Improving Reinforcement Learning with Context Detection

Bruno C. da Silva, Eduardo W. Basso, Filipo S. Perotto, Ana L.C. Bazzan, Paulo M. Engel
Instituto de Informática, UFRGS
Caixal Postal 15064 CEP 91.501-970
Porto Alegre, Brazil
{bcs,ewbasso,fsperotto,bazzan,engel}@inf.ufrgs.br

ABSTRACT

In this paper we propose a method for solving reinforcement learning problems in non-stationary environments. The basic idea is to create and simultaneously update multiple partial models of the environment dynamics. The learning mechanism is based on the detection of context changes, that is, on the detection of significant changes in the dynamics of the environment. Based on this motivation, we propose, formalize and show the efficiency of a method for detecting the current context and the associated model of prediction, as well as a method for updating each of the incrementally built models.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

Reinforcement learning, non-stationary environments, multi-model learning

1. INTRODUCTION

Dealing with non-stationary environments has always been a difficult task for learning algorithms. Specifically, non-stationarity affects standard reinforcement learning (RL) methods in a way that forces them to continuously relearn the policy from scratch. Our goal in this paper is to complement model-based RL algorithms with a method for dealing with a class of non-stationary environments in which the dynamics obeys a set of restrictions.

The non-stationary environments that we deal with consist of those composed by several different stationary dynamics. We call each type of dynamics a *context* and assume that it can only be estimated by observing the transitions and rewards. Our method creates models for these contexts and determines how to switch to the most appropriate model *on-the-fly*. Partial models have been used for this purpose by other authors, such as Choi [1] and Doya [2]. However, their approaches require a fixed number of models, and thus implicitly assume that the approximate number of different environment dynamics is known *a priori*. Since this assumption is not always realistic, our method tries to overcome it by incrementally building new models.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

2. MULTIPLE PARTIAL MODELS

Most RL problems are modeled as Markov Decision Processes (MDPs). MDPs are described by a set of states, $\mathcal{S}$, a set of actions, $\mathcal{A}$, a reward function $R(s,a) \rightarrow \Re$ and a probabilistic state transition function $T(s,a,s') \rightarrow [0,1]$. An experience tuple $\langle s,a,s',r\rangle$ denotes the fact that the agent was in state s, performed action a and ended up in s' with reward r. Given a MDP, the goal is to calculate the optimal policy π^*, which is a mapping from states to actions such that the future reward is maximized. In the next sections we assume that the reader is familiar with Q-Learning, a simple model-free RL algorithm, and with model-based Prioritized Sweeping. For more information, please refer to [3].

When dealing with non-stationary environments, both the model-free and the model-based RL approaches need to continuously relearn everything from scratch, since the policy which was calculated for a given environment is no longer valid after a change in dynamics. This causes a performance drop during the readjustment phase, and also forces the algorithm to relearn policies even for environment dynamics which have been previously experienced. Our hypothesis is that the use of multiple partial models makes the learning system capable of partitioning the knowledge into models, each model automatically assuming for itself the responsibility for "understanding" one kind of environment behavior. To each model, we assign a policy and a trace of prediction error of transitions and rewards.

3. RL WITH CONTEXT DETECTION

We propose that the creation of new models should be controlled by a continuous evaluation of the prediction errors generated by each partial model. In the following subsections we first describe how to learn contexts (i.e, estimate partial models), and then we show how to detect and switch to the most adequate model given a sequence of observations. Our method is called **RL-CD**, or Reinforcement Learning with Context Detection.

The class of non-stationary environments that we deal with is similar to the one studied by Hidden-Mode MDPs researchers [1]. We assume that the following properties hold: **1)** environmental changes are confined to a small number of contexts, which are stationary environments with distinct dynamics; **2)** the current context cannot be directly observed, but can be estimated according to the types of transitions and rewards observed; **3)** environmental context changes are independent of the agent's actions; and **4)** context changes are relatively infrequent.

3.1 Learning contexts

The mechanism for detecting context changes relies on a set of partial models for predicting the environment dynamics. A partial model m contains functions which estimate transition probabilities ($\hat{T}_m$) and rewards ($\hat{R}_m$). Standard model-based RL methods such as Prioritized Sweeping and Dyna can be used to compute the locally optimal policy $\pi_m(s)$.

Given an experience tuple $\varphi \equiv \langle s, a, s', r\rangle$, we update the current partial model m by adjusting its model of transition and rewards by $\Delta^{\hat{T}}_{m,\varphi}$ and $\Delta^{\hat{R}}_{m,\varphi}$, respectively. These adjustments are computed as follows:

$$\Delta^{\hat{T}}_{m,\varphi}(\kappa) = \frac{1}{\mathcal{N}_m(s,a)+1}\Big(\tau^{s'}_{\kappa} - \hat{T}_m(s,a,\kappa)\Big) \quad \forall \kappa \in \mathcal{S}$$

$$\Delta^{\hat{R}}_{m,\varphi} = \frac{1}{\mathcal{N}_m(s,a)+1}\Big(r - \hat{R}_m(s,a)\Big)$$

such that τ is the Kronecker Delta:

$$\tau^{s'}_{\kappa} = \begin{cases} 1, & \kappa = s' \\ 0, & \kappa \neq s' \end{cases}$$

The effect of τ is to update the transition probability $T(s,a,s')$ towards 1 and all other transitions $T(s,a,\kappa)$, for all $\kappa \in \mathcal{S}$, towards zero. The quantity $\mathcal{N}_m(s,a)$ reflects the number of times, in model m, action a was executed in state s. We compute $\mathcal{N}_m$ considering only a truncated (finite) memory of past M experiences:

$$\mathcal{N}_m(s,a) = min\Big(\mathcal{N}_m(s,a) + 1,\ M\Big) \tag{1}$$

A truncated value of $\mathcal{N}$ acts like a learning coefficient for $\hat{T}_m$ and $\hat{R}_m$, causing transitions to be updated faster in the initial observations and slower as the agent experiments more. Having the values for $\Delta^{\hat{T}}_{m,\varphi}$ and $\Delta^{\hat{R}}_{m,\varphi}$, we update the transition probabilities:

$$\hat{T}_m(s,a,\kappa) = \hat{T}_m(s,a,\kappa) + \Delta^{\hat{T}}_{m,\varphi}(\kappa), \quad \forall \kappa \in \mathcal{S} \tag{2}$$

and also the model of expected rewards:

$$\hat{R}_m(s,a) = \hat{R}_m(s,a) + \Delta^{\hat{R}}_{m,\varphi} \tag{3}$$

3.2 Detecting context changes

In order to detect context changes, the system must be able to evaluate how well the current partial model can predict the environment. Thus, an error signal is computed for each partial model. The *instantaneous error* is proportional to a *confidence value*, which reflects the number of times the agent tried an action in a state. Given a model m and an experience tuple $\varphi = \langle s, a, s', r\rangle$, we calculate the instantaneous error $e_{m,\varphi}$ and the confidence $c_m(s,a)$ as follows:

$$c_m(s,a) = \left(\frac{\mathcal{N}_m(s,a)}{M}\right)^2 \tag{4}$$

$$e_{m,\varphi} = c_m(s,a)\Big(\Omega(\Delta^{\hat{R}}_{m,\varphi})^2 + (1-\Omega)\sum_{\kappa\in\mathcal{S}} \Delta^{\hat{T}}_{m,\varphi}(\kappa)^2\Big) \tag{5}$$

where Ω specifies the relative importance of the reward and transition prediction errors for the assessment of the model's quality. Once the instantaneous error has been computed, the trace of prediction error E_m of the partial model is updated:

$$E_m = E_m + \rho\Big(e_{m,\varphi} - E_m\Big) \tag{6}$$

where ρ is the adjustment coefficient for the error.

The error E_m is updated after each iteration for every partial model m, but only the active model is corrected according to equations 2 and 3. A plasticity threshold λ is used to specify until when a partial model should be adjusted. When E_m becomes higher than λ, the predictions made by the model are considered sufficiently different from the real observations. In this case, a context change is detected and the model with lowest error is activated. A new partial model is created when there are no models with trace error smaller than the plasticity. The mechanism starts with only one model and then incrementally creates new partial models as they become necessary. Pseudo-code for RL-CD is presented in algorithm 1.

Algorithm 1 RL-CD algorithm

Let m_{cur} be the currently active partial model.
Let $\mathcal{M}$ be the set of all available models.
1: $m_{cur} \leftarrow newmodel()$
2: $\mathcal{M} \leftarrow \{m_{cur}\}$
3: $s \leftarrow s_0$, where s_0 is any starting state
4: **loop**
5: Let a be the action chosen, considering model m_{cur}
6: Observe next state s' and reward r
7: Update E_m, for all m, according to equation 6
8: $m_{cur} \leftarrow \arg\min_m (E_m)$
9: **if** $E_{m_{cur}} > \lambda$ **then**
10: $m_{cur} \leftarrow newmodel()$
11: $\mathcal{M} \leftarrow \mathcal{M} \cup \{m_{cur}\}$
12: **end if**
13: Update $\hat{T}_{m_{cur}}$ and $\hat{R}_{m_{cur}}$ (equations 2 and 3)
14: $\mathcal{N}_m(s,a) \leftarrow \min(\mathcal{N}_m(s,a)+1, M)$
15: $s \leftarrow s'$
16: **end loop**

The *newmodel*() routine is used to create a new partial model and initializes all estimates and variables to zero, except T_m, initialized with equally probable transitions. The values of parameters M, ρ, Ω and λ must be tunned according to the problem. Formal analytical investigation of these parameters is being developed but detailed discussions are not presented due to lack of space.

4. EMPIRICAL RESULTS

We now present some experiments which evaluate how well our system performs in non-stationary environments. The scenario used consists of a non-stationary toroidal grid of 15x15 cells. The agent is a cat whose goal is to catch a moving ball, which starts in a random column and row and can take one of following behaviors: **1)** moves to the left; **2)** moves to the right; **4)** moves down; or **3)** moves up.

The first experiment performed aims at testing the relative efficiency of classic model-free and model-based algorithms: Q-Learning (QL) and Prioritized Sweeping with Finite Memory (PS-M)[1]. The agent is trained for one of the ball behaviors at a time, until the algorithm converges to the optimal policy. After that, we change the environment dynamics by modifying the ball behavior and measure how fast each algorithm converges. It can be seen in figure 1 that, as the context changes, the time needed to recalculate the optimal policies, both in the model-free and the model-based approaches, is much superior to the convergence time of RL-CD. Even though PS-M performs better than Q-Learning, it still has to reestimate $\hat{T}$ after every context change. Our method, on the other hand, takes only a few steps until realizing that the context has changed. After that, it automatically selects the appropriate partial model for the new dynamics.

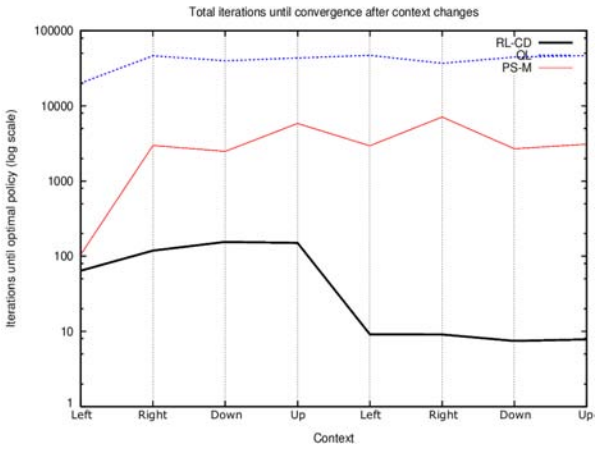

Figure 1: A comparison of convergence times for Q-Learning, PS-M and our approach.

It is also important to measure the average time the cat takes to catch the ball *while* the policy is being learned in a context. In order to implement this experiment, we runned the learning algorithms and changed the ball behavior 8 times. For each one of these, we performed 5 batches of 100 episodes, each episode corresponding to the cat being placed in a random place and running after the ball. We measured the average time of a batch as the average number of iterations needed to catch the ball during all episodes. The results are shown in figure 2.

In figure 2, vertical stripes are colored with different levels of gray, each indicating a different context. Each time the ball behavior changes, the average steps per episode grows (peaks in the graph), indicating that the algorithms take some time to relearn how to behave. During the first 4 contexts, our method is actually very similar to PS-M. However, as soon as the contexts begin to repeat, RL-CD is capable of acting optimally all the time, while the other algorithms present periods of relearning and suboptimal acting.

[1]We verified that standard PS requires exponential time to converge in non-stationary environments, and thus we designed and used **PS-M**, a modified PS with truncated memory instead of maximum likelihood estimation.

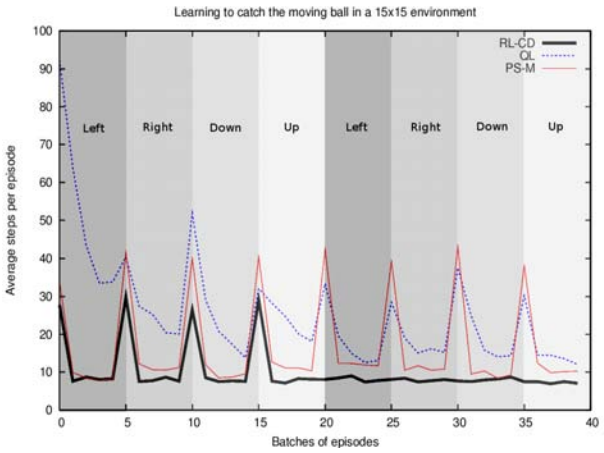

Figure 2: A comparison of performance for Q-Learning, PS-M and our approach.

From the empirical results discussed in this session we can imply that: **1)** our approach converges to the optimal policy much faster than classic model-free and model-based methods in non-stationary environments; and **2)** in our approach the readapting time is very small, which implies that the period of time with suboptimal acting is short.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced and formalized a simple method, called RL-CD, for solving reinforcement learning problems in non-stationary environments. We presented empirical results which show that both the classic model-free and model-based approaches have worse convergence times.

Formal analytical analysis of the RL-CD parameters is being developed, as well as a more detailed study regarding the trade-off between memory requirements and model quality in highly non-stationary environments. We are also studying the possibility of using context detection as a method for dimensionality reduction, since partial models might summarize the impact of a subset of hidden sensors which cause non-stationarity due to partial observations.

6. REFERENCES

[1] S. P. M. Choi, D.-Y. Yeung, and N. L. Zhang. Hidden-mode markov decision processes for nonstationary sequential decision making. In *Sequence Learning - Paradigms, Algorithms, and Applications*, pages 264–287, London, UK, 2001. Springer-Verlag.

[2] K. Doya, K. Samejima, K. ichi Katagiri, and M. Kawato. Multiple model-based reinforcement learning. *Neural Computation*, 14(6):1347–1369, 2002.

[3] L. P. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.