Apprentissage par imitation dans un cadre batch, off-policy et sans modèle

Edouard Klein, Matthieu Geist, Olivier Pietquin

▶ **To cite this version:**

Edouard Klein, Matthieu Geist, Olivier Pietquin. Apprentissage par imitation dans un cadre batch, off-policy et sans modèle. JFPDA 2011, Jun 2011, Rouen, France. pp.1-9. hal-00652762

HAL Id: hal-00652762

https://centralesupelec.hal.science/hal-00652762

Submitted on 17 Jan 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Apprentissage par imitation dans un cadre *batch*, *off-policy* et sans modèle

Edouard Klein[13], Matthieu Geist[1], and Olivier Pietquin[12]

1. Supélec-Metz Campus, IMS Research group, France, nom.prenom@supelec.fr
2. UMI 2958 CNRS - GeorgiaTech, France
3. Equipe ABC, LORIA-CNRS, France.

Résumé : Ce papier traite le problème de l'apprentissage par imitation, c'est à dire la résolution du problème du contrôle optimal à partir de données tirées de démonstrations d'un expert. L'apprentissage par renforcement inverse (IRL) propose un cadre efficace pour résoudre ce problème. En se basant sur l'hypothèse que l'expert maximise une fonction de valeur, l'IRL essaie d'apprendre la récompense qui définit cette dernière à partir de trajectoires d'exemple. Beaucoup d'algorithmes d'IRL font l'hypothèse de l'existence d'unn approximateur linéaire pour la fonction de récompense et calculent l'attribut moyen (le cumul moyen pondéré des fonctions de base, relatives à la paramétrisation linéaire supposée de la récompense, évaluées en les états d'une trajectoire associée à une certaine politique) via une estimation de Monte-Carlo. Cela implique d'avoir accès à des trajectoires complètes de l'expert ainsi qu'à au moins un modèle génératif pour tester les politiques intermédiaires. Dans ce papier nous introduisons une méthode de différences temporelles, LSTD-μ, pour calculer cet attribut moyen. Cela permet d'étendre l'apprentissage par imitation aux cas *batch* et *off-policy*.

1 Introduction

Optimal control consists in putting a machine in control of a system with the goal of fulfilling a specific task, optimality being defined as how well the task is performed. Various solutions to this problem exist from automation to planification. Notably, the reinforcement learning (RL) paradigm (Sutton & Barto, 1998) is a general machine learning framework in which an agent learns to control optimally a dynamic system through interactions with it. The task is specified through a reward function, the agent objective being to take sequential decisions so as to maximize the expected cumulative reward.

However, defining optimality (through the reward function) can itself be a challenge. If the system can be empirically controlled by an expert, even though his/her behavior can be difficult to describe formally, apprenticeship learning is a way to have a machine controlling the system by mimicking the expert. Rather than directly mimicking the expert with some supervised learning approach, Inverse Reinforcement Learning (IRL) (Ng & Russell, 2000) consists in learning a reward function under which the policy demonstrated by the expert is optimal. Mimicking the expert therefore ends up to learning an optimal policy according to this reward function. A significant advantage of such an approach is that expert's actions can be guessed in states which have not been encountered during demonstration. Firstly introduced in (Russell, 1998), another advantage claimed by the author would be to find a compact and complete representation of the task in the form of the reward function.

There roughly exists three families of IRL algorithms : feature-expectation-based (Abbeel & Ng, 2004; Syed *et al.*, 2008; Syed & Schapire, 2008; Ziebart *et al.*, 2008), marge-maximization-based (Ratliff *et al.*, 2006, 2007a,b; Kolter *et al.*, 2008) and approaches based on the parameterization of the policy by the reward function (Ramachandran & Amir, 2007; Neu & Szepesvári, 2007). The first family assumes a linearly parameterized reward function. This naturally leads to a linearly parameterized value function, the associated feature vector being the so-called feature expectation (defined as the cumulative discounted reward's feature vector under a given policy, see Section 2 for a formal definition). These approaches learn a reward function such that the feature expectation of

the optimal policy (according to the learnt reward function) is close to the feature expectation of the expert policy. This is a sufficient condition to have close value functions (for any parameterized reward function, and therefore particularly the optimized one). The second family expresses IRL as a constrained optimization problem in which expert's examples have higher expected cumulative reward than all other policies by a certain margin. Moreover, suboptimality of the expert can be considered through the introduction of slack variables. The last family parameterizes policies with a reward function. Assuming that the expert acts according to a Gibbs policy (respectively to the optimal value function related to the reward function which is optimized), it is possible to estimate the likelihood of a set of state-action pairs provided by the expert. The algorithms differs in the way this likelihood is maximized.

This paper focuses on the first family of algorithms, and more precisely on the seminal work of Abbeel & Ng (2004). All of them rely on the computation of the feature expectation (which depends on policies but not on rewards) of (i) the expert and (ii) some intermediate policies. The expert's feature expectation is computed using a simple Monte Carlo approach (which requires full trajectories of the expert). Other feature expectations are either computed exactly (which requires knowing analytically the dynamics of the system) or with a Monte Carlo approach (which requires simulating the system). The contribution of this paper is LSTD-μ, a new temporal-difference-based algorithm to estimate these feature expectations. It relaxes the predeceasing assumptions : transitions of the expert are sufficient (rather than full trajectories) and nor the model neither a simulator are necessary to compute intermediate feature expectations. This paper focuses on the algorithm introduced in (Abbeel & Ng, 2004), but we are confident that the proposed approach can be extended to other algorithms based on feature expectation computation (Syed *et al.*, 2008; Syed & Schapire, 2008; Ziebart *et al.*, 2008), or even (Ratliff *et al.*, 2006, 2007a,b).

The rest of this paper is organized as follows. Section 2 provides the necessary background, notably the definition of feature expectation and its use in the seminal IRL algorithm of Abbeel & Ng (2004). Section 3 presents LSTD-μ, our main contribution. Section 4 provides some preliminary experiments and Section 5 opens perspectives.

2 Background

A sequential decision problem is often framed as a Markov Decision Process (MDP) (Puterman, 1994). An MDP is a tuple $\{S, A, P, R, \gamma\}$ with S being the state space, A the action space, $P \in \mathcal{P}(S)^{S\times A}$ the set of Markovian transition probabilities, $R \in \mathbb{R}^S$ the reward function (assumed to be absolutely bounded by 1) and $\gamma \in [0, 1[$ a discounting factor. A policy $\pi \in A^S$ maps states to action. The quality of a policy is quantified by the associated value function V^π, which associates to each state the expected and discounted cumulative reward :

$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|s_0 = s, \pi] \tag{1}$$

Dynamic programming aims at finding the optimal policy π^*, that is one of the policies associated to the optimal value function, $V^* = \text{argmax}_\pi V^\pi$, which maximizes the value for each state. If the model (that is transition probabilities and the reward function) is unknown, learning the optimal control through interactions is addressed by reinforcement learning.

For inverse reinforcement learning, the problem is reversed. It is assumed that an expert acts according to an optimal policy π_E, this policy being optimal according to some unknown reward function R^*. The goal of inverse reinforcement learning is to learn this reward function from sampled trajectories of the expert. This is a difficult and ill-posed problem (Ng & Russell, 2000). Apprenticeship learning through IRL, which is the problem at hand in this paper, has a somehow weaker objective : it aims at learning a policy $\tilde{\pi}$ which is (approximately) as good as the expert policy π_E under the unknown reward function R^*, for a known initial state s_0 (this condition can be weakened by assuming a distribution over initial states ; this is not done here for clarity of exposition). Now, the approach proposed in (Abbeel & Ng, 2004) is presented.

We assume that the true reward funnotiction belongs to some hypothesis space $\mathcal{H}_\phi = \{\theta^T\phi(s), \theta \in \mathbb{R}^p\}$, of which we assume the basis functions to be bounded by 1 : $|\phi_i(s)| \leq 1, \forall s \in S, 1 \leq i \leq p$.

Therefore, there exists a parameter vector θ^* such that :

$$R^*(s) = (\theta^*)^T \phi(s) \tag{2}$$

In order to ensure that rewards are bounded, we impose that $\|\theta\|_2 \leq 1$. For any reward function belonging to $\mathcal{H}_\phi$ and for any policy π, the related value function $V^\pi(s)$ can be expressed as follows :

$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t \theta^T \phi(s_t) | s_0 = s, \pi] \tag{3}$$

$$= \theta^T E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | s_0 = s, \pi] \tag{4}$$

Therefore, the value function is also linearly parameterized, with the same weights and with basis functions being grouped into the so-called *feature expectation* μ^π :

$$\mu^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | s_0 = s, \pi] \tag{5}$$

Recall that the problem is to find a policy whose performance is close to that of the expert's for the starting state s_0, on the unknown reward function R^*. In order to achieve this goal, it is proposed in (Abbeel & Ng, 2004) to find a policy $\tilde{\pi}$ such that $\|\mu^{\pi_E}(s_0) - \mu^{\tilde{\pi}}(s_0)\|_2 \leq \epsilon$ for some (small) $\epsilon > 0$. Actually, this ensures that the value of the expert's policy and the value of the estimated policy (for the starting state s_0) are close for *any* reward function of $\mathcal{H}_\phi$:

$$|V^{\pi_E}(s_0) - V^{\tilde{\pi}}(s_0)| = |\theta^T(\mu^{\pi_E}(s_0) - \mu^{\tilde{\pi}}(s_0))| \tag{6}$$

$$\leq \|\mu^{\pi_E}(s_0) - \mu^{\tilde{\pi}}(s_0)\|_2 \tag{7}$$

This last equation uses the Cauchy-Schwartz inequality and the assumption that $\|\theta\|_2 \leq 1$. Therefore, the approach described here does not ensure to retrieve the true reward function R^*, but to act as well as the expert under this reward function (and actually under any reward function, notably the unknown optimal one).

Let us now describe the algorithm proposed in (Abbeel & Ng, 2004) to achieve this goal :

1. Starts with some initial policy $\pi^{(0)}$ and compute $\mu^{\pi^{(0)}}(s_0)$. Set $j = 1$;
2. Compute $t^{(j)} = \max_{\theta:\|\theta\|_2 \leq 1} \min_{k \in \{0,j-1\}} \theta^T(\mu^{\pi_E}(s_0) - \mu^{\pi^{(k)}}(s_0))$ and let $\theta^{(j)}$ be the value attaining this maximum. At this step, one searches for the reward function which maximizes the distance between the value of the expert at s_0 and the value of *any* policy computed so far (still at s_0). This optimization problem can be solved using a quadratic programming approach or a projection algorithm (Abbeel & Ng, 2004) ;
3. if $t^{(j)} \leq \epsilon$, terminate. The algorithm outputs a set of policies $\{\pi^{(0)}, \ldots, \pi^{(j-1)}\}$ among which the user chooses manually or automatically the closest to the expert (see (Abbeel & Ng, 2004) for details on how to choose this policy [1]). Notice that the last policy is not necessary the best (as illustrated in Section 4) ;
4. solve the MDP with the reward function $R^{(j)}(s) = (\theta^{(j)})^T \phi(s)$ and denote $\pi^{(j)}$ the associated optimal policy. Compute $\mu^{\pi^{(j)}}(s_0)$;
5. set $j \leftarrow j + 1$ and go back to step 2.

There remains three problems : computing the feature expectation of the expert, solving the MDP and computing feature expectations of intermediate policies.

As suggested in (Abbeel & Ng, 2004), solving the MDP can be done approximately by using any appropriate reinforcement learning algorithm. In this paper, we use the Least-Squares Policy Iteration (LSPI) algorithm (Lagoudakis & Parr, 2003). There remains to estimate feature expectations. In (Abbeel & Ng, 2004), $\mu^{\pi_E}(s_0)$ is estimated using a Monte Carlo approach over m trajectories :

$$\hat{\mu}_E(s_0) = \frac{1}{m} \sum_{h=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(h)}) \tag{8}$$

1. A simple solution, not mentioned in (Abbeel & Ng, 2004), is to choose the policy of which the feature expectation is the closest to the one of the expert.

This approach does not hold if only transitions of the expert are available, or if trajectories are too long (in this case, it is still possible to truncate them). For intermediate policies, it is also suggested to estimate associated feature expectations using a Monte Carlo approach (if they cannot be computed exactly). This is more constraining than for the expert, as this assumes that a simulator of the system is available. In order to address these problems, we introduce a temporal-difference-based algorithm to estimate these feature expectations.

3 LSTD-μ

Let us write the definition of the i^{th} component of a feature expectation $\mu^\pi(s)$ for some policy π :

$$\mu_i^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t \phi_i(s_t) | s_0 = s, \pi] \tag{9}$$

This is exactly the definition of the value function of the policy π for the MDP considered with the i^{th} basis function $\phi_i(s)$ as the reward function. There exists many algorithms to estimate a value function, any of them can be used to estimate μ_i^π. Based on this remark, we propose to use specifically the least-squares temporal difference (LSTD) algorithm (Bradtke & Barto, 1996) to estimate each component of the feature expectation (as each of these components can be understood as a value function related to a specific and known reward function).

Assume that a set of transitions $\{(s_t, r_t, s_{t+1})_{1 \leq t \leq n}\}$ sampled according to the policy π is available. We assume that value functions are searched for in some hypothesis space $\mathcal{H}_\psi = \{\hat{V}_\xi(s) = \sum_{i=1}^q \xi_i \psi_i(s) = \xi^T \psi(s), \xi \in \mathbb{R}^q\}$. As reward and value functions are possibly quite different, another hypothesis space is considered for value function estimation. But if $\mathcal{H}_\phi$ is rich enough, one can still consider $\mathcal{H}_\psi = \mathcal{H}_\phi$. Therefore, we are looking for an approximation of the following form :

$$\hat{\mu}_i^\pi(s) = (\xi_i^*)^T \psi(s) \tag{10}$$

The parameter vector ξ_i^* is here the LSTD estimate :

$$\xi_i^* = \left(\sum_{t=1}^n \psi(s_t)(\psi(s_t) - \gamma \psi(s_t'))^T \right)^{-1} \sum_{t=1}^n \psi(s_t)\phi_i(s_t) \tag{11}$$

For apprenticeship learning, we are interested more particularly in $\hat{\mu}^\pi(s_0)$. Let $\Psi = (\psi_i(s_t))_{t,i}$ be the $n \times q$ matrix of values predictors, $\Delta\Psi = (\psi_i(s_t) - \gamma \psi_i(s_t'))_{t,i}$ be the related $n \times q$ matrix and $\Phi = (\phi_i(s_t))_{t,i}$ the $n \times p$ matrix of reward predictors. It can be easily checked that $\hat{\mu}^\pi(s_0)$ satisfies :

$$(\hat{\mu}^\pi(s_0))^T = \psi(s_0)^T (\Psi^T \Delta\Psi)^{-1} \Psi^T \Phi \tag{12}$$

This provides an efficient way to estimate the feature expectation of the expert in s_0.

There remains to compute the feature expectations of intermediate policies, which should be done in an off-policy manner (that is without explicitly sampling trajectories according to the policy of interest). To do so, still interpreting each component of the feature expectation as a value function, we introduce a state-action feature expectation defined as follows (much as the classic Q-function extends the value function) :

$$\mu^\pi(s, a) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | s_0 = s, a_0 = a, \pi] \tag{13}$$

Compared to the classic feature expectation, this definition adds a degree of freedom on the first action to be chosen before following the policy π. With a slightly different definition of the related hypothesis space, each component of this feature expectation can still be estimated using the LSTD algorithm (namely using the LSTD-Q algorithm (Lagoudakis & Parr, 2003)). The clear advantage of introducing the state-action feature expectation is that this additional degree of freedom allows off-policy learning. Assuming that $\mathcal{H}_\psi$ defines state-action feature vectors and that a set of transitions $\{(s_t, a_t, r_t, s_{t+1})_{1 \leq t \leq n}\}$ sampled according to some behaviorial policy π_0 is available, the parameter

vector ξ_i^* related to the i^{th} component of the state-action feature expectation $\mu^\pi(s)$ (policy π being different from policy π_0) is given by :

$$\xi_i^* = \left(\sum_{t=1}^{n} \psi(s_t, a_t)(\psi(s_t, a_t) - \gamma\psi(s'_t, \pi(s'_t)))^T \right)^{-1} \sum_{t=1}^{n} \psi(s_t, a_t)\phi_i(s_t) \tag{14}$$

Extending LSTD-μ to state-action LSTD-μ is done in the same manner as LSTD is extended to LSTD-Q (Lagoudakis & Parr, 2003), technical details are not provided here for the clarity of exposition.

Given the (state-action) LSTD-μ algorithm, the apprenticeship learning algorithm of (Abbeel & Ng, 2004) (see Section 2) can be easily extended to a batch and off-policy setting. The solely available data is a set of transitions sampled according to the expert policy (and possibly a set of sub-optimal but good trajectories). The corresponding feature expectation for the starting state s_0 is estimated with the LSTD-μ algorithm. At step 4 of this algorithm, the MDP is (approximately) solved using LSPI (Lagoudakis & Parr, 2003) (an approximate policy iteration algorithm using LSTD-Q as the off-policy Q-function estimator). The corresponding feature expectation at state s_0 is estimated using the proposed state-action LSTD-μ.

Before presenting some experimental results, let us stress that LSTD-μ is simply the LSTD algorithm applied to a specific reward function. Although quite clear, the idea of using a temporal difference algorithm to estimate the feature expectation is new, as far as we know. A clear advantage of the proposed approach is that any theoretical result holding for LSTD also holds for LSTD-μ, such as convergence (Nedić & Bertsekas, 2003) or finite-sample (Lazaric *et al.*, 2010) analysis for example, also, the algorithm learns from transitions and not trajectories.

4 Experimental benchmark

This section provides some preliminary experimental results on a simple maze benchmark. Subsection 4.1 details the protocol and the results while subsection 4.2 inspects the meaning of the different quality criteria.

4.1 Experiment description and results

The experimental benchmark chosen here is one of those proposed in (Ng & Russell, 2000), a 5x5 grid world. The agent is in one of the cell of the grid (whose coordinates is the state) and can choose at each step one of the four compass directions (the action). With probability 0.9, the agent moves in the intended direction. With probability 0.1, the direction is randomly drawn (and thus have probability 0.25 to match the original direction). The reward optimized by the expert is 0 everywhere except in the upper-right cell, where it is 1. For every policy, an episode ends when the upper right cell is attained, or after 20 steps. At the start of each episode, the agent is in the lower-left corner of the grid (the opposite of where the reward is).

Both the state and action spaces are finite and of small cardinality. Hence, the chosen feature functions ϕ and ψ (see equations (9) and (10) for a remainder of their meaning) are the typical features of a tabular representation :0 everywhere except the component corresponding to the current state (-action pair).

Both Abbeel & Ng (2004)'s algorithm (from now on referred to as the MC variant) and our adaptation (referred to as the LSTD variant) are tested side by side. The MDP solver of the mc variant is LSPI with a sample source that covers the whole grid (each state has a mean visitation count of more than 150) and draws its action randomly. Both $\mu^{\pi^{(j)}}(s_0)$ and $\mu^{\pi_E}(s_0)$ are computed thanks to a Monte-Carlo estimation with enough samples to make the variance negligible. We consider both these computations as perfect theoretical solvers for all intended purpose on this toy problem. We thus are in the case intended by Abbeel & Ng (2004).
On the other hand the LSTD variant is used without accessing a simulator. It uses LSPI and LSTDμ, fed only with the expert's transitions (although we could also use non expert transitions

if available). This corresponds to a real-life setting where data generation is expensive and the system can not be controlled by an untrained machine.

We want to compare the efficiency of both versions of the algorithm with repect to the number of samples available from the expert, as these samples usually are the bottleneck. Indeed as they are quite costly (in both means and human time) to generate they are often not in abundance hence the critical need for an algorithm to be expert-sample efficient. Having a simulator at sake can also be difficult. For the simple problem we use this is however not an issue. The discussion about the choice of the performance metric has its own dedicated subsection (subsection 4.2). We use here the $||\mu^{\pi^{(j)}}(s_0) - \mu^{\pi_E}(s_0)||_2$ error term.

Figure 1 shows, for some numbers of samples from the expert, the value of $||\mu^{\pi^{(j)}}(s_0) - \mu^{\pi_E}(s_0)||_2$ after the algorithm converged or attained the maximum number of iterations (we used 40). The best policy is found by LSTD variant after one iteration only [2] whereas in the MC variant, convergence happens after at least ten iterations. The best policy is not always the last, and although it experimentally always have been with the LSTD variant, there is absolutely no way to tell whether this is a feature. The score of the best policy (not the last) according to the $||\mu^{\pi^{(j)}}(s_0) - \mu^{\pi_E}(s_0)||_2$ error term is plotted here.

We can see that although the LSTD variant is not as good as the MC variant when very few samples are available, both algorithm quickly converge to almost the same value ; our version converges to a slightly lower error value. The error bars shown correspond to the standard deviation over 100 runs. The fact that our variant can work in a batch, off-policy and model-free setting should make it suitable to a range of task inaccessible to the MC variant.

4.2 Discussion about the quality criterion

Figure 2 and 3 illustrate the difficulty of choosing the quality criterion ; Figure 2 shows four different quality ciretria during a run of the MC variant. Figure 3 shows the same criteria for several runs of the LSTD variant, as the abscissa can not be the number of iterations (as in Figure 2) because it always converges in one iteration. The $||\mu^{\pi^{(j)}}(s_0) - \mu^{\pi_E}(s_0)||_2$ term is widely discussed in (Abbeel & Ng, 2004)'s additional material. It bears an interesting relation with the difference between the expert's value function and the current value function in the initial state with respect *to the current reward* (equation 6).

The fact that the curves of the true error term $||\mu^{\pi^{(j)}}(s_0) - \mu^{\pi_E}(s_0)||_2$ and its estimate $||\hat{\mu}^{\pi^{(j)}}(s_0) - \hat{\mu}^{\pi_E}(s_0)||_2$ are indistinguishable in Figure 2 means that, for it has access to a cheap simulator, the MC variant works as if it had access to the exact values. This however can not be said of the LSTD variant, for which the two curves are indeed different (Figure 3). Not knowing the true value of $\mu^{\pi_E}(s_0)$ may be a problem for our variant, as it can introduce an error in the stopping criterion of the algorithm [3], $t = ||\bar{\mu}^{(j)} - \hat{\mu}^{\pi_E}(s_0)||$ where $\bar{\mu}^{(j)}$ is the projection of μ^{π_E} on $(\mu^{\pi^{(j-1)}}, \mu^{\pi^{(j)}})$. After the algorithm has converged there is no way to know whether the policy is actually good or not other than actually testing it.

It shall be noted that although it plays its role, the halt criterion is not a good measure of the quality of the current policy in the MC variant either, as it can be low (and thus halt the algorithm) when the policy is bad. The best policy, however, can be easily chosen among all those created during the execution of the algorithm thanks to the $||\mu^{\pi^{(j)}}(s_0) - \mu^{\pi_E}(s_0)||_2$ term, which the MC variant can compute. When this term is low, the objective performance (that is, $V^{\pi_E}(s_0) - V^{\pi^{(j)}}(s_0)$ with respect to the unknow true reward function) is low too.

2. Precise reasons for what it happens are not clear now, but certainly have something to do with the fact that all the estimations are made along the same distribution of samples.

3. This equation differs from what has been given as the second step of the algorithm in section 2. This is because we implemented the version of the algorithm that uses a projection (see Abbeel & Ng (2004)'s supplementary material for an in-depth discussion of both versions). Although the mathematical expressions are different, both convey the same meaning and are exchangeable.

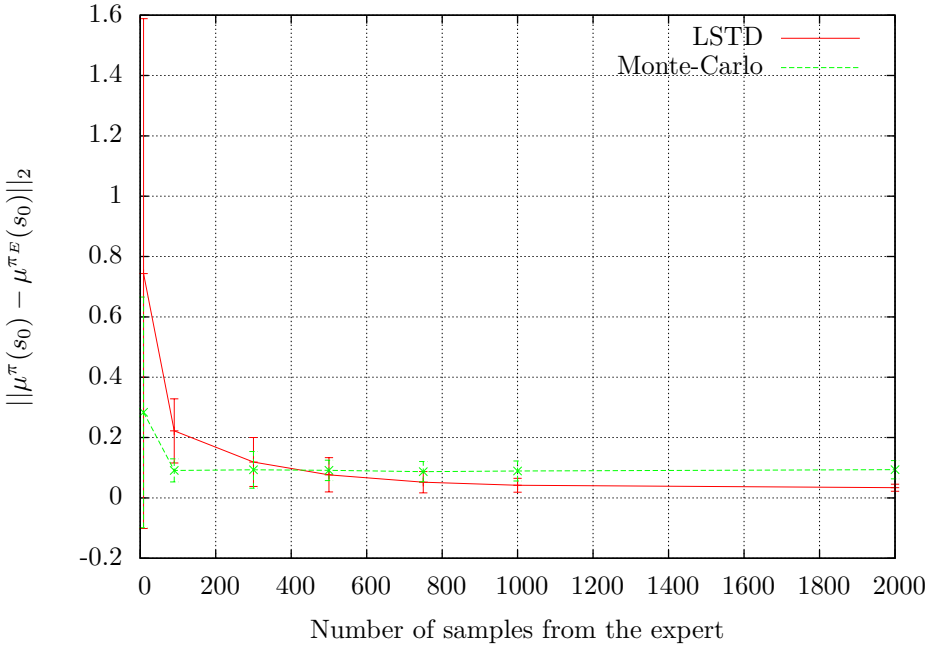

FIGURE 1 – $||\mu^{\pi}(s_0) - \mu^{\pi_E}(s_0)||_2$ where π is the policy for which this term is the lowest for the whole run, with respect to the number of samples available from the expert. Our algorithm is not as good as Abbeel & Ng (2004) with very few samples, but make up for it very soon and converges to a lower value. The error bars represent the standard deviation over 100 runs.

5 Conclusion

Given some transitions generated by an expert controlling a system and maximizing in the long run an unknown reward, we ported Abbeel & Ng (2004)'s approach to apprenticeship learning vie inverse reinforcement learning to a batch, model-free, off-policy setting. Experimentally, there is a need for a bigger number of samples from the expert. We believe this cost is not prohibitive as our approach only requires isolated samples which are often less difficult to get than whole trajectories as needed by the original approach. The simple idea of using LSTD to estimate the feature expectation could be applied to other algorithms as well, for example (Abbeel & Ng, 2004; Syed *et al.*, 2008; Syed & Schapire, 2008; Ziebart *et al.*, 2008).

Références

ABBEEL P. & NG A. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, p.1̃ : ACM.

BRADTKE S. & BARTO A. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, **22**(1), 33–57.

KOLTER J., ABBEEL P. & NG A. (2008). Hierarchical apprenticeship learning with application to quadruped locomotion. In *Neural information processing systems*, volume 20 : Citeseer.

LAGOUDAKIS M. & PARR R. (2003). Least-squares policy iteration. *The Journal of Machine Learning Research*, **4**, 1107–1149.

LAZARIC A., GHAVAMZADEH M. & MUNOS R. (2010). Finite-sample analysis of lstd. In *Proceedings of the 27th International Conference on Machine Learning* : Citeseer.

NEDIĆ A. & BERTSEKAS D. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, **13**(1), 79–110.

NEU G. & SZEPESVÁRI C. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proc. UAI*, p. 295–302 : Citeseer.

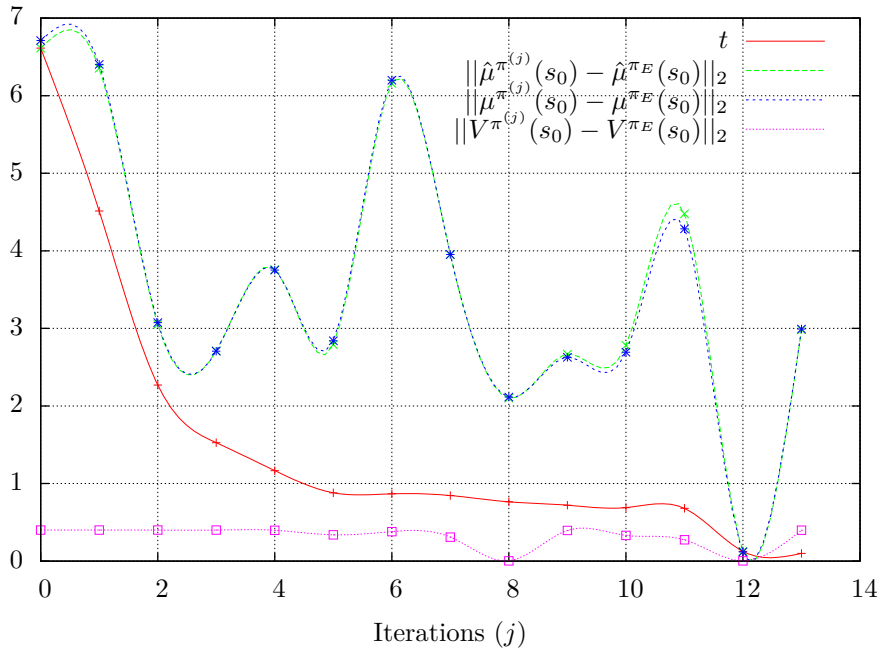

FIGURE 2 – Different criteria with respect to the number of iterations for a run of the MC variant.

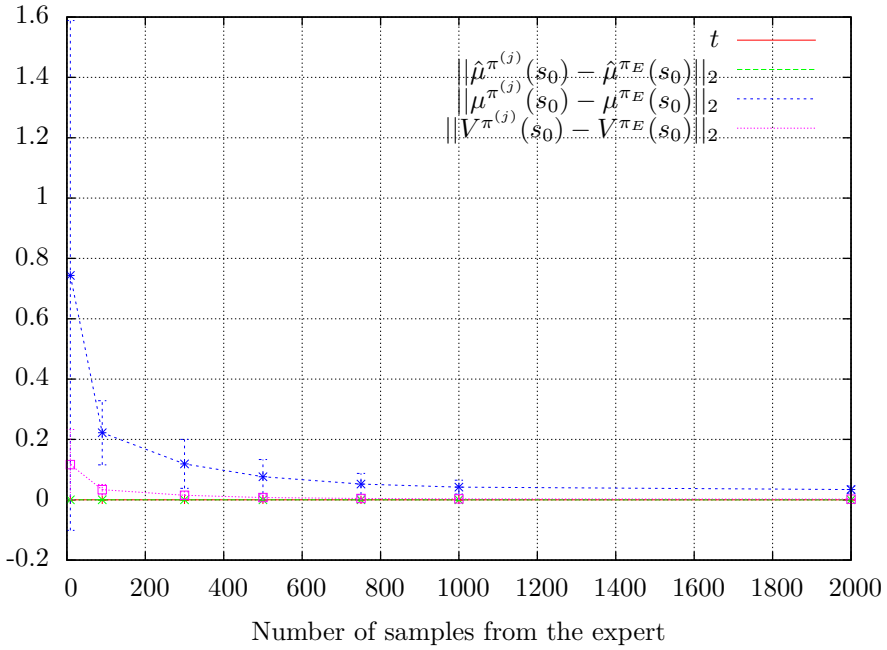

FIGURE 3 – Different criteria with respect to the number of samples from the expert, for several runs of the LSTD variant. We can see that the algorithm is blind, as all it has access to is always zero. The true error values, however, smoothly converge to something small. Knowing how many expert samples are actually needed in a real world problem is an open question. The error bars represent the standard deviation over 100 runs.

Ng A. & Russell S. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, p. 663–670 : Morgan Kaufmann Publishers Inc.

Puterman M. (1994). *Markov decision processes : Discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, NY, USA.

Ramachandran D. & Amir E. (2007). Bayesian inverse reinforcement learning. *Urbana*, **51**, 61801.

Ratliff N., Bagnell J. & Srinivasa S. (2007a). Imitation learning for locomotion and manipulation. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, p. 392–397 : IEEE.

Ratliff N., Bagnell J. & Zinkevich M. (2006). Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, p. 736 : ACM.

Ratliff N., Bradley D., Bagnell J. & Chestnutt J. (2007b). Boosting structured prediction for imitation learning. *Advances in Neural Information Processing Systems*, **19**, 1153.

Russell S. (1998). Learning agents for uncertain environments (extended abstract). In *Proceedings of the eleventh annual conference on Computational learning theory*, p. 103 : ACM.

Sutton R. & Barto A. (1998). *Reinforcement learning*. MIT Press.

Syed U., Bowling M. & Schapire R. (2008). Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, p. 1032–1039 : ACM.

Syed U. & Schapire R. (2008). A game-theoretic approach to apprenticeship learning. *Advances in neural information processing systems*, **20**, 1449–1456.

Ziebart B., Maas A., Bagnell J. & Dey A. (2008). Maximum entropy inverse reinforcement learning. In *Proc. AAAI*, p. 1433–1438.