# Dynamic reservoir for developmental reinforcement learning

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

***Abstract*—We present in this paper an original neural architecture based on a Dynamic Self-Organizing Map (DSOM). In a reservoir computing paradigm, this architecture is used as a function approximation in a reinforcement learning setting where the state $\times$ action space is difficult to handle. 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.**

**As this work is largely in progress, valid and sound results are not yet available.**

## I. Overview

In the field of artificial neural networks, Reservoir Computing [1] refers to the use of a large set of randomly connected neurons so as to create a non-linear transform of the input that can then be read by a linear regressor. Given enough neurons and samples, this kind of architecture can be used to learn an approximation of any function.

In the context of Reinforcement Learning [2], function approximation is crucial when the state $\times$ action space is continuous or discrete but very large. Current works often revolve around the use of least-square linear regression with well chosen base functions of the state to learn an approximation of *Q-Values* [3].

Our work aims at combining an original kind of reservoir computing based function approximation and reinforcement learning. Furthermore, having in mind robotic applications where the task involves a rich, continuous and complex sensory-motor space, we explore a developmental approach to learning. Inspired by observation about the development of young infants, we put forward a solution where the size of the sensory-motor space grows during learning. This should ease the necessary exploration of this sensory-motor space by the agent using reinforcement learning. To that end, the "reservoir network" used is in fact a Dynamic Self-Organizing Map (DSOM) **??** that displays life-long online learning properties. This special architecture should be more suited to the constant evolution of the sensory-motor space during learning.

## II. Developmental Reinforcement Learning

The main idea behind our Developmental Reinforcement Learning is to take inspiration from the growing field of Developmental Robotics [4] to improve the efficiency of Reinforcement Learning on problems with a large state $\times$ action space. Thus, we are especially interested in simultaneously growing the state and action spaces, as the performances of the learning agent increase. Our aim is to have the agent learn difficult tasks, requiring rich sensory-motors spaces without the agent being lost in exploring endlessly and randomly its sensory-motor environment.

In this work, we are interested in a robotic task where the richness of the sensory-motor space is characterized by :

- the dimensionality of the continuous state space (perception of the robot),
- the number of the discrete actions available to the robot.

## III. System architectures

The learning system is organized around a *Q-Learning* [5] approach of Reinforcement Learning. Evolving in a continuous state space, the system relies on approximating the *Q-value* at every point of the state $\times$ action space using artificial neural networks. Three configurations are explored.

- **One MLP by action**. As depicted on figure 1, for every distinct action $a$ available to the robot, one multi-layer perceptron, fed with perceptive information $s$, learns $Q(s, a)$. When the state space increases, more input neurons are added to the MLPs. When the number of action increases, more MLPs are added to the system.
- **A global MLP**. The system has only one MLP with one output neuron for every distinct action available to the robot. When the state space increases, more input neurons are added to the MLP. When the number of action increases, more output neurons are added to the MLP.
- **DSOM and linear perceptron**. The perceptive state $s$ of the robot is fed to a Self-Organizing Map (DSOM) [6]. A linear one-layer perceptron, with as many output as the number of actions, fed with the activity of the DSOM, learns the *Q-values* $Q(s, a)$ in a supervised way. Figure 2 illustrates this original architecture.

The learning framework for configurations (1) and (2) is the following. First, a set of real world transitions[1] is acquired by the robot using an exploratory policy where actions are chosen using a uniform probability distribution. This set of transitions is played and re-played to the MLPs so as to learn the *Q-value* on the state $\times$ action space. Periodically, the quality of

[1] A transition is the given of a starting state $s$, an action chosen by the robot $a$, the next state perceived by the robot $s'$ and the reward received $r$ that can be null.

the learned *greedy policy* (the policy that selects the action with the maximal *Q-value*) is assessed either visually or on the robot itself by measuring the amount of reward collected in a given number of actions. Along learning, the action or perception space can be increased and the transition set grows accordingly.

Configuration (1) has been studied and described in more detail in [7]. Configuration (2) has been the subject of a recent master thesis [8]. A brief description of the main results obtained is given in section V. The third configuration, still under investigation, will be more thoroughly detailed in the next section.

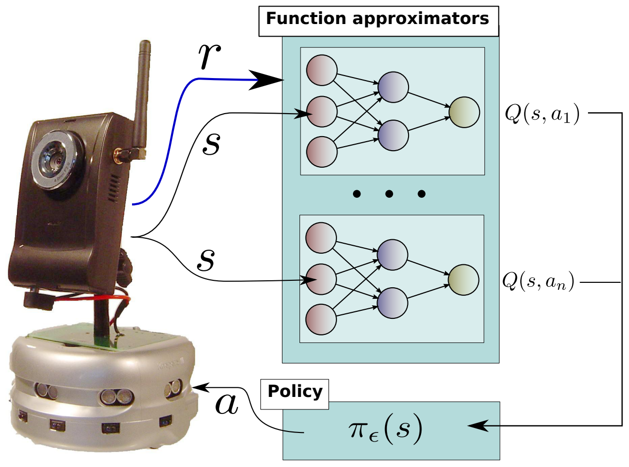

Fig. 1. One MLP for every action of the robot to learn $Q(s,a)$

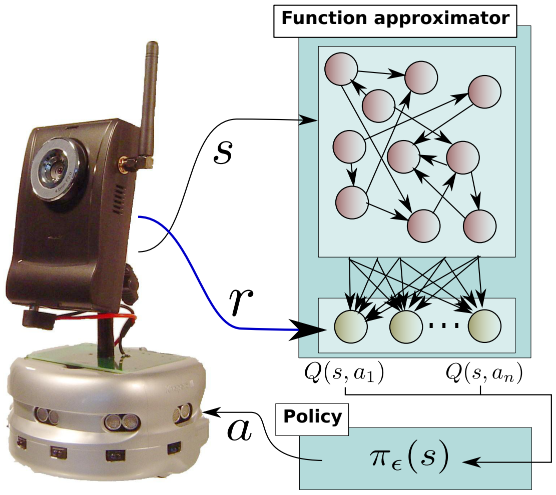

Fig. 2. One DSOM feeds a linear perceptron with one output for each action

## IV. DSOM AND LINEAR PERCEPTRON

A Dynamic Self-Organizing Map (DSOM) is a variation of self-organizing maps [9] that is especially suited to life-long online learning [6]. It learns an unsupervised vector quantization of its input space which can start with few examples and adapt while the number of examples increases. Thus, a DSOM with as many inputs as the maximum number of dimensions of the perceptive space is used in our setting. At start, when the robot only has access to poor/rough perceptions (*i.e.* when perception is a vector of lower dimension that the number of input neurons) this rough perceptions are duplicated on the input neurons. Then, progressively, input neurons are linked to a specific perception dimension.

Three parameters govern the behavior of the DSOM architecture:

- *learning rate* $\epsilon_D$ influences the speed with which neurons are attracted to samples.
- *elasticity* $\mathrm{ela}$ influences the spreading of the neurons on the input space
- *topology of neighbor*. Classically, neurons of the DSOM are arranged in a 2D or 3D way. As nothing tells us that the topology of the perceptive space of the robot is regular, we investigate also non-regular topologies where one neuron can have a given number of neighbors randomly chosen (typically 1, 2 or 3), in a kind of *small world* approach.

The DSOM outputs are the inputs of linear perceptron, with as many outputs as the number of actions in the robot current action set. From the activity of the DSOM (the activity of a DSOM neuron is linked to its closeness to the perception input), the perceptron learns to approximate the *Q-values* using a supervised scheme where the targeted output for the neuron linked to action $a$ is:

$$\Delta Q(\mathrm{DSOM}(s), a) = \alpha[r + \gamma \max_{a'} Q(\mathrm{DSOM}(s'), a')]$$

where $s'$ is the state perceived after executing action $a$ in state $s$ and $r$ is the reward given by the environment. $\alpha$ and $\gamma$ are classical parameters of *Q-Learning*.

When the number of available actions grows, output neurons are added to the linear perceptron. Their weights are initialized with values taken from output neurons linked to actions that are "semantically" close. For example, a new action `turn right slow` could be initialized with the weights of the `turn right` neuron.

## V. FIRST RESULTS

Learning architectures are tested on a real KheperaIII robot set in a simple red colored environment with a blue target. Using a common web cam, the perception of the robot is a vector giving the ratio of a given color for several vertical stripes set at various positions in the image (see figure 3). The robot can turn right or left, at several speeds (turning either 17 or 34 degree), or stay still. It is rewarded when facing approximately a given colored target (a reward of 1.0 is received when the pondered sum of perception vector is bigger than a given threshold, corresponding to the target viewed from an angle of ±15 degrees).

As detailed in [7] and [8], learning the *Q-value* with one MLP for every action or one unique global MLP is possible but not very efficient. With 5 actions, more than 100.00 iterations

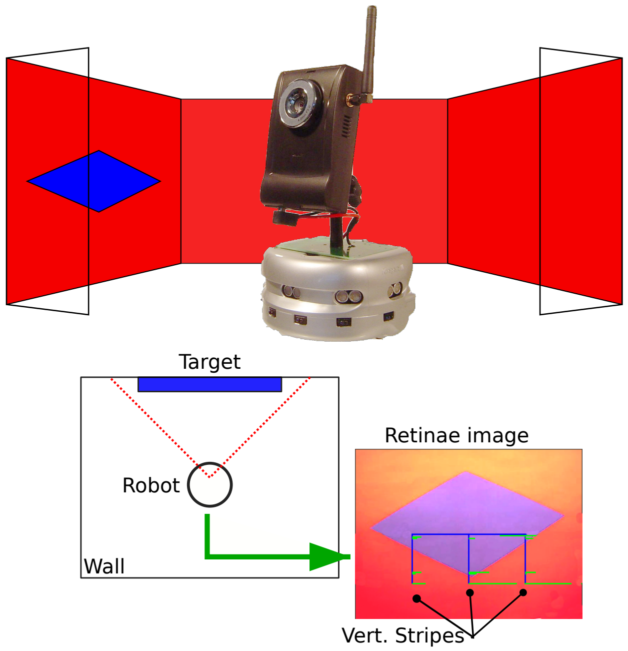

Fig. 3. **Setting**. The robot is placed in a red box with a blue target. Its perception is a $n$-dimension vector indicating, for a given set of $n$ vertical stripes, the ratio of `blue` in the stripes. Here, for example, the perception vector is $(0.28, 0.61, 0.26)$.

are needed (i.e reusing 50 times the set made of 2.000 real transitions) in order to have a proper estimation of the *Q-value*, leading to an efficient policy (like the one depicted in figure 4).

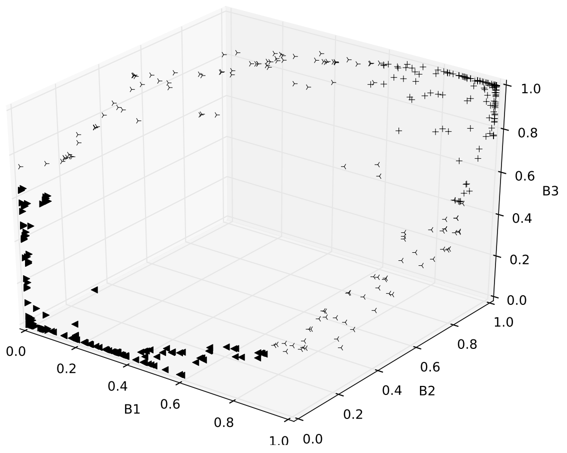

Fig. 4. **Display of a policy**. In the sensory space with 3 dimensions (B1,B2,B3 are the ratio of blue of 3 vertical stripes), the shapes of points tells the best action to do: "+" for `stay`, "◀" for `left`, "<" for `slowLeft`, "▶" for `right` and ">" for `slowRight`.

Many reasons can explain the relatively poor performances of this learning scheme: parameters of the algorithms could be better tuned, intrinsic difficulties of learning *Q-values* in a supervised way as the target of learning is defined from the current learned values, no proof of convergence for non-linear estimation of *Q-values* [10], [11]. Nevertheless, this crude architecture did succeed in showing the interest of developmental reinforcement learning. Using one MLP for every action or a unique global MLP, the robot is able to learn more efficiently if it starts with only 3 actions and then switches to 5 actions than if the robot starts directly with 5 actions (see figures 5 and 6). But, growing the perception space instead did not give interesting results. Furthermore, learning is slow and not compatible with a robot operating in real time. Using one MLP for every action needs more than 150.000 iterations to learn a good policy. With one global MLP, "only" 80.000 iterations are needed.

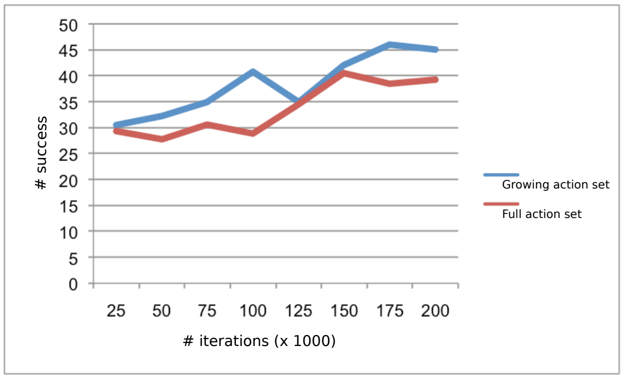

Fig. 5. **Using configuration (1)**. Directly learning with the full set of actions is less efficient than learning with only 3 actions and, then (around iteration 100.000) using 5 actions. This results comes from only one experiment with the robot, as policy evaluation, done every 25.000 iteration, takes about 20 minutes.

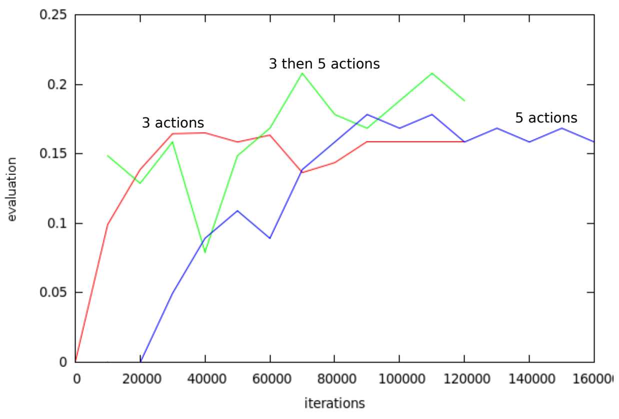

Fig. 6. **Using configuration (2)**. Directly learning with the full set of actions is less efficient than learning with only 3 actions and, then (around iteration 30.000) using 5 actions. This results comes from only one experiment with the robot, as policy evaluation, done every 10.000 iteration, takes about 20 minutes.

Experiments with the DSOM+LP architecture are still underway. So far, experiments are conducted with re-using the samples collected during MLP learning or with a crude simulator. Although results are promising it is a bit too soon to be really assertive.

Our current experiments with the DSOM+LP architectures aim at:

- Checking if the DSOM network can adequately cover the sensor. Our preliminary results show that the discrepancy error of the DSOM decreases nicely and more quickly with a random neighborhood than with a classical grid topology.
- Finding good DSOM parameters, both in term of convergence speed and quality of the coverage of the sensor space. Our preliminary results lead to some good parameters combinations that can learn in about 5.000 iterations.
- Checking whether it is better to let the DSOM learn first and stabilize itself before learning the weights of the readout or if both learning scheme can be used right from the start. It turns out that both nets can learn from the start without loss of performance. Furthermore, learning is very quick as less than 20.000 iterations are needed.
- Finding good DSOM+LIN parameters for the 3 observation, 3 actions setting and for the the 3 observation, 5 actions setting. Currently under way.
- Comparing learning with 3 actions then 5 actions to learning directly with 5 actions. To be done.
- Growing the sensor space, and growing even further the action space. To be done.

## VI. Discussion

Of course, it is difficult to discuss our approach when validated results are still to come. Nevertheless, several improvements or questions to explore are linked to our work.

- **Automated grow of sensory-motor space**. An important question for the future will be to automatically grow the state and action space of the learning agent. This could be "scripted" in advance (similar to human development) or, more appropriately, it could be to linked to the increase in performance of the robot.
- **Selective memory**. Learning speed, although better than with MLPs, could still be improved. One idea is to selectively memorize sequences of actions that lead to reward and replay them when the robot learning module is idle (the robot is executing action for example). The key idea is to replay meaningful sequences as rewarded situations are quite rare, even in the simple setting we have experimented with.
- **Sequence of tasks**. The next big step for the concept of Developmental Reinforcement Learning as we see it will be to go from one task to another. If the next tasks relies on parts of the sensory-motor space that have not been exploited by the current task, learning should not bring any problem. But, if the new task overlaps the old one, on the sensory-motor space or with conflicting rewards, the question is still open.

## References

[1] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, August 2009.

[2] R. Sutton and A. Barto, *Reinforcement Learning*. Bradford Book, MIT Press, Cambridge, MA, 1998.

[3] M. Lagoudakis, R. Parr, and M. Littman, "Least-squares methods in reinforcement learning for control," in *Proc; of the 2nd Hellenic Conference on Artificial Intelligence (SETN-02)*, ser. Lecture Notes on Artificial Intelligence, no. 2308, 2002, pp. 249–260.

[4] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini, "Developmental robotics: a survey," *Connection Science*, vol. 15, no. 4, pp. 151–190, 2003. [Online]. Available: http://dx.doi.org/10.1080/09540090310001655110

[5] C. Watkins, "Learning from delayed rewards." Ph.D. dissertation, King's College of Cambridge, UK., 1989.

[6] N. P. Rougier and Y. Boniface, "Dynamic Self-Organising Map," *Neurocomputing*, vol. 74, no. 11, pp. 1840–1847, 2011. [Online]. Available: http://hal.inria.fr/inria-00495827/en/

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

[8] J. Legrand, "Apprentissage par renforcement développemental," Master's thesis, Université Henri Poincaré, Nancy I, 2011.

[9] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59–69, 1982.

[10] D. Bertsekas and J. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific, Belmont, MA, 1996.

[11] C. Szepesvári, *Algorithms for Reinforcement Learning (Synthesis Lectures on Artificial Intelligence and Machine Learning)*. Morgan and Claypool, 2010.