A Non-Parametric Approach to Approximate Dynamic Programming

Hadrien Glaude, Fadi Akrimi, Matthieu Geist, Olivier Pietquin

A Non-Parametric Approach to Approximate Dynamic Programming

Hadrien Glaude*, Fadi Akrimi*, Matthieu Geist*† and Olivier Pietquin*†
*SUPELEC (IMS Research Group)
†UMI 2958 (GeorgiaTech - CNRS)
2 rue Edouard Belin - 57070 Metz (France)
Email: firstname.lastname@supelec.fr

***Abstract*—Approximate Dynamic Programming (ADP) is a machine learning method aiming at learning an optimal control policy for a dynamic and stochastic system from a logged set of observed interactions between the system and one or several non-optimal controlers. It defines a class of particular Reinforcement Learning (RL) algorithms which is a general paradigm for learning such a control policy from interactions. ADP addresses the problem of systems exhibiting a state space which is too large to be enumerated in the memory of a computer. Because of this, approximation schemes are used to generalize estimates over continuous state spaces. Nevertheless, RL still suffers from a lack of scalability to multidimensional continuous state spaces. In this paper, we propose the use of the Locally Weighted Projection Regression (LWPR) method to handle this scalability problem. We prove the efficacy of our approach on two standard benchmarks modified to exhibit larger state spaces.**

I. Introduction

Reinforcement Learning [1] is concerned with learning by interactions how an agent ought to take actions in a stochastic environment so as to adopt an optimal behavior. An interaction can be described as a modification of the state of the environment as an effect of an action performed by an intelligent agent. After each interaction, the intelligent agent is provided with a immediate numerical reward which indicates how good it was to perform the action in the given environment state. The optimal behavior is the one that maximizes some function of the long-term cumulative reward. Originally, this optimal control problem was solved by formulating the it as a Markov Decision Process (MDP) [2]. An MDP is a tuple $\{S, A, T, R, \gamma\}$ where S is the set of the possible environment states, A is the set of agent actions, T is a family of transition probabilities describing the one-step dynamics of the environment, R is the reward function associating a reward to each state transition, and γ is some discounting factor described later. If all these quantities are known, Dynamic Programming (DP) [2] provides an exact solution to the control policy optimization. However, an exact representation of the MDP is not tractable for large environments since the enumeration of all the possible states is not feasible and the transition probabilities cannot be estimated for every transition. So it is not suitable for real applications. Thus many solutions have been proposed to deal with some approximations of the MDP among which Approximate Dynamic Programming (ADP) methods [3], [4].

Least Square Policy Iteration (LSPI) [5], is such an ADP algorithm which combines value-function approximation with linear architectures and approximate policy iteration. It is a batch algorithm learning from a fixed set of logged interactions. The approach was motivated by the Least Square Temporal Difference (LSTD) [6]. The approach proposed here finds its inspiration in [7], where the authors have studied performances of the combination of an approximate value function approximation [8] algorithm (Fitted-Q) with tree-based regressors. Although ADP has known dramatic improvements during the last decade, it still suffers from a lack of scalability and is still unable to handle multidimensional continuous state spaces with many dimensions, especially if some dimension are irrelevant for the control problem [9]. In this article we study the performances of the combination of the Fitted-Q algorithm with an efficient regressor widely-used in robotics, namely the Locally Weighted Projection Regression (LWPR) [10]. The ability of LWPR to locally select the useful dimensions will be used to scale-up to large state spaces with the Fitted-Q algorithm.

This paper is organized as follows. In Section II, we describe the reinforcement learning theory, introduce the concept of the Q-function and the value function and explain the Fitted-Q iteration algorithm. In Section III, we describe the Locally Weighted Projection Regression (LWPR) algorithm and its advantages. In Section IV, we present the different configurations we tested to combine the Fitted-Q iteration algorithm and LWPR. Section V is dedicated to the experiments where we apply the Fitted-Q iteration algorithm used with LWPR to solve the inverted pendulum problem and the mountain car problem. In order to assess the performance of our approach in high dimensional problems we run experiments on the Cartesian product of problems and on problems with noise components. Section VI concludes our work and also provides our main directions for further research.

II. Reinforcement learning theory

As described in Section I, reinforcement learning aims at designing algorithms by which an intelligent agent learns to behave consistently in some environment, from its interaction with this environment or from observations gathered from the environment. After each interaction, the intelligent agent observes a reward or a penalty, finding the optimal control policy is then realized by maximizing the cumulation of these rewards in the long term. The quality of the control is often described by a value function V which associates to each state of the system the expected accumulated rewards starting from that state and then following the policy. Our study focuses mainly on an off-policy learning in batch mode, which means that the optimal policy is learned using recorded data generated by a random or a non optimal policy (off-policy), and without new interaction with the environment.

A. Value function and Q-function

We consider an agent operating in discrete time, observing at time t the environment state s_t, taking an action a_t, and receiving back information from observing the environment (the next state s_{t+1} and the instantaneous reward r_t). After some finite time, the experience the agent has gathered from interacting with the environment may thus be represented by a set of four-tuples $(s_t,a_t,r_t,s_{t+1})_{t=0,\cdots,T}$. The value function V evaluates the expected reward in the long term in each state ($\forall s \in S$) following the policy π.

$$V_\gamma^\pi(s) = E^\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}(s_t, a_t)|s_0 = s] \quad (1)$$

where S is the state space and γ is a discount factor ($0 \leq \gamma < 1$) that weights short-term rewards more than long-term ones.

Although state-values suffice to define optimality, we can use a local value function of the long term reward if a particular action is chosen ($\forall s \in S, \forall a \in A$),

$$Q_\gamma^\pi(s) = E^\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}(s_t, a_t)|s_0 = s, a_0 = a] \quad (2)$$

where A is the action space. The optimal policy π^* is the policy that maximizes V_γ(s) for each state s.

$$V_\gamma^*(s) = \max_\pi V_\gamma^\pi(s) = \max_{a \in A} Q_\gamma^*(s, a) \quad (3)$$

$$Q_\gamma^*(s, a) = \max_\pi Q_\gamma^\pi(s, a) \quad (4)$$

B. Value iteration with approximation

The Value Iteration with Approximation algorithm, more known as the Fitted-Q Iteration algorithm as described in [7], computes an approximation $\hat{Q}^*(s, a)$ of the optimal Q-function from a set of four-tuples F = {(s_t^l,a_t^l,r_t^l,s_{t+1}^l),l=1,...,$\#F$}. From this, a policy $\hat{\pi}^*$ is derived by selecting the greedy action according to the Q-function: $\hat{\pi}^*(s) = \text{argmax}_a \hat{Q}^*(s, a)$.

For a temporal horizon of N steps, we define the sequence of Q_N-functions on $S \times A$ by

$$\begin{aligned} & Q_0(s, a) = 0 \\ \forall N > 0 \qquad & Q_N(s, a) = (HQ_{N-1})(s, a) \end{aligned} \quad (5)$$

This set of functions should converge to the optimal Q-function, defined as the (unique) solution of the Bellman equation

$$Q(s, a) = (HQ)(s, a)) \quad (6)$$

where H is the Bellman operator mapping any function $K : S \times A \rightarrow \mathbb{R}$ and defined as follows

$$(HK)(s, a) = E[r(s, a) + \gamma max_{a \in A} K(s, a, a')] \quad (7)$$

At each step this algorithm may use the full set of four-tuples gathered from observations of the system together with the function computed at the previous step to determine a new training set which is used by a supervised learning (regression) method to compute the next function. It produces a sequence of $\hat{Q}_N$-functions, approximations of the Q_N-functions defined by eq. (5). This is summarized in the following algorithm:

Inputs: a set of four-tuples F and a regression algorithm (LWPR in our case).
Initialization: Set N to 0. Let $\hat{Q}_N$ be a function equal to zero everywhere on $S \times A$.
Iterations:
Repeat until stopping conditions are reached
- $N \leftarrow N + 1$.
- Build the training set $TS = \{(i^l, o^l), l = 1, ..., \#F\}$ based on the the function $\hat{Q}_{N-1}$ and on the full set of four-tuples F :

$$\begin{aligned} i^l &= (s_t^l, a_t^l) \\ o^l &= r_t^l + \gamma max_{a \in A} \hat{Q}_{N-1}(s_{t+1}^l, a) \end{aligned}$$

- Use the regression algorithm to induce from *TS* the function $\hat{Q}_N$(s,a).

A stopping condition is required to decide at which iteration (i.e., for which value of N) the process can be stopped. A simple way to stop the process is to define *a priori* a maximum number of iterations. Another possibility would be to stop the iterative process when the distance between $\hat{Q}_N$ and $\hat{Q}_{N-1}$ drops below a certain threshold.

III. Locally Weighted Projection Regression

In this section, the description of LWPR is largely inspired by [10]. As mentioned earlier, the Fitted-Q Iteration algorithm needs a function approximation scheme to induce an estimate $\hat{Q}_N$(s,a) from the training set TS.

LWPR is an algorithm for incremental non-linear function approximation in high-dimensional spaces with locally redundant and irrelevant input dimensions. LWPR is widely used in robotics. At its core, it employs non-parametric regression with locally linear models. In order to stay computationally efficient and numerically robust, each local model employs the partial least-square regression (PLS) [11] to perform the linear regression analysis on a small number of selected directions in input space.

The advantage of using LWPR is that it learns rapidly with learning methods based on incremental training, uses statistically sound stochastic leave-one-out cross validation for learning without the need to memorize training data, adjusts its weighting kernels based on only local information in order to minimize the danger of negative interference of incremental learning, has a computational complexity that is linear in the number of inputs, can deal with a large number of - possibly redundant - inputs by locally reducing input dimensions, and does not need an *a priori* information about the structure to be approximated.

In order to reduce dimensionality, a weighted PLS is locally used. PLS recursively computes orthogonal projections of the input data and performs single-variable regressions along these projections on the residuals of the previous iteration step.

For non linear function approximation, the core concept of the LWPR is to find approximation by means of piecewise linear models called receptive fields. Learning involves automatically determining the appropriate number of receptive fields, the region of validity and if wanted the size of each receptive field. The prediction step is then a weighting between local receptive fields predictions.

A measure of locality for each data point, the weight w_i, is computed from a Gaussian kernel

$$w_i = exp(-\frac{1}{2}(x - c_k)^T D_k (x - c_k)) \quad (8)$$

where D is a positive semi-definite distance metric that determines the size and shape of the neighborhood contributing to the local model and c the center of the receptive field. w is also called the activation of a receptive field.

Given a query point x, every linear model calculates a prediction $\hat{y}_k$(x). The total output of the learning system is the normalized weighted mean of all K linear models

$$\hat{y} = \frac{\sum_{k=1}^K w_k \hat{y}_k}{\sum_{k=1}^K w_k} \quad (9)$$

The distance metric D and, hence, the locality of the receptive fields, can be learned for each local model individually by stochastic gradient descent in a penalized leave-one-out cross-validation cost

function

$$J = \frac{1}{\sum_{i=1}^{M} w_i} \sum_{i=1}^{M} w_i (y_i - \hat{y}_{i,-i})^2 + \frac{P}{N} \sum_{i,j=1}^{N} D_{ij}^2 \qquad (10)$$

where M denotes the number of data points in the training set. The first term of the cost function is the mean leave-one-out cross-validation error of the local model (indicated by the subscript i, $-i$) which ensures proper generalization. The second term, the penalty term, makes sure that receptive fields cannot shrink indefinitely in case of large amounts of training data.

This cost function adjusts also the PLS axis and makes possible the adding and the removal of receptive fields in a relevant way.

IV. Explored leads

There are several ways to combine LWPR with the Fitted-Q algorithm among which we selected some which are explained hereafter. First and foremost, at each iteration the training set is presented only once in the exact same order to the LWPR algorithm. In [10], the author advises to present the training set many times in a random order if there are not enough samples.

As we first consider problems with discrete actions, we chose not to regress on action dimensions. Instead of having one regressor working on the input space composed by state-action pairs, our algorithm uses as many regressors as there are actions. Each regressor takes the state space as input. This choice is motivated by two reasons. First, it may be irrelevant to interpolate between actions because it implies a metric to exist between those. For example, if we consider a robot, grabbing an item and moving to the north have no relationship. Secondly, LWPR tries to reduce the number of dimensions of the input space by selecting some relevant dimensions through the PLS regression. But, with this process, there is no guarantee to select at least one action dimension. So, for a specific state, if no action dimension is selected by the regressor, the optimal policy will choose a random action. Therefore, with continuous actions, keeping or not action dimensions in the input space should be more closely examined.

Without an *a priori* on the state space, we remark that normalizing the input gives better results, even in presence of irrelevant dimensions, otherwise dimensions with high variations will be privileged. Indeed, the penalty terms in eq. 10 tends to produce round receptive fields by penalizing ellipsis with a big eccentricity. That is why, $D = dI$, where D is the distance metric, I is the identity matrix and d is a scale parameter. The value of d determines the initial size of receptive fields.

One of the most important parts of our researches was to figure out what are the consequences of allowing the LWPR algorithm to update D. As we explain in the previous section, the matrix D defines the distance metric. By default, D is optimized at each step of the Fitted-Q algorithm by a gradient descent. But keeping D constant has some advantages, even if the mean squared error of the LWPR increase. As the training set F is presented to the incremental LWPR algorithm with the same order at each iteration of Fitted-Q, the set of receptive fields, which is built from the training set, does not change over iterations. In other words, let H be the hypothesis space, so that $\forall i, \hat{Q}_i \in H$. This property ensures a better stability of the value iteration algorithm [7]. The complexity of H is determined by the constant size of receptive fields, which is set at the initialization.

On the other hand, letting the algorithm update D ought to lead to better results in theory. Indeed, in this case, the algorithm searches $\hat{Q}$ in a higher complex space. To avoid overfitting LWPR minimizes the cost J, which is the sum of two costs. The first one is a cross validation leave-one-out cost. The second is bound to the size of receptive field. Thus the regression tries to minimize the mean squared error without having too small receptive fields – i.e. having too many receptive fields (*e.g.* one for each sample). The complexity of H is related to the number of receptive fields. When D is optimized over iterations, the initialization of D is less critical. But the penalty term, which controls the complexity of H, must be chosen adequately because it determines the complexity of H. The penalty term can be determined either from assessments of the maximal local curvature of the function to be approximated [12] or empirically.

An important point to make is that H have to be complex enough not only for estimating accurately Q^* but also each Q_i to ensure a proper convergence.

Some others less important features of LWPR need to be clarify. We treated the distance metric and all related quantities as diagonal matrices because all tested problems are simple enough to avoid using this computationally expensive feature.

Another parameter is the weight activation threshold w_{gen}. If a training example has a maximal activation below w_{gen}, a new receptive field is added. Thus w_{gen} is related to the overlap between receptive fields. The higher w_{gen}, the more receptive fields will overlap and the more numerous they will be. So, w_{gen} is bound to the complexity of H, but it also smooths $\hat{Q}$. After some empirical tries, we kept the default value of 0.2.

To update the distances metrics, we choose to use second order adaptation of learning rates by the Incremental Delta Bar Delta (IDBD) algorithm, in order to avoid tuning α – the learning rate parameter.

On-line Fitted-Q iteration

As LWPR allows incremental learning we developed an off-policy online mode Q-learning with approximation algorithm. Instead of constructing a set of inputs and outputs using on the transition training set, the Bellman operator and the previous approximation $\hat{Q_{i-1}}$, this version will update the approximation of Q at each transition. Thus, the approximation of Q is constantly improved and the training set is seen many times in the same order. The algorithm is described below:

Inputs: a set of four-tuples F and an incremental regression algorithm.

Initialization: Let $\hat{Q}$ be a function equal to zero everywhere on $S \times A$.

Repeat until stopping conditions are reached:

For each four-tuple (s_t,a_t,r_t,s_{t+1}) of F :

- Build (i, o) as :

$$i = (s_t, a_t)$$
$$o = r_t + \gamma max_{a \in A} \hat{Q}_{N-1}(s_{t+1}, a)$$

- Use the incremental regression algorithm to learn (i, o).

This algorithm should work because the LWPR algorithm uses a forgetting factor to update a local model with a new training sample. While the convergence of this algorithm is far to be established, performances could be much better. In Section V, we present some results with the on-line version.

V. Results

In this section, we present results of three versions of the algorithm combining LWPR and Fitted-Q on three problems described below. The three versions of the algorithm are: "Fitted-Q + LWPR" with a

constant distance metric (D), "Fitted-Q + LWPR" with an updated D and the "Online Fitted-Q + LWPR" with a constant D.

We focus on the role of two parameters. The first is the size of the training set expressed as the number of random trajectories generated. The initial states distribution and the maximum length of a random trajectory is precised for each problem. The second stands for the complexity of the hypothesis space. In the case of D is not updated, the complexity of H is set by the distance metric. In the other case, the complexity is controlled by the penalty term (P in eq. (10)).

A. Problems tested

In order to assess the performance of our algorithm, we have run tests on three different problems.

The mountain car problem in which a car with limited power is held between two mountains. The goal is to find a policy that helps the car reach the top of one mountain by moving backward and forward repetitively in order to gain enough energy to reach the top. The state is composed of the position and the speed of the car between the mountains and there are three possible actions: moving forward, moving backward and doing nothing. State of the art shows a lot different versions of this problem [1]. In our version, the car has to reach the right mountain which corresponds to the position 0.5 with a positive speed. The maximum speed limit is 0.07, the minimum is -0.07. The position is between -1.2 and 0.5. Speed and position are updated by the following equations,

$$s_{t+1} = s_t + 0.001 * a - 0.0025 * \cos(3 * p_t),$$
$$p_{t+1} = p_t + s_{t+1},$$

where a can take the value -1 for moving backward, 0 for doing nothing and 1 for moving forward.

The inverted pendulum problem is the classic problem of the inverted pendulum laying on a cart (see [5]). The states are the angular positions and speed of the pendulum, and the algorithm seeks to find a policy which assures the stability of the system by choosing at each step one of the actions: move right, move left and do nothing.

The Cartesian product of several problems is the combination of two or more problems. We defined the noise problem as a problem with an unique action and a one-dimension state space. After each action, the state change randomly. As we normalized the state space before the learning, the standard deviation of the state space has no influence. The Cartesian product of two problems is defined by a states space equals to the cartesian product of the states spaces of the two subproblems and likewise for the actions. The combination of easy problems with noise problems, let us to analyze the ability of LWPR through the PLS to eliminate irrelevant dimensions. We resolved the problem of one pendulum with one to four noise components, the states space is the product of the two states spaces and likewise for the actions.

B. Inverted pendulum

For this problem we set the maximum length of trajectories to 3000 steps. Initial states for producing random trajectories of the training set and for computing evaluation criteria are randomly generated near the equilibrium according to a uniform distribution. The support of this distribution is [-1e-3;1e-3]

We used the inverted pendulum problem as a first test bench for each version of the algorithm. However, the inverted pendulum is too easy to solve and is not suitable for assessing performances.

Anyway, we give some results with the three versions of the algorithm in Table I. On ten runs, we have counted how many of them have terminated in less than 200 iterations of Fitted-Q with a policy keeping the pendulum in equilibrium during 3000 steps.

TABLE I
NUMBER OF SUCCESSFUL RUNS ON TEN

Size of the training set	50	100	200
LWPR + Fitted-Q constant D	0	0	6
LWPR + Fitted-Q D learning	0	2	5
Online version	2	5	9

The version of the algorithm which updates D at each iteration showed some stability problems. In fact, the algorithm converges really fast in the beginning and then diverges. More explicitly, at each iteration we computed the mean length of trajectories by starting from a random states and then by following the current policy. In maximum ten iterations, the algorithm reaches a very good policy – the mean length of trajectories is equal to the max length. And then, during next iterations, performances of the current policy decrease. Moreover, the performances of the policy seems really sensitive to the training set and to the set of parameters.

The online version is really performing well for this simple problem but this version shown some troubles with more complex problems as the mountain car (see Section VI).

C. Mountain car

For this problem we set the maximum length of trajectories to 200 steps for the training set and to 2000 for calculating evaluation criteria. Initial states for producing random trajectories of the training set and for computing evaluation criteria are randomly generated according to an uniform distribution covering all permitted speeds and positions.

We mainly conducted benchmarks on the mountain car problem because its complexity let to scale the performance of each version of the algorithm with different parameters. Results are summarized in plots. To mesure performances of the "Fitted-Q + LWPR" with a constant D algorithm, for each couple of parameters, we run eight simulations and then computes the means for each evaluation criteria. For the "Fitted-Q + LWPR" with an updated D, we run only two simulations for each couple of parameters.

1) Evaluation criteria

There are many methods to evaluate the performances of an algorithm that have been used in the literature, we considered two of them, a policy evaluation criterion and the Bellman residual.

Policy evaluation The policy evaluation criterion computes the average time for a policy to reach the solution. It is the expectation of the number n of steps needed to reach the goal without exceeding a maximum number of steps, *E[n]*. This score does not take into account inconclusive trajectories – *i.e* trajectories that do not reach the goal without exceeding the maximum number of steps. We approximate the expectation by generating a random set of starting states independently from the training set, with the same distribution than the one used to generate the training set. The size of the evaluation set is 500. Results are shown on Figures 1 and 2. One can see that learning the size of the receptive fields induces instability of the results. Anyway, the number of steps to solve the problem is very satisfying (only 40 to 60 steps which is really low compared to state of the art [13], [14]) Notice that the scales are not identical between the two graphs.

Bellman residual The Bellman residual (BR) criterion as described in [7] is defined as the difference between the two sides of the Bellman equation (eq. (6)), the Q-function being the only function

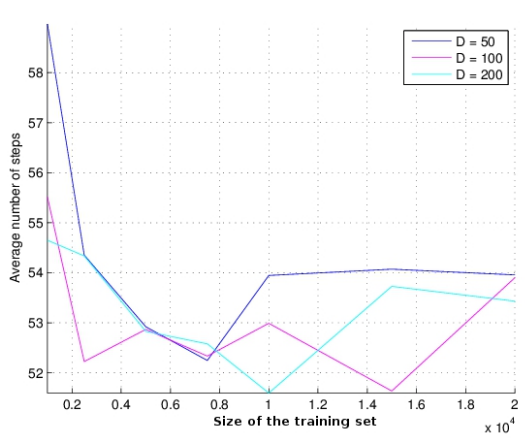

Fig. 1. Constant D : Expectation of the length of trajectories function of the training set size

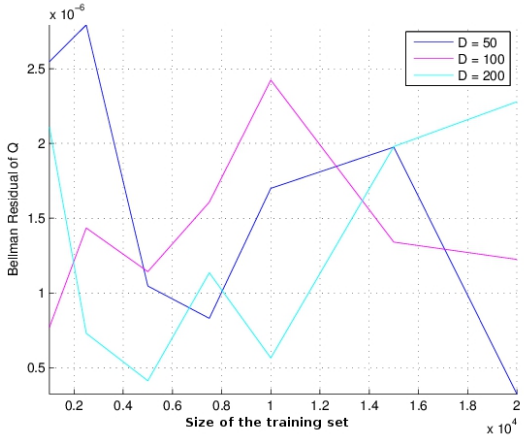

Fig. 3. Constant D : Evolution of the Bellman residual function of the training set size

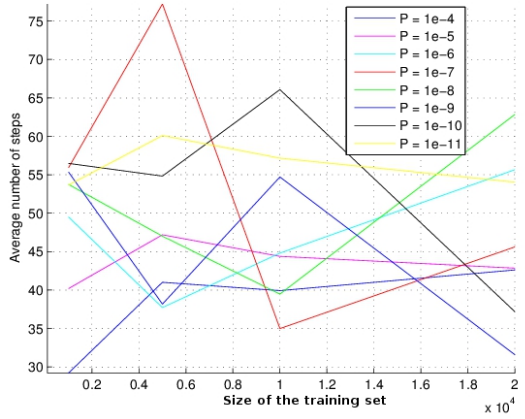

Fig. 2. Learning D : Expectation of the length of trajectories function of the training set size

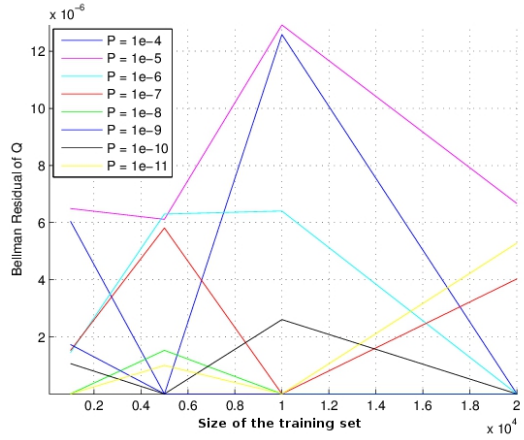

Fig. 4. Learning D : Evolution of the Bellman residual function of the training set size

leading to a zero Bellman residual for every state-action pair. To estimate the quality of a function $\hat{Q}$, we exploit the Bellman residual concept by associating to $\hat{Q}$ the mean square of the Bellman residual over the set $S \times A$, value that will be referred to as the Bellman residual (BR) of $\hat{Q}$. In order to estimate the Bellman residual, we use an independently generated set of state-action pairs. States are generated with the same distribution than the one used to generate the training set. Actions are generated with an uniform distribution.

$$\text{BR of } \hat{Q} = \frac{\sum_{(s,a)\in(S\times A)}(\hat{Q}(s,a) - (H\hat{Q})(s,a))^2}{\#(S \times A)} \qquad (11)$$

Results are shown on Figures 3 and 4. Here again, instability is brought by the learning of the receptive fields shape. Yet, the Bellman residual's value is around 10^{-6} which can be considered as very good relatively to the state of the art given the number of samples (in [7] the BR is around .15).

D. Inverted pendulum with noise

To assess performances of the combination of Fitted-Q with LWPR in environment with irrelevant inputs, we compute the average length of trajectories following the policy returned after 300 iterations of LWPR + Fitted-Q with a constant D learning with a training set composed of 5000 random trajectories starting from a random state. In our experiments, $D = 5 * I$ and the evaluation set is composed of 500 starting states. For both learning and evaluation set, starting states are generated near the equilibrium as mention for the inverted pendulum without noise. In order to be independent of the training set, we averaged over four runs with different training sets and evaluation sets. Results are given in the Table II.

TABLE II
POLICY EVALUATION ON INVERTED PENDULUM WITH NOISE

Problems	Average length of trajectories
One pendulum with three noise problems	1791
One pendulum with four noise problems	1757

This is a quite hard problem in RL because it is a 4 (resp. 5) dimensional problem in which only one is useful [9]. Our algorithm manages to solve the problem and a deeper analysis shows that only relevant dimensions are selected by LWPR to control the pendulum.

VI. CONCLUSION AND FUTURE WORKS

The main issue in reinforcement learning is the gap between theoretical results, performances on toy problems and applications on real life due to the curse of dimensionality and noises. These

problems have motived the uses of LWPR, which is well-known to be efficient in robotics where the high dimensional inputs contains a lot of noise, or irrelevant dimensions, and redundant informations. In this paper, we have shown that an approach combining Fitted-Q and LWPR works well and leads to competitive results on toy problems with irrelevant inputs and on high dimension toy problems. Combining Fitted-Q and LWPR has raised many decisions that we tried to assess by a good understanding of the workings of LWPR. We especially discuss the complexity and the stability of the hypothesis space throughout the iteration of Fitted-Q.

Our experiments show that this approach can be exploited with good results. The basic version of the algorithm without updating the Distance matrix have competitive performances. On two toy problems – inverted pendulum and mountain car – the algorithm is able to learn a very good policy from a small learning set of random trajectories. The scalability of the algorithm has also been demonstrated by our experiments on cross problems.

Experiments with a Distance matrix updated at each iteration have pointed out some stability issues. We think that instability is caused by changes in the structure of the approximation model during executions. Finally, runs with the online version of the algorithm can lead to good results in a impressive small number of iterations. But on complex problem which required a important number of iterations in order to propagate the reward from the final state to other initial states, like the mountain car problem, the algorithm may tend to converge to a very bad solution. Actually, the algorithm tend to approximate the Q-function by a plane. We believe that this problem is caused by competition between the two learning phenomena. The first is achieved by LWPR. The second is provided by Fitted-Q.

But it remains a lot of leads to explore in order to improve the performance of our implementation.

As the initialization of D, and P when D is updated, is crucial for the convergence. We thought about using an heuristic to find a good one. This heuristic would be based on the mean distance between each pairs of successive states of the training transitions set. In the case where D is kept constant, by modifying the LWPR algorithm, we can set the value of D for each receptive fields. For example, the size of each new receptive field could be initialized to a slightly larger area than the mean area covered between two successive states of which one of them is close to the center of the new receptive field. When D is updated, we can still use the heuristic described above for the initialization of new receptive fields during the first iteration of Fitted-Q. For each next iteration, the size of each new receptive field could be initialized with the one of the nearest receptive field in the model of the previous iteration.

In future works, it would be interesting to find a theoretical proof of convergence, at least in the case where D is not updated. This could be help to understand stability issues when D is updated. If problems of stability are too important, when D is updated, fixing the structure of receptive fields, beyond certain number of iteration of Fitted-Q, could be a good idea.

As evoked previously, we chose to not regress on action dimensions in case of discrete actions. For continuous action, there are some possibilities. The first one is to include action dimensions in the regression. But as explained before, the PLS could misunderstand the importance of action dimensions during the dimension reduction.

An another one is to use clustering in actions space. We could then apply the previous method on each class and employ another regression algorithm to interpolate between classes.

The last idea that could be tested is to present to the regression the learning data many times in a random order. Indeed, even if it increases the computation time, this method is advised in [10]in case of small learning sets.

After some researches and improvements it could be very interesting to assess the performance of the combination between Fitted-Q and LWPR, on more complex problems with redundant dimension and on real problems, for instance in robotics, because this approach as well as the on-line version seem very promising.

VII. Acknowledgement

The authors want to thank the European Fund for Regional Development for funding within the framework of the INTERREG IV A Allegro Project as well as the Région Lorraine.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 3rd ed. The MIT Press, March 1998.

[2] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[3] R. Bellman and S. Dreyfus, "Functional approximation and dynamic programming," *Mathematical Tables and Other Aids to Computation*, vol. 13, pp. 247–251, 1959.

[4] M. Geist and O. Pietquin, "Parametric Value Function Approximation: a Unified View," in *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2011)*, Paris (France), April 2011, pp. 9 – 16.

[5] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, December 2003.

[6] S. J. Bradtke and A. G. Barto, "Linear Least-Squares algorithms for temporal difference learning," *Machine Learning*, vol. 22, no. 1-3, pp. 33–57, 1996.

[7] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, December 2005.

[8] G. Gordon, "Stable Function Approximation in Dynamic Programming," in *Proceedings of the International Conference on Machine Learning (ICML), Bonn, Germany*, 1995.

[9] A. Nouri and M. Littman, "Dimension reduction and its application to model-based exploration in continuous spaces," *Machine Learning*, vol. 81, pp. 85–98, 2010.

[10] S. Vijayakumar, A. D'souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural Comput.*, vol. 17, pp. 2602–2634, December 2005.

[11] H. Wold, *Multivariate Analysis*. New York: Academic Press, 1966, ch. Estimation of principal components and related models by iterative least squares, pp. 391–420.

[12] S. Schaal and C. G. Atkeson, "Receptive Field Weighted Regression," ATR Human Information Processing Laboratories, Tech. Rep. TR-H-209, 1997.

[13] M. Geist, O. Pietquin, and G. Fricout, "Bayesian reward filtering," in *Recent Advances in Reinforcement Learning*, S. Girgin and colleagues, Eds. Springer Verlag, 2008, vol. 5323, pp. 96–109, revised and selected papers of EWRL 2008.

[14] ——, "Tracking in reinforcement learning," in *Proceedings of the 16th International Conference on Neural Information Processing (ICONIP 2009)*, vol. 5863, Part I. Springer LNCS, 2009, pp. 502–511, ENNS best student paper award.