renowned best selling text. It gives a focused introduction to the primary themes in a Robot learning course and demonstrates the relevance and practicality of various Machine Learning algorithms to a wide variety of real-world applications from evolutionary techniques to reinforcement learning, classification, control, uncertainty and many other important fields. Salient features: - Comprehensive coverage of Evolutionary Techniques, Reinforcement Learning and Uncertainty. - Precise mathematical language used without excessive formalism and abstraction. - Included applications demonstrate the utility of the subject in terms of real-world problems. - A separate chapter on Anticipatory-mechanisms-of-human-sensory-motor-coordination and biped locomotion. - Collection of most recent research on Robot Learning.

Osman Hassab Elgawi (2010). Reinforcement-based Robotic Memory Controller, Robot Learning, Suraiya Jabin (Ed.), ISBN: 978-953-307-104-6, InTech, Available from: http://www.intechopen.com/books/robot-learning/reinforcement-based-robotic-memory-controller

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821