Inverse Kinematics On-line Learning: a Kernel-Based Policy-Gradient approach

Emmanuel Daucé
UMR 6233
Ecole Centrale de Marseille
Technopôle de Chateau Gombert
Marseille, France
edauce@centrale-marseille.fr

Alain Dutech
LORIA/INRIA - Maia Team
Campus Scientifique, BP 239
Nancy, France
alain.dutech@loria.fr

September 24, 2010

Abstract

In machine learning, "kernel methods" give a consistent framework for applying the perceptron algorithm to non-linear problems. In reinforcement learning, an analog of the perceptron delta-rule can be derived from the "policy-gradient" approach proposed by Williams in 1992 in the framework of stochastic neural networks. Despite its generality and straighforward applicability to continuous command problems, quite few developments of the method had been proposed since. Here we present an account of the use of a kernel transformation of the perception space for the *on-line* learning of a motor command, in the case of eye orientation and multi-joint arm control. We show first that such a setting allows the system to solve non-linear problems, like the log-like resolution of a foveated retina, or the transformation from a cartesian perception space to the "angular" command of the multi-joint arm. More interestingly, the on-line recurrent learning we propose is simple and fully operant in changing environments, and allows for constant improvements of the politics, on the basis of simple and measurables error terms.

1 Overview

Adaptivity to mechanical or perceptual changes is a crucial expected property in artificial devices having to deal with the real environment. It is an important source of robustness, if the user changes or adapts its behaviour (in case of machine/human interfaces, BCI, etc...), if the opponent changes its strategy (prey/predator), or in case of damage (when the device looses a captor and/or an effector).

The reinforcement learning framework offers good hints to solve this family of problem, at the condition that action and movement spaces are considered to be continuous and not discrete 'grid-like' states. In the case we consider further on (inverse kinematics problems), it is not crucial to solve highly instable systems (like inverted "bang-bang" pendulum). We rather try to exploit at best the adaptivity capabilities offered by reinforcement in the case of visually controlled multi-joint arm target reaching. Similarly, it is important to define rewards that exploit available information to the best. In the case of visually guided arm movement for instance, using the known visual error instead of a binary "good/bad" reward can notably improve the speed and accuracy of adaptation.

2 Background

We tackle here the problem of finding a good *continuous* action u on the basis of a vector of features x and a reward r. The policy giving the command u is parametrized by $\boldsymbol{W}$. Finding the good $\boldsymbol{W}$'s when the environment is unknown is a reinforcement learning problem. A bunch of methods exists in the case of discrete state and action spaces, ranging from "temporal differences" methods to "actor/critic" frameworks. The case of continuous states and actions is not such well documented, despite a wide range of potential applications in robotics and control. Typical extensions of the actor-critic approach to a continuous state space imply to use regression methods in order to estimate state values $V(x)$ or transition values $Q(x, u)$ on the basis of observed samples [1]. Those value functions are not expected to be linear, and thus all the limitations and constraints related to nonlinear regression apply (subsampling, overfitting, ...). The problem is even more drastic as the value function is evaluated *on-line* (no resampling, no "bootstrapping", no cross-validation etc...) for it is expected to evolve and improve with time.

Our initial intuition is that the problem of learning continuous action should be more tractable when the regression technique is applied on the parameters of the politics directly, like in the "direct policy gradient" approach proposed by Williams [2] and developped by Bartlett and Baxter [3]. The advantage should come from the fact that, in this setting, the class of the controller is well-known (parametrized linear combination of feature functions) while the class of the value fonction is not as the value function is unknown a priori.

The idea of using a gradient ascent algorithm to directly search the space of parametrized policy (i.e. controller) can be traced back to the family of Reinforce algorithms proposed by Williams [2]. Mostly binary stochastic neurones and episodic algorithms were considered. By casting this concept in the framework of Markov Decision Processes, Baxter and Bartlett added more theoretical and experimental results [3]. Their work was still focussed on binary stochastic neurons but with results on the estimation of the gradient of the value function with respect to the parameters of the controller.

In this paper, we use *scalar* outputs to compute the command. This had quickly been suggested by Williams for gaussian neurons where first and second order moments are differentiable and thus updatable using gradient ascent. Different variants of this linear-gaussian approach can be found in the litterature, for instance in the work of Oyama *et al.* in learning a visual feedback controller for a human arm [4]. In their model, the use of noise is also crucial for learning, but they still need an inverse model of the system in order to properly use the visual feedback signal to compensate for the cases where a too small noise causes inaccuracy in the learning.

Recent developments in the field of policy gradient reinforcement learning have led to several actor-critic algorithm using "natural gradient" [5, 6]. The natural actor-critic of Peters and Schaal, compatible with scalar command, is very efficient although episodic and relying on a sort of bootstraping. Besides, a specific critic term is introduced in order to compute the natural gradient. Bathnagar *et al.* very recent work on natural actor-critic leads also to *online* algorithms but only for discrete and finite commands.

On-line adaptation of a continuous controller remains an open and challenging task in the case, of course, when no model of the environment is given (contrarily to Kalman filtering or optimal feedback control cases), i.e. when learning the good feedback control relies on "raw" input features and rewards only.

3 Principles

We pursue in this paper the exploration of the adaptive properties of linear/stochastic controllers in the on-line/closed-loop case. We recall in this first section the principles supporting our method.

3.1 Linear-gaussian Policy-gradient principles

In the reinforcement learning framework, a parametrized stochastic neurocontroller computes a command $\mathbf{u}$ on the basis of observed state $\mathbf{x}$. The environment, whose dynamics is unknown, is altered by this command and ends up in a new state $\mathbf{x}'$ while the controller receives a scalar reward r linked to the quality of the command. The objective is then to find the "best" neurocontroller, i.e. the set of parameters $\boldsymbol{W}$ which maximizes an objective function J of the rewards. The algorithm is based on a gradient ascent in the space of parameters along the gradient of J according to $\boldsymbol{W}$.

More formally, let us consider that, for a state $\mathbf{x}$, a command $\mathbf{u}$ is chosen according to a density of probability $q(\mathbf{x}, \mathbf{u}, \boldsymbol{W})$. If the reward received at time

t is r_t, the objective function for an horizon of T is:

$$J = \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} r_t\right]. \tag{1}$$

Then, it has been shown [3, 7] that the gradient of J can be rewritten in term of the expectation of the gradient of the logarithm of the policy, as

$$\nabla_{\boldsymbol{W}} J = \mathbb{E}\left[\frac{1}{T}\sum_{k=1}^{T} r(\mathbf{x}_k)\sum_{i=0}^{k-1}\nabla_{\boldsymbol{W}} \log q(\mathbf{x}_i, \mathbf{u}_i, \boldsymbol{W})\right]. \tag{2}$$

In the case of a gaussian multivariate noise, the probability density of chosing a command $\mathbf{u}$ is

$$q(\vec{\mathbf{x}}, \vec{\mathbf{u}}, \boldsymbol{W}) \sim \vec{\mathcal{N}}(\vec{\mathbf{u}} - \boldsymbol{W}.\vec{\mathbf{x}}, \Sigma) \tag{3}$$

$$= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp\left(-\frac{1}{2}(\vec{\mathbf{u}} - \boldsymbol{W}.\vec{\mathbf{x}})^\top \boldsymbol{\Sigma}^{-1}(\vec{\mathbf{u}} - \boldsymbol{W}.\vec{\mathbf{x}})\right). \tag{4}$$

where D is the dimension of the command vector. Then, we have

$$\nabla_{\boldsymbol{W}} \log q(\vec{\mathbf{x}}, \vec{\mathbf{u}}, \boldsymbol{W}) = \nabla_{\boldsymbol{W}}\left[-\frac{1}{2}(\vec{\mathbf{u}} - \boldsymbol{W}.\vec{\mathbf{x}})^\top \boldsymbol{\Sigma}^{-1}(\vec{\mathbf{u}} - \boldsymbol{W}.\vec{\mathbf{x}})\right] \tag{5}$$

$$= \boldsymbol{\Sigma}^{-1}(\vec{\mathbf{u}} - \boldsymbol{W}.\vec{\mathbf{x}})\vec{\mathbf{x}}^\top. \tag{6}$$

3.2 Basic algorithm

The controller works in a closed loop, i.e. learning is based on a sequential exploration of the environment where the new observation is a consequence of the previous commands sent to the environment. As suggested in the on-line stochastic approximation of the gradient used in the OLPOMDP algorithm of [7], our algorithm maintains a "trace" of the gradient so as to estimate its expectation consistently with eq.(2). Using a gradient descent approach, we modify step by step the value of parameters $\boldsymbol{W}$ by a fraction of the estimate of the gradient (where α is the "learning rate"). We end up with the following algorithm. Repeat, for $k \in 1, ..., T$:

1. For the state $\mathbf{x}_k$, compute $\mathbf{u}_k$ according to $q(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{W}_k)$;
2. Read reward r_k (possibly null) and new state $\mathbf{x}_{k+1}$;
3. Update the traces and the weights:
   (a) $z_k = \beta z_k + (1-\beta)\boldsymbol{\Sigma}^{-1}(\mathbf{u}_k - \boldsymbol{W}.\mathbf{x}_k){\mathbf{x}_k}^\top$
   (b) $\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + \alpha r_k z_k$

with $\beta \in [0, 1[$ where $\frac{1}{1-\beta}$ defines the "width" of the trace.

In the "open-loop" setting, no sequence of actions is considered. Instead, many independent inputs are presented to the system and the weights update is based on immediate reward only. The objective function is then limited to an horizon of 1 (i.e., $T = 1$) and there is no trace ($\beta = 0$). As a result, step (3) of the algorithm is then:

3. Update the weights with: $\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + \alpha \mathbf{\Sigma}^{-1}(\mathbf{u}_k - \boldsymbol{W}.\mathbf{x}_k)\mathbf{x}_k{}^\top$.

3.3 Topographic recoding of the input

For its implementation, a policy gradient-based neuro-controller needs only two layers, i.e. one layer for the input features and one layer for the output. The setup thus resembles strongly to the classical linear perceptron regressor and the learning rule itself shares similarity with the widrow-hoff learning rule at the difference that the "real" error is unknown and replaced by the product of the reward and the noise.

Of course, a genuine linear function approximator is bound to a very limited class of problems. Efficient single layer regressors can however be obtained if the input is recoded in a higher dimension space composed of a set of features extracted from the input. Indeed, the dimension of our input space is not very high (limited to the coordinates of targets and points of interest in the surroundings of the controller) and, in order to generalize correctly, the feature space is expected to be very redundant (very "smooth"). This property is respected in the following "topographic coding" for it is expected to reproduce on a map the coordinates of the input. If N is the initial dimension of the input space, consider a projection $g : \mathbb{R}^N \rightarrow \mathbb{R}^P$ doing the topographic coding on a map of dimension P. The stochastic neurocontroller we use in the following uses this recombination with an added noise $\vec{\eta}$. As such, we have $\vec{\mathbf{u}} = \boldsymbol{W}.g(\vec{\mathbf{x}}) + \vec{\eta}$.

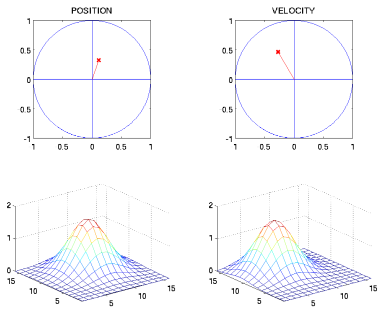

Figure 1: Kernel-based topographic coding.

In our implementation, we use a finite grid of neurons that recombine the input in the form of a radial bump of activity (see figure 1). The sensory system we implement is composed of several 1D or 2D grids of regularly spaced radial basis functions. The real values of $\vec{\mathbf{x}} = (x_1, x_2, ...)$ observed from the environment are bounded (angular position of a joint, coordinates of a point in a visual scene, etc...). This allows to recode every value on one axis of the the RBF grid. Typically, when a visual target appears at subjective coordinates (x_1, x_2), the neurons of the visual layer are set to : $g_i(x_1, x_2) = G_\sigma(x_1 - s_1^i, x_2 - s_2^i)$ where G_σ is a bidimensional Gaussian kernel of radius σ and (s_1^i, s_2^i) is the center of the i^{th} neuron of the grid.

The feature functions of the input we chose lead to a formulation of the controller that is very similar to the kernelized perceptron [8, 9]. If the $\vec{\mathrm{s}}_i$ are the centroïds of the RBF grid used in the input layer, then the activation $f(\vec{\mathbf{x}})$ of the output layer (without the exploration noise) can be written $\langle \boldsymbol{W}, g(\vec{\mathbf{x}})\rangle_{\mathbb{R}^N}$, where $\boldsymbol{W} = (f(\vec{\mathrm{s}}_1), ..., f(\vec{\mathrm{s}}_i), ..., f(\vec{\mathrm{s}}_p))$ and $g(\vec{\mathbf{x}}) = (K(\vec{\mathrm{s}}_1, \vec{\mathbf{x}}), ..., K(\vec{\mathrm{s}}_p, \vec{\mathbf{x}}))$ where K is a gaussian Kernel. It looks as if a kernelized perceptron had been first fed with the p different centroïds so as to use them as some "support"-like vectors for the controller. As such, this could be seen as another online approximation scheme of kernel learning, in the spirit of Kivinen *et al.* [10]. In fact, our weights update fit into the general update rule of Kivinen *et al.* that is

$$f \leftarrow f + \alpha \nabla_f E(J)|_{\vec{\mathbf{x}}_t, \vec{\mathbf{u}}_t} K(\vec{\mathbf{x}}_t, .)$$

Our objectives ar clearly different from those of classical kernel-based learning, but having this in mind could maybe bring new insights or improvements to our model.

4 Applications

4.1 Simple example: open-loop "saccadic" command

In order to validate our approach, let us consider first a simple open-loop non-linear control problem: orienting the eye toward a visual target with a non-isotropic retina. The visual field is the mapping of an external cartesian target position to the subjective retinocentric position (eccentricity of the target relative to the center of the retina). Consider a log-polar representation of the visual input: a target appearing at (ρ, θ) is 'internally' perceived at coordinates $\mathbf{x} = (\frac{\log(1+B\rho)}{\log(1+B)}, \theta)$ where B gives the "distortion" ($B = 4$ in simulation). Suppose the velocity of the target is similarly encoded in log-polar coordinates $\mathbf{y} = (\frac{\log(1+B\dot{\rho})}{\log(1+B)}, \dot{\theta}))$. Then, the motor command that orientates the eye to the visual target can be learned on the basis of visual error.

The aim of visual 'saccadic' movement being to put the target at the center of the retina, we define reward $r = \left(b - \frac{\log(1+B\rho')}{1+B}\right)$ with $b > 0$ where ρ' is the visual error (final distance from the retina at the end of the movement). For the kernel encoding, we use a 16×16 grid for the encoding of x and a 16×16 grid for the encoding of y. The motor command is composed of 4 elementary directions (up, down, left, right). In total, the number of input units is 512 and

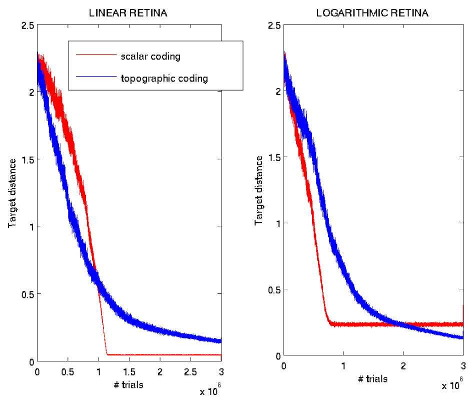

Figure 2: Mean visal error during learning in a saccadic control task. Left : linear retina. Right : Log-Polar retina. Red : scalar input. Blue : topographic input.

the number of output units is 4. The targets appear at random positions and speed (centered random draw, standard deviation 0.3).

The weights vector **W** is initially zero (the output is thus initially purely noisy). The noise used for learning is rather small ($\sigma = 0.03$) in order to allow for a precise control in the end, at the expense of duration (the smaller the noise, the longer the learning session). We apply then the off-line algorithm on 3×10^6 examples taken at random. We compare in figure 2 the learning curves for the genuine and topographic controllers, in the case of linear and log-polar retina. As expected, the genuine controller can not achieve exponential transformation of the input while the topographic controller does, so that its residual error is comparable to the one obtained in the linear retina case.

4.2 On-line adaptivity in a closed-loop setup : visually guided adaptive multi-joint arm control

Consider the task of reaching a target appearing in your visual field by appropriately sending a contraction command to every muscle controlling your arm. A visual check allows you to verify whether you reached or missed the target. Finding and continuously improving the combination of commands leading the arm to the target (whatever the initial target and arm position) belongs to the class of inverse kinematics problem. This control problem must be considered in general as non-linear, non-invertible and noisy. In the non-adaptive case, the solution is computed on the basis of the known characteristics of the captors and the effectors, possibly with a direct and inverse models of the controlled environment. The classical implementation of such a controller implies the use of dynamic programming approach to obtain recursively an estimate of the controller minimizing a certain cost function [11]. In the adaptive setting, one need to improve the control trial by trial on the basis of sensory feedback only. Unexpected changes in the control system must then be compensated on-line.

The task is to control an arm composed of D segments on the basis of visual and proprioceptive signals in a bi-dimensional space. For $d \in \{1, ..., D\}$, each segment is of length ℓ_d. Consider (x_0, y_0) the coordinates of the first joint, then for d in $1, ..., D$: $(x_d, y_d) = (x_{d-1} + \ell_d \cos(\theta_d), y_{d-1} + \ell_d \sin(\theta_d))$ where $\theta_d \in [-\pi, \pi]$ is the angular direction of the d^{th} joint. The end-position of the arm (x_D, y_D) is thus given by the combination of joint angles $(\theta_1, ..., \theta_D)$. Note that for $D \geq 2$, several combinations of angles give the same end-point. The coding of a coordinate in term of segments and joint angles is strongly redundant.

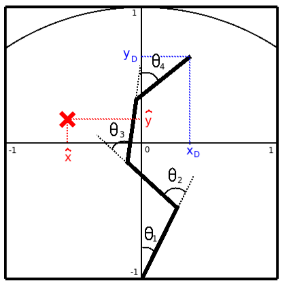

Figure 3: Multi-joint arm control setup

The task of reaching a target appearing at (x, y) with a command on joint angles $(\theta_1, ..., \theta_D)$ is a prototypic example of inverse kinematics problem. By many ways a given target can be reached by the arm, but other constraints must be taken into account in order to have an efficient controller. First of all, a certain "smoothness" is needed in the control : if two targets appear at close-by location, the response of the controller is expected to be quite similar (i.e. the command signal must be continuous if a target moves continuously from position A to position B). Other principles like minimizing the total displacement, second and third order derivatives of the command, taking into account limited joint course, etc... must also be considered in order to obtain 'optimal' control.

In our setting, no direct or indirect model of the system is needed as the reward signal is directly used to evaluate a command and thus guide the learning. As a consequence, this reward signal can be expressed in the sensory space, e.g. the current distance of the arm to the current target, and still allow the controller to be updated in motor space. A classical implementation consists in using the square of the current error $r(t) = -||\mathbf{x}_D(t) - \hat{\mathbf{x}}(t)||_2{}^2$ ("mean-square" error). In a closed-loop setup, the distance criterion is not necessarily the only one to consider. For instance, Todorov and Jordan [11] propose to consider also the current angle velocity $\dot{\theta}(t)$ as a criterion to be minimized for optimizing the displacement, i.e. $r(t) = -||\mathbf{x}_D(t) - \hat{\mathbf{x}}(t)||_2{}^2 - a||\dot{\theta}(t)||_2{}^2$ where a defines the "weight" affected to the second criterion.

Open-loop setup In the open-loop setup, the association between a visual input and a joint angle command can be learned in a very similar way as previously shown, with a reward based on the square of the visual error only. In this simple setup, with 3 or 4 segments, every run leads to a different controller as the space of solutions is very large. Those results are not shown here in reason of limited space.

Closed-loop setup The closed-loop setup offers a more interesting challenge. Consider first the task : the controller owns a camera which allows him to visulize the coordinates x and y of the target in an arena of 2×2 meters (with coordinates in interval $[-1, 1]$). The effector is an arm composed of 4 segments of length 0.5 attached at position $(x_0, y_0) = (0, -1)$ so that a 2 meter radius circle can be reached from that point (see figure 3).

One target is present in the arena at coordinates $(\hat{x}, \hat{y})$. Every 4 seconds, the target jumps to a new position taken at random uniformly in the portion of the square that can be reached by the arm. The objective is to learn a command for the arm to move from target to target, i.e reach new targets when they appear and then stabilize around them. The command is now the angular velocity (i.e. $\dot{\theta}$) and not the absolute angular position, so that the controller is expected to learn a *displacement* and not a *position*.

We consider that the end-point is not directly visible (the agent is not expected to “see” its own body) like in many robotic or real-life situations. Instead, the controller perceives a “proprioceptive” information, i.e. the current values of the joint angles. The controller must thus deduce end-position on the basis of joint angles (direct model), compute the difference with current target position, and send a motor command (inverse model) that orientates the arm toward the target. This combination of direct and inverse computation in a single operation is specific, we think, to our approach.

The visual input is encoded as previously on a 16×16 grid with a 2D gaussian kernel of radius 0.5. Every joint angle ($\in [-\pi, \pi]$) is encoded in a separate vector of 16 units uniformly spaced in $[-\pi, \pi]$ with a 1D Gaussian encoding of radius 0.5 also. With a 4 joint proprioceptive input, the total size of the first layer is 320 units. The output is composed of 4 units, which as usual receive the linear combination of the input with weights matrix $\mathbf{W}$, plus a Gaussian noise whose standard deviation is small ($\sigma = 0.003$). If $\mathbf{u}$ is the output, the final motor command is $\dot{\boldsymbol{\theta}} = 200 \times \pi \times \mathbf{u}$ rad/s. In simulation we consider discrete time steps of 50 ms so that $\Delta\theta = 10 \times \pi \times \mathbf{u}$ rad.

The reward is based on multiple criteria : visual error minimization, angular velocity minimization and also angular minimization (trying to avoid large values for θ). This gives concretely

$$r(t) = -0.1 \left(||\mathbf{x}_D - \hat{\mathbf{x}}||_2{}^2 + 0.025 \times ||\dot{\theta}||_2{}^2 + 0.01 \times ||\theta||_2{}^2 \right)$$

The reward is sent every time step, and the trace is updated with $\beta = 0.9$ (time constant of 500 ms for the trace). The learning parameter α is taken such that $\frac{\alpha}{\sigma^2} = 0.01$ where σ is the standard deviation of the noise injected into the system. The weights are initially 0.

Figure 4 (left) gives the cumulative loss (the opposite of the reward) during a session lasting 12×10^6 time steps. The level of noise remains constant as well as the learning rate (the system is continuously learning). The slope of the cumulative loss decreases and stabilizes at a value corresponding to a good achievement of the task (the arm smoothly moves from one target to the other and stabilizes on it). Note that with our setting, the loss can not be zero. At $t = 9.3 \times 10^6$ time steps, we 'block' the third joint angle at $\theta_3 = 0$. This prevents the controller from reaching the targets, and an increase in penalty (decrease in mean reward) is observed, which is progressively compensated so that the controller can reach the target anew with new combinations of commands on the remaining joint angles (fig. 4 (d)). It must be noticed that the achievement of the task is quite good despite the fact that the loss never reaches zero. In particular, the arm starts its goal-oriented movement immediately after the target has jumped at a new position, with speed decreasing with the distance to target.

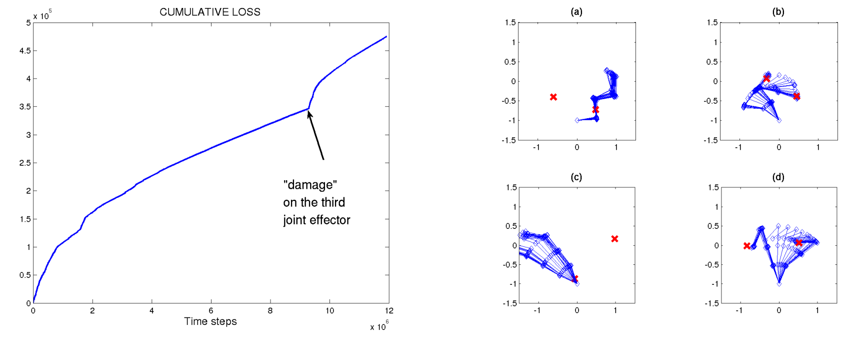

Figure 4: Left : cumulative loss during learning. A damage is caused on the device at $t = 9.3 \times 10^6$. Right : typical motor responses during target switch (targets reprsented by red crosses), arm position represented on 30 time steps (a) initially (b) after learning (c) after damage (third joint is blocked at 0) (d) after recovery.

5 Discussion

First, despite the difficulty of the task and setup, our on-line gradient algorithm is able to derive a valid and pertinent controller. As pointed out by Baxter and Bartlett, this is not something easy to obtain [7]. With our settings, improvement of the politics is ensured at the condition of small changes (the noise level and learning coefficients are small so that the learning sessions are quite long). Faster and still reliable convergence is an objective that can be attained in episodic learning, so as to get better estimate of the gradient. In a perpective of autonomy however, on-line adaptation to new constraints is an important property that could not be attained in an episodic setup without explicit failure detection (as learning is separated from exploitation in an episodic approach).

The arm system we consider, although non-linear and redundant is still a bit too simple when considering realistic mechanical constraints. Joints have unlimited course and two segments can cross each other without collision. This obviously facilitates the task. In principle, proprioceptive signal should allow to overcome such constraints, but this needs to be tested and verified.

The reward used in the closed-loop setting can have many different formulations. The one we used is quite informative and gives good results, but it should be theoretically possible to learn a good controller with a more basic formulation, e.g. giving only a non-nul reward in the vicinity of the target. Another interesting point could be to explore how the formulation of the reward influence the shape, the speed or the timing of the movement. Some general properties of human gestures, like isochrony or bell-shaped speed curve [12, 13], might be measured.

Using a gaussian noise as the underlying probability of the outputs neurons can be interpreted as looking for a good controller in L^2-norm. As shown by [14], using the L^1-norm is more adapted when the objective is to discriminate features in the input signal. Selecting relevant commands in the case of stongly redundant command space is an interesting perspective that could be tested using an exponential distribution in the stochastic nodes of the controller.

6 Conclusion

In this paper, we tried to shift the perspective of the reinforcement learning applications, traditionally devoted to difficult but discrete control achievement (like bang-bang polecart control for instance). Here we rather emphasize the ability of RL methods to face and compensate unpredictable changes taking place in the environment. The method we propose relies on a simple (noisy and linear) regressor to which on-line policy gradient is applied in order to learn a continuous command, on the basis of visual error mainly. The encoding method we use in the sensory layer, inspired by the kernel approach, allows to pass-by the limitations of linear regressors. Well adapted to the control problems we tackle here, it is however not scalable to high dimensional input spaces. The good achievement of the method in a complex learning setup offers, we hope, good perspectives for applications in real-world robotic tasks, as well as for modelling natural learning processes.

References

[1] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[2] Ronald Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[3] J. Baxter and P. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[4] E. Oyama, Nak Y. Chong, A. Agah, T. Maeda, S. Tachi, and K. F. MacDorman. Learning a coordinate transformation for a human visual feedback controller based on disturbance noise and the feedback error signal. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, pages 4186–4193. IEEE, 2001.

[5] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, March 2008.

[6] Shakabh Bhatnagar, Richard Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor-critic algorithms. Technical Report TR09-10, Univ. of Alberta, Dept. of Computing Sciences, June 2009.

[7] J. Baxter, P. Bartlett, and L. Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381, 2001.

[8] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA., 2001.

[9] Ranjeeth Dasineni. Kernel methods and factorization for image and video analysis. Master's thesis, International Institute of Information Technology, Hyderabad, India, 2007.

[10] J. Kivinen, A.J. Smola, and R.C. Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.

[11] E. Todorov and M. I. Jordan. Optimal feedback control as a theory of motor coordination. *Nature neuroscience*, 5(11):1226–1235, 2002.

[12] P. Viviani and R. Schneider. A developmental study of the relationship between geometry and kinematics in drawing movements. *Journal of Experimental Psychology*, 17(1):198–218, 1991.

[13] T. Flash and N. Hogan. The coordination of arm movements : An experimentally confirmed mathematical model. *The Journal of Neuroscience*, 5(7):1688–1703, 1985.

[14] J. A. Tropp. Greed is good: algorithmic results for sparse approximation. *Information Theory, IEEE Transactions on*, 50(10):2231–2242, 2004.