Self-organizing developmental reinforcement learning

Alain Dutech

Self-organizing developmental reinforcement learning

Alain Dutech
LORIA - INRIA
Campus Scientifique - BP 239
54506 Vandoeuvre les Nancy, FRANCE
alain.dutech@loria.fr

June 7, 2012

Abstract

This paper presents a developmental reinforcement learning framework aimed at exploring rich, complex and large sensorimotor spaces. The core of this architecture is made of a function approximator based on a Dynamic Self-Organizing Map (DSOM). The life-long online learning property of the DSOM allows us to take a developmental approach to learning a robotic task: the perception and motor skills of the robot can grow in richness and complexity during learning. This architecture is tested on a robotic task that looks simple but is still challenging for reinforcement learning.

1 Overview

The framework of reinforcement learning [Sutton and Barto, 1998] is particularly attractive because it enables an artificial agent to learn an action plan to solve a problem that may be unfamiliar or uncertain while only using a scalar signal to distinguish between good and bad situations. Learning is not supervised and does not require an expert or an oracle. Moreover, Markov Decision Processes (MDP) [Puterman, 1994] provide a formal background for analyzing theoretical properties of many algorithms, ensuring, for example, the existence of an optimal solution to the learning problem and convergence to this optimal solution.

Although reinforcement learning led to some successful applications, the practical use of reinforcement learning for realistic problems is not easy. Difficulties usually arise from the size of the problems to be solved that can only be described through a very large number of states, making them computationally intractable for reinforcement learning algorithms searching an exact solution. Algorithms providing approximate solutions do exist (approximate value iteration, LSPI, direct policy search through gradient ascent, *etc.*, see Chap. 3,4 of [Sigaud and Buffet, 2010],[Lagoudakis et al., 2002]), but they are far from providing answers to all problems encountered in practice.

We are particularly interested in the use of reinforcement learning in the context of autonomous robotics. Several difficulties reduce the applicability of the learning algorithms. The sensor data are continuous in value, often noisy, expensive to obtain (in time and energy) and very (too) rich in information (*e.g.* in the case of a video stream). Our main concern is for the agent to be able to explore efficiently these rich and complex environments. We propose a *developmental* approach where the agent starts with crude actions and perceptions that gradually get more complex. Thus, starting from a smaller and simpler sensorimotor space, the agent should be able to learn simple tasks and, building over acquired behaviors, learn more complex tasks as its sensorimotor world gets richer. This developmental learning is made possible by using a self-organizing adaptive map architecture for function approximation in a reinforcement learning framework.

This paper first quickly presents the reinforcement learning framework used and original architecture for a developmental approach (Sections 2 and 3). Then, the experimental setting is detailed and results are given (Sections 4 and 5). Section 6 discusses the our work, pointing out some limitations and offering perspectives for future work.

2 Principles and Motivations

Our approach of developmental reinforcement learning is largely based on the *Q-learning* algorithm Watkins [1989] and therefore relies on estimating the optimal value function Q^* in the sensorimotor space $\mathcal{S} \times \mathcal{A}$. The principle of the algorithm is to use each training sample (s_t, a_t, s_{t+1}, r_t) provided at time t through an interaction between the agent and its environment (the agent perceives s_t, chooses action a_t that changes the environment now perceived as s_{t+1}, and gets a reward r_t) to update the estimate of the value function Q^* for the pair (s_t, a_t) as follows:

$$Q^*_{t+1}(s_t, a_t) \longleftarrow Q^*_t(s_t, a_t) + \overbrace{\alpha \left(r + \gamma \max_{a' \in \mathcal{A}} Q^*_t(s_{t+1}, a') - Q^*_t(s_t, a_t) \right)}^{\Delta}, \qquad (1)$$

where α is a learning coefficient, which may depend on s, a and t.

A major limitation to the practical use of reinforcement learning methods is related to the complexity of algorithms. Theoretically, to guarantee the convergence of reinforcement learning algorithms to an optimal solution one requires that each (s, a) pair is visited an infinite number of times. Even with only a very large number of visits, this constraint is one of the main limitations to the practical use of reinforcement learning techniques, and particularly in the context of robotics, and the only way to learn good behavior would be to explore intelligently and efficiently the environment.

The sensorimotor space of a robot is rich, continuous and complicated, furthermore it can be time and energy consuming to get a new training sample, and also risky (like bumping in a wall). As a result, an efficient and complete exploration of the unknown environment is very unlikely. The main idea of our developmental reinforcement learning framework is to ease the exploration by starting with a very small and limited sensorimotor space and gradually increasing it when the robot learned performances improve. Here, this support to the agent mainly takes the form of a gradual increase of perceptions richness and potential actions of the agent as its performance in the learning task is improving.

From this perspective, several problems arise. First, the learning architecture chosen must allow to progressively increase the sensorimotor space of the agent and the difficulty of the task. Second, we want the agent to rely on what he has already learned to learn in a richer environment. This last issue is particularly delicate, it is similar to the problem of knowledge transfer, a research field still wide open [Caruana, 1997]. Within the framework of reinforcement learning, a central element of learning is the estimation of the value function of the sensorimotor space. The problems we have mentioned are related to estimating a function: how to take advantage of the current estimate of a function to extend it to a modified input space (when sensorimotor space is enriched).

3 Architecture for developmental learning

The core of our architecture is an adaptive function approximation to learn the *Q-value* function in a continuous sensorimotor space. The principle of our architecture is depicted in Fig. 1. Perceptions of the robot (we give more details on these perceptions in Section 4), denoted s, feed a Dynamic Self-Organizing Map (DSOM) [Rougier and Boniface, 2011]. The activity of this self-organizing map is the input of a perceptron with one layer of neurons that has as many outputs as possible actions. Thus, the output neuron associated with the action a learns the value function $Q(s,a)$ in a supervised way using the *Q-learning* update equation (1) a linear regression pattern, by modifying its weights ω using: $\omega \leftarrow \omega - \epsilon_L \Delta$. The DSOM map uses unsupervised learning to adapt to the perceptual inputs received.

Although we have not used a more conventional architecture like LSTD or LSQ [Bradtke and Barto, 1996; Lagoudakis et al., 2002], our system is actually quite close. One can consider that the use of a DSOM map projects the perceptions on basis function $\Phi(s)$ that evolve over time, and that the linear regression estimates the value function as a linear combination $w^T\Phi(s)$ of these basis functions. The search for optimal coefficients of this linear combination is made by a stochastic gradient descent and the basis functions being adaptable, our algorithm does not guarantee convergence.

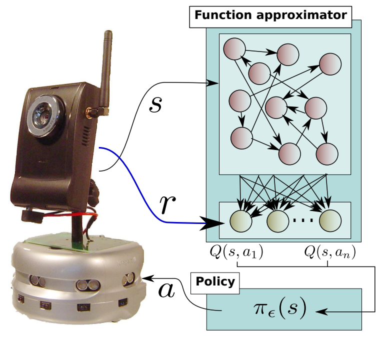

Figure 1: **Learning architecture**. Robot perceptions s are the input of a dynamic self-organizing map. A linear output layer with one neuron for each action$a$ learns $Q(s,a)$.

The neuronal learning architecture is well adapted to the developmental approach that we want to experience. The action space is discrete, its richness is closely tied to its cardinality. Increasing the richness of the action space can be dealt with by adding neurons to the output layer. These new neurons take advantage of the already adapted DSOM map. The learning of the coefficients of the linear combination of a new output neuron can be accelerated by initialising these coefficients using the coefficient of neurons coding "semantically" close actions. In the example that we present below (see Section 4), a new neuron connected with action `turn-right-slowly` will be initialized with the coefficients related to the neuron for the action `turn-right`.

For the continuous state space, the richness comes in part from to the dimension of this space. Increasing the size of the input space is not as simple as for the output space. One solution is to increase the size of all vectors of input weight without changing their projection on the previous perceptual space. Another solution that we would like to test is to provide the architecture with input weight of the maximum dimension of the input space and, while the current perceptual vector has dimension less than this maximum, to artificially increase its size by cloning a few of these values to the extra dimensions. The latter approach seems better adapted to transfer what is learned in small size to larger tasks.

4 Experimental platform

Our experiments are performed on a KheperaIII robot of K-Team company[1]. Although this robot embarks a processor, we chose to deport processing and computation on an external computer, communication between this computer and the robot is done by wifi.

The robot moves using two drive wheels which allow it to turn on the spot. The robot can use discrete actions, such as moving forward or backward a certain distance (on average, as the actions of the robots are noisy) or turn right or left for a given angle. We use discrete actions because we want to explore a framework based on Q-Learning and estimating a value function for continuous actions is still largely an open problem. The robot's action space is a discrete space and its richness is characterized by the number of available actions.

The main sensor we use is a wireless camera placed above the robot which captures a 320x200 bitmap color image, the image is then processed to provide sensory information in a lower dimensional space. Through a logical sensor we call "*retina*", the robot is provided with perceptions made of a vector of dimension $\dim^{\text{per}}$ where each coefficient of the vector, between 0 and 1, is the ratio of a particular hue computed on a vertical strip of the image. The number and positions of the vertical stripes are customizable and can evolve during the experiments. On the example of the right side of Fig. 2, the robot is facing a blue diamond on a red background, the retina is configured to show the ratio of blue vertical stripes - named `B1`, `B2` and `B3` - positioned at the abscissa 80, 160 and 240 of the image. Perceptions of the robot in this case are $[0.42, 0.67, 0.33]$. The state space of the robot is a continuous space with a richness measured by the dimension of this space (that is to say the number of vertical stripes).

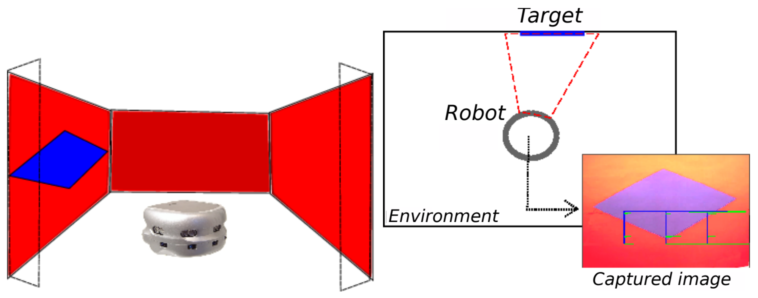

Figure 2: Left: the robot in its environment. Right: the perception given by the `retina` of the robot. For every vertical stripe, positioned at $(80, 160, 240)$, the ratio of blue is computed, here $[0,42; 0,67; 0,33]$.

Given our experimental platform, the tasks we want to teach our robot are navigation tasks in mazes with visual cues. Our idea is to set visual cues in

[1]See `http://www.k-team.com`.

the environment to guide the robot, *e.g.* arrows that tell in which direction to head in order to reach the exit. If the robot needs a fine perception to distinguish and recognize the arrows, it does not need this degree of perception in order to be "attracted" by the highly contrasted symbols. In addition, the robot can learn "satisfactory" behaviors with only crude actions. Then, these behaviors can be refined and precised as actions and perceptions become richer. We believe that this kind of scenario, which can be broken down into sub-tasks of varying complexity, is well suited to the experimentation of the concepts of developmental reinforcement learning.

5 Experimental results

The learning architecture proposed is tested in a very simple testbed. This experiment, which only involves an increase in the richness of the action space of the robot, is only a preliminary to other experiments. Increase in perceptions space and in the complexity of the task are scheduled.

In this experiment, the robot is positioned in a closed environment. The surrounding walls are red and a blue diamond (the target) is set on one of the walls. The robot must choose between 5 discrete actions: `turn-right` and `turn-left` (about 34 degree turn), `stop`, `turn-right-slowly` and `turn-left-slowly` (about 20 degree). These action are noisy by nature because of the imprecision in the robot command. The perceptual space is made of 3 vertical stripes that return the ratio of blue. These stripes, called $B1$, $B2$ and $B3$ are respectively set to the extreme left, at the middle and at the extreme right of the image given by the camera of the robot, their relative angle to the front of the robot are -16, 0 and 16 degree. The robot has to learn to face the target and it receives a reward if it stops while the target is in front of it. If $0.7B1 + B2 + 0.7B3 > 1.4$ and the robot stops, then it gets a reward of $+1$, otherwise the reward is zero. After getting a non-zero reward, of after 12 moves without success, the robot is repositioned randomly.

Learning samples come from real movements and perceptions on the robotic platform but our learning algorithm needs too many iterations and the sample set is not enough for that. As explained later, $10,000$ to $15,000$ iterations are needed for the algorithm to learn. Thus, we use, and more importantly reuse, the same set of $2,000$ samples generated on the robot. Each iteration of the algorithm is done with one sample randomly chosen from the sample set, with respect to the learning scenario (*e.g.*, a sample must use one of the actions currently allowed to the robot).

Three different learning scenarii are tested. In the first one, the robot can only use 3 actions `turn-left`, `turn-left` and `stop`. In the second scenario, the robot can choose among the 5 different actions. In the "*developmental*" scenario, the robot starts learning with 3 actions during N_B iterations then it can also use the 2 other actions, with some optional adaptation of learning parameters.

Results presented were obtained using the following parameters: 64 neurons for the DSOM map, with a "small world" neighborhood and at least on neighbor

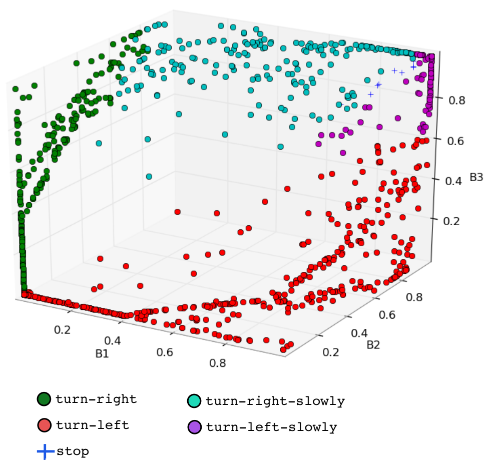

Figure 3: Representation of a policy learned with 5 actions in the sensorimotor space (the coordinate B1, B2 and B3 are the dimension of the perceptual space). The agent stops when facing the target (toward the (1,1,1) vertex) and uses big turns when the target is far from the center (weak values for B1 or B3).

for every neuron ($\text{nb}_{\text{neur}} = 64$, $\text{nb}_{\text{link}} = 1$) ; $\epsilon_D = 0.5$ and $\eta = 1$ are the learning parameters of the DSOM map; $\alpha = 0.1$ and $\gamma = 0.9$ are parameters for the reinforcement learning framework; the exploration policy used is an epsilon-greedy policy, with $\epsilon_\pi = 0.25$. The learning parameter ϵ_L of the linear regressor varies with the experiments (usually between 0.001 and 0.5). This parameter seems to have a more important influence on the quality of the results and on the speed with which performances are reached. These parameters have been set by hand, they give good results, as suggested by the learned policy depicted on Fig. 3.

For every experiment, the quality of learning is evaluated every $1,000$ iterations by testing the greedy policy on a fixed set of starting positions. The set of positions is defined so as to discriminate as much as possible the different policies. These positions are given by the following angles: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 25, 30, 35, 40, 45, 50, 90, 180 and their symmetric. The mean of the reward received from these positions is a measure of the quality of a policy, and thus of the learning. The best hand-made policy using 3 actions has a performance of 0.138 and, with 5 actions, one can reach a performance of 0.39.

Fig. 4 shows the performance reached by the algorithm as a function of the number of iterations when using 3 or 5 actions, with $\epsilon_L = 0.05$ and $\epsilon_L = 0.1$. Each graph is made of 10 learning trajectories. The points are the performances measured and the line are sliding mean of these trajectories. For this set of parameters, as for most of the other settings tested, one can notice a

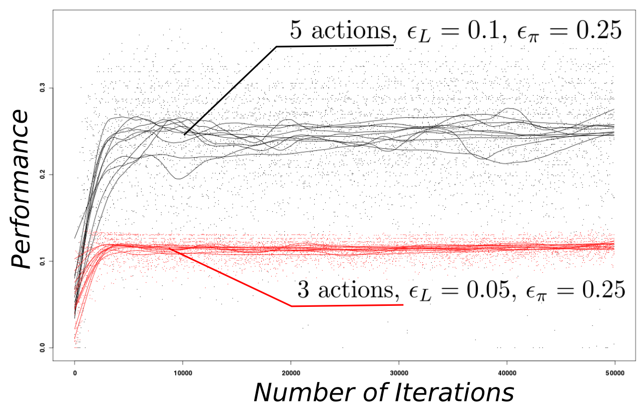

Figure 4: Learning with 3 and 5 actions, points depict the performances every 100 iterations, for 10 different trajectories. Each trajectory is smoothed and represented by a line.

large variability in the performance of the algorithm along a trajectory. The variability can be limited by using very low ϵ_L value.

We want to point out that the learning speed is largely increased with the DSOM based architecture. In a previous work using a multi-layer perceptron and eligibility traces [Sarzyniec et al., 2011], $100,000$ iterations were needed before reaching good performances, whereas here only $5,000$ to $15,000$ are required.

The main focus of the experiments is the exploration of the developmental scenario. Fig. 5 compares the performances of the algorithm when using directly 5 actions with performances obtained with the developmental scenario (3 then 5 actions). The direct learning is quicker but the developmental learning can rather quickly (around $10,000$ iterations) reach better performances. To reach these performances, the learning parameter ϵ_L of the linear regressor must be adapted during the developmental scenario: starting at a value of 0.1, it is reduced to 0.01 when adding the two new actions. If this parameter is kept at 0.1, the performance are reduced (red curve in figure 5, right).

Fig. 5:right shows that, in a developmental scenario, the number if iterations before adding the last 2 actions does not influence the reached performance. For this set of parameters ($\epsilon_L = 0.1$ and $\epsilon_\pi = 0.25$), as for many others, performances tend to decrease after $20,000$ iterations. This fact, combined with the large variability within a learning trajectory, may suggest that the task to learn is not that simple. This may be related to the difficulties of using non-linear approximators in a reinforcement learning framework [Bertsekas and Tsitsiklis, 1996; Coulom, 2002].

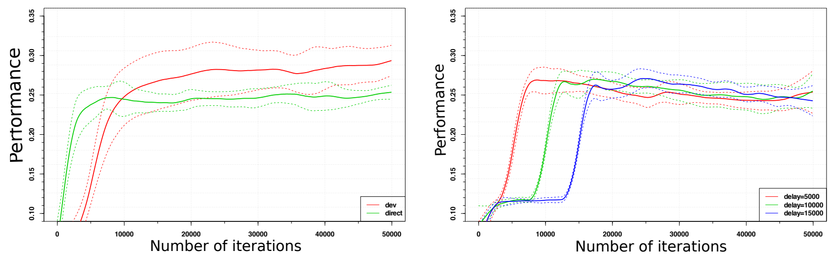

Figure 5: **Developmental learning**. Left: The direct (5 actions) and developmental learning (3 actions, then 2 more after $N_B = 5000$ iterations, ϵ_L goes from 0.1 to 0.01) are compared. The curve show the mean and variance for 10 trajectories. Right: Developmental learning with various delay before using the full action set ($\epsilon_L = 0.1$, $N_B = \{5000, 10000, 15000\}$.

6 Discussion

Results presented here are still preliminary and are part of a work in progress. Many parameters influence the learning algorithm and these parameters should be more systematically explored. Some of these parameters seem to balance each other, like, for example, the number of neurons in the DSOM map and its learning parameters: with a large number of neurons, the DSOM map does not need to be very adaptive and ϵ_D can be low.

The intrinsic properties of DSOM map, like the capacity to adapt continuously to the input while not being oversensible to the distribution of the input, seem particularly interesting when the perceptual space of the agent will change with time. We want to test this, by increasing the perceptions space (add more stripes to the retina) but also by moving these stripes or making them more noisy.

Learning performances are very unstable. In only 100 iterations, the performance of the learned policy can change a lot. Even when the DSOM map is stabilized, which happens very quickly in our experiments as the perceptual space is stable and "small", performances are still unstable, even for low learning parameters of the linear regressor. We would like to study more systematically how the various parameters could be tuned to decrease this variability while preserving both the level of performance and the learning speed, but these criteria are qualitatively assessed right now. We lack an automated procedure for comparing the overall performance (speed, stability, level) of different parameter settings. This kind of automated overall performance evaluation could also be used to guide and control the increase of the richness of the sensorimotor space of the agent. Thus, using original developmental path, it could be possible to link the increase in performance and the "maturation" process of the agent, as done in [Baranes and Oudeyer, 2010] for example.

7 Conclusion

In this paper, we have presented the principles of a developmental reinforcement learning architecture in order to ease the exploration of large and complicated sensorimotor spaces. At the heart of this architecture lies a dynamic self-organizing map that participates in learning an approximation of the value function of the sensorimotor space. Thus, our learning algorithm is a kind of developmental neuro *Q-Learning* algorithm.

Experiments conducted so far on a simple robotic task are quite promising but far from complete, a lot of testing is still needed. When increasing the action space, the learning architecture learns more quickly than previous architectures [Sarzyniec et al., 2011; Dutech, 2011]. Furthermore, even on this simple task, the developmental scenario tested brings better results than a direct approach. This work opens a lot of perspective and future work, in particular a more systematic study of the parameters, the testing of evolving perceptual spaces and more complex tasks.

References

Baranes, A. and Oudeyer, P.-Y. (2010). Maturationally-constrained competence-based intrinsically motivated learning. In *Proc. of IEEE Int. Conference on Development and Learning (ICDL 2010)*, Ann Arbor, Michigan, USA.

Bertsekas, D. and Tsitsiklis, J. (1996). *Neuro-dynamic programming.* Athena Scientific, Belmont, MA.

Bradtke, S. and Barto, A. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57.

Caruana, R. (1997). *Learning to Learn*, chapter Multitask Learning. Kluwer Academic Publishers.

Coulom, R. (2002). *Reinforcement learning using Neural Networks, with Applications to Motor Control.* PhD thesis, Institut National Polytechnique de Grenoble.

Dutech, A. (2011). Dynamic reservoir for developmental reinforcement learning. In *Workhop on Development and Learning in Artificial Neural Networks (DevLeaNN)*, Paris.

Lagoudakis, M., Parr, R., and Littman, M. (2002). Least-squares methods in reinforcement learning for control. In *Proc; of the 2nd Hellenic Conference on Artificial Intelligence (SETN-02)*, number 2308 in Lecture Notes on Artificial Intelligence, pages 249–260.

Puterman, M. (1994). *Markov Decision Processes: discrete stochastic dynamic programming.* John Wiley & Sons, Inc. New York, NY.

Rougier, N. P. and Boniface, Y. (2011). Dynamic Self-Organising Map. *Neurocomputing*, 74(11):1840–1847.

Sarzyniec, L., Buffet, O., and Dutech, A. (2011). Apprentissage par renforcement développemental en robotique autonome. In *Conférence Francophone d'Apprentissage (CAp 2011)*, Chambéry.

Sigaud, O. and Buffet, O., editors (2010). *Markov Decision Processes in Artificial Intelligence.* ISTE - Jonh Wiley & Sons.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning.* Bradford Book, MIT Press, Cambridge, MA.

Watkins, C. (1989). *Learning from delayed rewards.* PhD thesis, King's College of Cambridge, UK.