HAL
archives-ouvertes.fr

Sparse Temporal Difference Learning using LASSO

Manuel Loth, Manuel Davy, Philippe Preux

▶ **To cite this version:**

Manuel Loth, Manuel Davy, Philippe Preux. Sparse Temporal Difference Learning using LASSO. IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, Apr 2007, Hawaï, USA, United States. 2007. <inria-00117075>

HAL Id: inria-00117075
https://hal.inria.fr/inria-00117075

Submitted on 30 Nov 2006

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.

Sparse temporal difference learning using LASSO

Manuel Loth
SequeL, INRIA-Futurs
LIFL, CNRS
University of Lille, France
Email: manuel.loth@inria.fr

Manuel Davy
SequeL, INRIA-Futurs
LAGIS, CNRS
Ecole Centrale de Lille, France
Email: manuel.davy@inria.fr

Philippe Preux
SequeL, INRIA-Futurs
LIFL, CNRS
University of Lille, France
Email: philippe.preux@inria.fr

***Abstract*— We consider the problem of on-line value function estimation in reinforcement learning. We concentrate on the function approximator to use. To try to break the curse of dimensionality, we focus on non parametric function approximators. We propose to fit the use of kernels into the temporal difference algorithms by using regression via the LASSO. We introduce the equi-gradient descent algorithm (EGD) which is a direct adaptation of the one recently introduced in the LARS algorithm family for solving the LASSO. We advocate our choice of the EGD as a judicious algorithm for these tasks. We present the EGD algorithm in details as well as some experimental results. We insist on the qualities of the EGD for reinforcement learning.**

I. Introduction

Whether value-function based, or direct policy search based, the approximation of a real function is a key component of reinforcement learning algorithms. To this date, various approaches have dealt with that point which fall into two broad categories, either parametric, or non parametric. Parametric means that we aim at approximating a certain function f by $\hat{f} = g(\sum_i \beta_i \phi_i)$ where the ϕ's are given *a priori* (the β's are real weights to be adjusted/learned); furthermore, when g is the identity function, the approximation is said to be linear; most parametric approximators are linear, a noteworthy exception being neural networks using a non linear activation function. In non parametric approximations, the ϕ's are defined on the fly, that is, while learning is being performed.

Most of the time, parametric approaches have been used: tiling, CMAC, and radial basis function (RBF) networks [15], [13], Gaussian processes [5], neural networks [14], [3] are well-known. Parametric approaches suffer from the fact that basis functions are set *a priori* in the state space so that there is no guarantee that they are set where they are really needed; this eventually leads to a large number of basis functions being used, while only a small number of them would be enough for a good approximation. One very attractive property of parametric approaches is that they are often amenable to a formal analysis of their capabilities, such as the convergence of the algorithm.

There are also non parametric approaches. [11] is one of the earliest attempt in the field. More recently, there has been several efforts in this direction, variable resolution grids [9], [10], locally weighted regression [1], gaussian processes [5] and sparse distributed memories [12] are well-known. The keypoint here is to obtain a sparse approximation of the function; the drawback is that we generally lose formal proofs of convergence but we have experimental evidences that this approach is appealing.

However, the reinforcement learning (RL) problem is not a pure regression problem: the data to learn from are not (observation, response) couples. In RL, the response is the return following an action and we do not want to learn the return function. Furthermore, in RL, we have to learn on-line and we do not expect the set of all "examples" to be available at once: indeed, the agent has to act and to learn to act while acting. Another noteworthy point is that there is no lack of data samples; to the opposite, we typically face millions of data points to learn from. That leads to serious computational costs.

In this paper, we are interested in non parametric approximation of the value function, being performed on-line. We consider non parametric rather than parametric approaches because we want sparse solutions. The method relies on minimizing a cost operator, the *Least Absolute Shrinkage and Selection Operator (LASSO)*, which is made of two terms, the error term (E) and the regularization (reg) term, the two being combined by way of a regularization constant: $E + \lambda$reg. λ lets us tune the importance of sparsity w.r.t. the error. This minimization was only approximated by costly heuristics until [4] proposed an algorithm that computes the entire path of regularization while keeping the computational cost very reasonnable [4]; this algorithm has been initially used for variable selection, and then for regression [7]. We wish to use this algorithm as a function approximator in RL problems.

Section 2 presents this algorithm into a renewed guise, in the framework of regression. We provide a simpler interpretation and proof of its behavior w.r.t. the LASSO and emphasize the relations between this algorithm and the classical scheme of gradient descent.

Section 3 introduces kernel versions of three notorious temporal difference algorithms, namely TD(λ), Least-Squares TD, and residual-gradient TD. These kernelizations are achieved by emphasizing the relations between TD(λ) and the gradient descent scheme, and providing a way to ensure sparsity through a sequence of independant equi-gradient descents.

Section 4 briefly states the benefits awaited from these algorithms, and section 5 shows some experimental evidences of these benefits.

II. LASSO

A. Linear function approximation

Linear approximation consists in estimating a function $f : \mathcal{X} \to \mathbb{R}$ ($\mathcal{X}$ being an arbitrary set) as a linear combination of pseudo-variates:
$\mathcal{X}$ is mapped into an m-dimensional space φ by a set of m fixed basis functions $\{\phi_i : \mathcal{X} \to \mathbb{R}\}$, and the search space $\mathcal{H}$ for the estimate is restricted to linear functions over φ:
$\hat{f}(\boldsymbol{\omega}, x) = \sum_{i=1}^{m} \omega_i \phi_i(x)$

This restriction permits the use of simple and convergent algorithms. However, φ has to be chosen so that the best choice for $\hat{f}$ in $\mathcal{H}$ is sufficiently close to f, with respect to both the empirical and real risk. It has been shown (Barron) that for any choice of m fixed basis functions, the error of the approximation has a worst-case lower bound in $O((\frac{1}{m})^{\frac{1}{d}})$ where d is the dimension of $\mathcal{X}$. This emphasizes the necessity of using non-parametric methods when $\mathcal{X}$ is high-dimensional.

Among them, kernel methods escape this issue by using the representer theorem: given a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, a high (typically infinite)-dimensional space φ is used, corresponding to the following infinitely dense grid: $\{\phi = k(x, \cdot), x \in \mathcal{X}\}$, and the representer theorem asserts that the best choice for $\hat{f}$ w.r.t. the empirical risk, given samples on $\{x_1, \ldots, x_n\}$, uses only the basis functions $k(x_1, \cdot), \ldots, k(x_n, \cdot)$.

Achieving even more sparsity over the basis functions is useful, not only for computational issues, but especially for reducing the real risk by avoiding overfitting. This has been studied in the regularization theory ([17]). SVM regression achieves sparsity by means equivalent to adding a regularization term to a loss function ([6]). Gaussian process regression treats sparsity at the same level as the representer theorem: considering only the sample points, independently of the sampled values ([5]).

B. LASSO

The Least Absolute Shrinkage and Selection Operator (LASSO, [16]) aims at characterizing a linear function $\varphi \to \mathbb{R}$ that both reduces an empirical risk and is sparse, with respect to a value $\lambda \in \mathcal{R}^+$ that sets the relative importance given to these two criterions. The basis of φ can be fixed arbitrarily, using the representer set of a kernel, or the union of such sets for several kernels, or any finite set of features $\{\phi_i, i \in 1, \ldots, m\}$.

Let us note:

- $\mathbf{x} = (x_1, \ldots, x_n)^\mathsf{T}$ the vector of sample points.
- $\mathbf{y} = (y_1, \ldots, y_n)^\mathsf{T}$ the vector of sampled values of f at these points.
- $\phi : \mathcal{X} \to \varphi, \quad \phi(x) = \big(\phi_1(x), \ldots, \phi_m(x)\big)^\mathsf{T}$
- $\hat{f}(\boldsymbol{\omega}, x) = \phi(x)^\mathsf{T}\boldsymbol{\omega}$
- $\hat{y} = \big(\hat{f}(x_1), \ldots, \hat{f}(x_1)\big)^\mathsf{T} = \boldsymbol{\Phi}\boldsymbol{\omega}$, with $\boldsymbol{\Phi} = \big(\phi(x_1), \ldots, \phi(x_n)\big)^\mathsf{T}$

Let us consider the squared-loss function for minimizing the empirical risk. The sparsity of $\hat{f}$ can be constrained by minimizing its pseudo-L1 norm: the L1 norm of $\boldsymbol{\omega} = \sum_{i=1}^{m} |\omega_i|$ [4].

For a given compromise parameter λ, the problem can be formalized by the LASSO equation:

$$\boldsymbol{\omega}^* = \arg\min \quad \|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}\|_2^2 + \lambda \sum_{i=1}^{m} |\omega_i| \tag{1}$$

The weights on each feature are equally penalized: a weight ω on any feature increases the regularized loss function by $\lambda\omega$, regardless of how much it decreases it through the squared residual. So, to do a "fair" regularization (without arbitrarily penalizing some features more than others), the features should have a similar effect on $\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}\|_2^2$, which can be achieved by scaling each feature ϕ by $\big(\sum_{i=1}^{n} \phi(x_i)^2\big)^{-\frac{1}{2}}$. The scale factors can also be determined analytically as $\int_{\mathcal{X}} \phi(x)^2 dx$.

(1) cannot be solved straightforwardly, mainly because the regularization term $\sum_{i=1}^{m} |\omega_i|$ is not differentiable. However, justifications and connections between several heuristic regression algorithms were studied in [4], and it was shown that a slight modification of a basis pursuit algorithm could recursively and exacltly solve the LASSO. The recursion is done on λ and computes the Pareto front of this dual optimization problem, from $\lambda = +\infty$ to $\lambda = 0$. One major benefit is that the choice for λ does not have to be made *a priori*, or by some cross-validation procedure; it is done on the fly, considering relevant informations like the empirical loss or the number of features used. This does not come at a high cost, as will be shown below.

The family of algorithms studied in [4] is known under the name *LARS*, for Least Angle Regression Stagewise/laSSo. In the next subsection, the recursive LASSO procedure is presented. A demonstration that is simpler and more concise than the original one is provided. The following subsection exposes a practical algorithm and considerations on its complexity.

C. Solving the LASSO by a recursion over λ

Let us consider the Pareto front, or *regularization path* Ω, that is the set of solutions for all possible values of λ:

$$\Omega = \{\boldsymbol{\omega}_\lambda = \arg\min_{\omega} \quad \|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}\|_2^2 + \lambda \sum_{i=1}^{m} |\omega_i|, \quad \lambda \in \mathbb{R}^+\}$$

There exists λ_0 such that if $\lambda > \lambda_0$, the solution of (1) consists in $\boldsymbol{\omega} = \mathbf{0}$: any weight ω_i on any feature ϕ_i would increase the regularization term more than it would reduce the loss.

Let us divide the path into the largest intervals in which the solutions have a constant sign: $\{\lambda_0, \ldots, \lambda_p = 0\}$ such that

$$\begin{cases} \bullet\ \forall i \in 0, \ldots, \lambda_{p-1}, \\ \quad \begin{cases} \lambda_i \geq \lambda_{i+1} \\ \forall \lambda, \lambda' \in]\lambda_i, \lambda_{i+1}[^2, \mathbf{sgn}(\boldsymbol{\omega}_\lambda) = \mathbf{sgn}(\boldsymbol{\omega}_{\lambda'}) \end{cases} \\ \bullet\ p \text{ is minimum} \end{cases}$$

(1) being convex w.r.t. both $\boldsymbol{\omega}$ and λ, this path of solutions is continuous: all components of $\boldsymbol{\omega}$ are continuous w.r.t. λ. So contiguous intervals $]\lambda_{i-1}, \lambda_i[$ and $]\lambda_i, \lambda_{i+1}[$ differ only on a single component of $\boldsymbol{\omega}$: either it has a non-zero value

in the first interval and is zeroed at λ_i (and beyond) after a continuous decrease (de-activation of a feature), or it is zero in $[\lambda_{i-1}, \lambda_{i+1}]$ and non-zero in $]\lambda_i, \lambda_{i+1}[$ (activation). In the cases —of probability 0— where activation/de-activation of several features occur at the same point, the intervals have a length 0: $\lambda_i = \lambda_{i+1}$.

In these open intervals, the sign vector of $\boldsymbol{\omega}$ being known, one can:

- prune the problem of inactive features ($sgn(\omega_i) = 0$): if they are not involved in the solution, they might as well have never existed. Notations will remain the same for the pruned vectors and matrices.
- solve (1), using the fact that $\sum |\omega_i|$ is differentiable w.r.t. the pruned $\boldsymbol{\omega}$ in the interval.

One can actually consider the closed intervals, for the function sgn is still differentiable if it takes a zero value at a bound of the interval.

The sequence (λ_i) and the signs of the elements of $\boldsymbol{\omega}$ is recursively determined as follows:

start of the recursion:
$[\lambda_0, \lambda_1]$ involves only one feature ϕ_i, which has a weight $\omega_i = 0$ at λ_0. This weight satisfies

$$\frac{\partial \left\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}\right\|_2^2}{\partial \omega_i} + \lambda \frac{\partial |\omega_i|}{\partial \omega_i} = 0$$

At λ_0, this gives $\quad \boldsymbol{\phi_i}^\mathsf{T}\mathbf{y} = \lambda_0 sgn(\omega_i)$.
Note that here, $sgn(\omega_i)$ is the sign that ω_i has in the open interval: this equation only generalizes what stands in the open interval to its lower bound.

So λ_0 and i are given by $\begin{cases} \lambda_0 = \max_j |\boldsymbol{\phi_j}^\mathsf{T}\mathbf{y}| \\ i = \arg\max_j |\boldsymbol{\phi_j}^\mathsf{T}\mathbf{y}| \end{cases}$

recursion:
Let us suppose that λ_j and $\mathbf{s} = \mathrm{sgn}(\boldsymbol{\omega})$ in $]\lambda_j, \lambda_{j+1}[$ are known, as well as the solution at λ_j ($\boldsymbol{\omega}^{(\lambda_j)}$). Let us solve (1) for λ in the interval, using the variables $d\lambda = \lambda_j - \lambda$ and $d\boldsymbol{\omega} = \boldsymbol{\omega}^{(\lambda)} - \boldsymbol{\omega}^{(\lambda_j)}$:

$$\frac{\partial \left\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}\right\|_2^2}{\partial \boldsymbol{\omega}} + \lambda \frac{\partial \sum_i |\omega_i|}{\partial \boldsymbol{\omega}} = 0$$

$$\boldsymbol{\Phi}^\mathsf{T}\left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}^{(\lambda_j)} - \boldsymbol{\Phi} d\boldsymbol{\omega}\right) = \lambda_j \mathbf{s} - d\lambda \mathbf{s}$$

$$\underbrace{\boldsymbol{\Phi}^\mathsf{T}\left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}^{(\lambda_j)}\right) - \lambda_j \mathbf{s}}_{0} - \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} d\boldsymbol{\omega} = d\lambda \mathbf{s}$$

$$\boldsymbol{\omega}^{(\lambda)} = \boldsymbol{\omega}^{(\lambda_j)} + (\lambda_j - \lambda) \underbrace{\left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\right)^{-1}\mathbf{s}}_{\mathbf{w}}$$

This indicates that the solutions in the interval are linear w.r.t. the decrease of λ. The direction of the change of $\boldsymbol{\omega}$ is $\mathbf{w} = \left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\right)^{-1}\mathbf{s}$ and the factor is $(\lambda_j - \lambda)$. This allows to compute the point λ_{i+1} easily:

It is the first point where whether one weight is zeroed, and by definition another interval begins, or (1) admits a solution involving one more feature, in which case the pruning is not valid anymore and this feature gets activated in the next interval.

An active feature ϕ_i is de-activated if:

$$d\lambda = \frac{-\omega_i^{(\lambda_j)}}{w_i}$$

An inactive feature ϕ_i is activated if, as well as for the active features, the gradient of the LASSO loss function w.r.t. ω_i equals 0. Again, the (non-zero) sign of ω_i in $]\lambda_{j+1}, \lambda_{j+2}[$ is considered and generalized to the bound λ_{j+1}, at which ω_i is still 0. Let us note this sign s_i.

$$\boldsymbol{\phi}_i^\mathsf{T}\left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}^{(\lambda_j)} - \boldsymbol{\Phi} d\boldsymbol{\omega}\right) = \lambda_j s_i + d\lambda s_i$$

$$\iff \boldsymbol{\phi}_i^\mathsf{T}\left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}^{(\lambda_j)}\right) - d\lambda \boldsymbol{\phi}_i^\mathsf{T}\boldsymbol{\Phi}\mathbf{w} = \lambda_j s_i + d\lambda s_i$$

$$\iff d\lambda = \frac{\boldsymbol{\phi}_i^\mathsf{T}\left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}^{(\lambda_j)}\right) - s\lambda_j}{\boldsymbol{\phi}_i^\mathsf{T}\boldsymbol{\Phi}\mathbf{w} + s}$$

The objective being to find $\lambda_{j+1} \leq \lambda_j$, λ_{j+1} is given by $\lambda_j - d\lambda$ with $d\lambda$ being the least positive or zero of the above quantities:

$$\left\{\begin{array}{l} \text{if } \phi_i \text{ active then } \frac{-\omega_i^{(\lambda_j)}}{w_i} \\ \text{else } \frac{\boldsymbol{\phi}_i^\mathsf{T}\left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}^{(\lambda_j)}\right) - s\lambda_j}{\boldsymbol{\phi}_i^\mathsf{T}\boldsymbol{\Phi}\mathbf{w} + s} \text{ with } s = \pm 1, \\ i \in 1, \ldots, m \end{array}\right\}$$

Two restrictions must be made to that set: a feature that has just been activated at λ_j must not be considered for de-activation, and one that has just been de-activated must not be considered for activation: they would candidate for an immediate change of their status, being at the frontier of two intervals where they have different status.

This gives both λ_{j+1} and the change of $\mathbf{sgn}(\boldsymbol{\omega})$: $\mathrm{sgn}(\omega_i)$ either becomes 0, or goes from 0 to 1, or from 0 to -1.

D. The equi-gradient descent algorithm

Let us first note the similarities and differences between the gradient descent method and the method exposed above, which is therefore baptized here *equi-gradient descent*.

Gradient descent consists in a sequence of steps in which each weight is modified proportionally to its gradient on the residual. In the linear regression problem, each step (of rank i) consists in:

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha_i \boldsymbol{\Phi}^\mathsf{T}\left(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}\right)$$

If the α_i's are sufficiently small and decreasing, each change of weights approximates:

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \arg\min_{\boldsymbol{\delta\omega}} \left\|\mathbf{y} - \boldsymbol{\Phi}(\boldsymbol{\omega} + \boldsymbol{\delta\omega})\right\|_2^2 + \frac{1}{\alpha_i}\sum_{i=1}^{m}(\delta\omega_i)^2$$

and the sequence asymptotically minimizes:

$$\left\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}\right\|_2^2 + \lambda \sum_{i=1}^{m}(\omega_i)^2$$

with $\lambda_0 \leq \lambda \leq \lim_{i\to\infty} \frac{1}{\alpha_i}$, λ_0 being the threshold beyond which

$$\arg\min \|\mathbf{y} - \mathbf{\Phi}\boldsymbol{\omega}\|_2^2 + \lambda \sum_{i=1}^{m} (\omega_i)^2 = \arg\min \|\mathbf{y} - \mathbf{\Phi}\boldsymbol{\omega}\|_2^2$$

The equi-gradient descent only modifies weights that have the highest gradient on the residual. This modification is made in the direction that would lead the closest to the target, in contrast to its approximation in gradient descent. This direction has the property to keep the gradients equal, and allows the analytical computation of the length of the step: it stops at the point where a new feature has the same –highest– gradient on the residual as the active ones.

The practical algorithm is exposed in Alg. 1. The complexity of an iteration of the loop is O(nm): the most complex operation is the $\arg\min$ of two functions of two dot-products ($O(n)$), in the set of inactive features ($O(m)$); the update of $\left(\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi}\right)^{-1}$ is only $O(na)$ where a is the number of active features.

The number of iterations has been empirically observed to be $O(a^2)$, a being the final number of active features when stopping the equi-gradient descent. A semi-formal explanation holds in the following facts:

- L1 regularization being strongly correlated to sparsity, the number of actives features is quasi-monotonous throughout the iterations,
- a configuration (the set of active features) can only occur in a single iteration,
- the number of selected configurations of size p is probably logarithmic in the number of possible configurations (2^p), and thus $O(p)$, which makes the number of iterations $O(\sum_{p=1}^{a} p) = O(a^2)$.

III. Using equi-gradient descent in temporal difference learning

A. Sparse Least Squares TD

The Least-Squares TD algorithm ([2]) is a policy evaluation scheme. Its principle is to directly solve the system of Bellman equations on a set of samples obtained either from trajectories or in any other way. The system is solved by minimizing the sum of the squared Bellman residuals:

Let $\hat{v}_\theta$ be the parametric approximator of the value function of the policy to be evaluated.

Let $s_1, \ldots, s_n$ be the sampled states and $\mathbf{v}_\theta = (\hat{v}_\theta(s_1), \ldots, \hat{v}_\theta(s_1))^\mathsf{T}$

Let $\mathbf{B}$ be the Bellman matrix connecting states related to each other by a Bellman equation; for example, if states come from a single trajectory and a fixed discount factor γ is used:

$$\mathbf{B} = \begin{bmatrix} 1 & -\gamma & & \mathbf{0} \\ & 1 & -\gamma & \\ & & & \ddots \\ \mathbf{0} & & & \end{bmatrix}$$

The vector of Bellman residuals is $\mathbf{r} - \mathbf{B}\hat{\mathbf{v}}$ where $\mathbf{r}$ is the vector of rewards sampled between connected states. LSTD computes $\arg\min_\theta \|\mathbf{r} - \mathbf{B}\hat{\mathbf{v}}_\theta\|_2^2$

Algorithm 1: Equi-gradient descent

for i = *1* **to** m **do** $\boldsymbol{\phi_i} \leftarrow (\phi_i(x_1), \ldots, \phi_i(x_n))^\mathsf{T}$
$\mathbf{res} \leftarrow (y_1, \ldots, y_n)^\mathsf{T}$
$\mathbf{s} \leftarrow ()$; $\boldsymbol{\omega} \leftarrow ()$; $\mathbf{\Phi} \leftarrow []$
$\boldsymbol{\phi} \leftarrow \arg\max_{\boldsymbol{\phi}_i} \left|\boldsymbol{\phi_i}^\mathsf{T}\mathbf{res}\right|$
$\lambda \leftarrow \boldsymbol{\phi}^\mathsf{T}\mathbf{res}$
$s \leftarrow \mathrm{sgn}(\boldsymbol{\phi}^\mathsf{T}\mathbf{res})$
$todo \leftarrow$ activate ϕ with sign s
while not *stopping criterion* **and not** *todo*=done **do**
  switch $todo$ **do**
    case activate ϕ with sign s
      $\mathbf{\Phi} \leftarrow \left[\begin{array}{c|c} \mathbf{\Phi} & \boldsymbol{\phi} \end{array}\right]$
      $\mathbf{s} \leftarrow \begin{pmatrix} \mathbf{s} \\ s \end{pmatrix}$
      $\boldsymbol{\omega} \leftarrow \begin{pmatrix} \boldsymbol{\omega} \\ 0 \end{pmatrix}$
    case de-activate j-th active feature
      remove j-th element of $\mathbf{\Phi}, \boldsymbol{\omega}, \mathbf{s}$
  end
[1] $d\boldsymbol{\omega} \leftarrow \left(\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi}\right)^{-1}\mathbf{s}$
  $d\mathbf{res} \leftarrow \mathbf{\Phi}\boldsymbol{\delta\omega}$
[2] [3] $(d\lambda_+, \boldsymbol{\phi}_+) \leftarrow (\min, \arg\min)_{\boldsymbol{\phi}\text{ inactive}} \left[\frac{\lambda - \boldsymbol{\phi}^\mathsf{T}\mathbf{res}}{1 - \boldsymbol{\phi}^\mathsf{T} d\mathbf{res}}\right]_{\geq 0}$
  $(d\lambda_-, \boldsymbol{\phi}_-) \leftarrow (\min, \arg\min)_{\boldsymbol{\phi}\text{ inactive}} \left[\frac{\lambda + \boldsymbol{\phi}^\mathsf{T}\mathbf{res}}{1 + \boldsymbol{\phi}^\mathsf{T} d\mathbf{res}}\right]_{\geq 0}$
  $(d\lambda_0, j) \leftarrow (\min, \arg\min)_{j\in 1\ldots\text{nb act. features}} \left[\frac{-\omega_j}{d\omega_j}\right]_{\geq 0}$
  $d\lambda \leftarrow \min(d\lambda_+, d\lambda_-, d\lambda_0)$
[2] **if** $d\lambda$ *undefined* **then**
    $todo \leftarrow$ done
    $d\lambda \leftarrow \lambda$
  else if $d\lambda = d\lambda_+$ **then**
    $todo \leftarrow$ activate $\boldsymbol{\phi}_+$ with sign +1
  else if $d\lambda = d\lambda_-$ **then**
    $todo \leftarrow$ activate $\boldsymbol{\phi}_-$ with sign -1
  else if $d\lambda = d\lambda_0$ **then**
    $todo \leftarrow$ de-activate j-th active feature
  $\lambda \leftarrow \lambda - d\lambda$
  $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + d\lambda * d\boldsymbol{\omega}$
  $\mathbf{res} \leftarrow \mathbf{res} - d\lambda * d\mathbf{res}$
end

1: $\left(\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi}\right)^{-1}$ should not be computed at each iteration of the loop, but stored and modified iteratively, using block matrix inversion.

2: $[expression]_{\geq 0}$ means that only positive values are considered, which means that $\arg\min_x [f(x)]_{\geq 0}$ is undefined if $f(x)$ takes no positive value.

3: As explained in II.C, the $\min - \arg\min$ searches must not consider the feature that has just been [de]activated.

In the case of linear approximators (including kernel methods) where $\hat{\mathbf{v}}_\omega = \mathbf{\Phi\omega}$, the application of equi-gradient descent is immediate: it can be used as is on the following LASSO problem:
minimize $\arg\min_\theta \|\mathbf{r} - \mathbf{\Phi'\omega}\|_2^2 + \lambda \sum_i \omega_i$ with $\mathbf{\Phi'} = \mathbf{B\Phi}$.

The main benefit is to obtain a sparse solution, which both saves computational time and avoids over-fitting. One can then use dense grids or kernel features more easily.

B. Kernel TD(λ)

TD(λ), when used for evaluating a fixed policy, uses the following scheme: considering an estimation $\hat{v}_\theta$ of v and a trajectory $s_0 \xrightarrow{r_1} s_1 \dots \xrightarrow{r_n} s_n$,

$$\begin{aligned} & v(s_0) = r_1 + \gamma v(s_1) \\ \iff \quad & v(s_0) - \hat{v}_\theta(s_0) = r_1 - \hat{v}_\theta(s_0) + \gamma v(s_1) \end{aligned}$$

Assuming that $\hat{v}_\theta(s_1) \simeq v(s_1)$, the residual at $s_0 = v(s_0) - \hat{v}_\theta(s_0)$ is estimated by $r_1 - \hat{v}_\theta(s_0) + \gamma\hat{v}_\theta(s_1)$ (principle of *value iteration*).
The same reasoning is applied to estimate it by

$$\begin{aligned} & r_1 + \gamma r_2 + \gamma^2 \hat{v}_\theta(s_2) - \hat{v}_\theta(s_0) \\ \text{or} \quad & r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 \hat{v}_\theta(s_3) - \hat{v}_\theta(s_0) \\ \dots & \end{aligned}$$

These estimates are averaged with coefficients such that the final estimate is:

$$v(s_0) - \hat{v}_\theta(s_0) \simeq \sum_i^n (\lambda\gamma)^{i-1} \left(r_i + \gamma\hat{v}_\theta(s_i) - \hat{v}_\theta(s_{i-1})\right)$$

$v(s_i) - \hat{v}_\theta(s_i)$ are estimated the same way for $i \in 2 \dots n-1$. Using a matricial formulation, TD(λ) considers that:

$$\mathbf{res} = \mathbf{v} - \hat{\mathbf{v}}_\theta \simeq \mathbf{L}\left(\mathbf{r} - \mathbf{B}\hat{\mathbf{v}}_\theta\right), \quad \text{with } \mathbf{L} = \begin{bmatrix} 1 & \lambda\gamma & (\lambda\gamma)^2 & \dots \\ & 1 & \lambda\gamma & \dots \\ & & 1 & \dots \\ \mathbf{0} & & & \ddots \end{bmatrix}$$

and performs the following update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \hat{\mathbf{v}}_\theta}{\partial \theta} \mathbf{L}\left(\mathbf{r} - \mathbf{B}\hat{\mathbf{v}}_\theta\right)$$

which is equivalent to considering $\widehat{\mathbf{res}} = \mathbf{L}\left(\mathbf{r} - \mathbf{B}\hat{\mathbf{v}}_\theta\right)$ as a (fixed) estimation of the residual at the points sampled from a trajectory, and performing a gradient descent step to minimize $\|\widehat{\boldsymbol{\delta}\hat{\mathbf{v}}_\theta} - \boldsymbol{\delta}\hat{\mathbf{v}}_\theta\|_2^2$. The form of $\widehat{\mathbf{res}}$ allows to perform the update sequentially, using the computational trick of eligibility traces.

TD(λ) is used for policy improvement by evaluating a moving policy – greedy w.r.t. $\hat{\mathbf{v}}_\theta$. The difference is that the estimated error is $\widehat{\mathbf{v}^{\pi'} - \hat{\mathbf{v}}_\theta^\pi}$, using the asumption that $\hat{v}_\theta^\pi(s) \simeq v_\theta^{\pi'}(s)$, where π and π' are respectively the previous and current policies.

This algorithm and derivates like Q-Learning and SARSA have been widely used with linear approximators. It has also been used with success with neural networks, by propagating the updates on linear hyper-parameters to the non-linear ones by backpropagation. The use of kernel methods seems more problematic, since the non-linear parameters are not numeric: they consist in the choice of kernel centers (sampled points). Given the previous consideration on how TD(λ) relates to regression by gradient descent, an algorithm is proposed in the following that permits the use of TD(λ) with a kernel approximator.

The principle of TD(λ) on linear approximators can be summarized the following way: after a sampled trajectory, a piece of regression is made to approximate the estimated residual $\widehat{\mathbf{v} - \mathbf{\Phi\omega}}$ by a corrective term $\mathbf{\Phi\delta\omega}$. $\hat{\mathbf{v}}(x) = \sum_i \omega_i\phi_i(x)$ is then updated to $\hat{\mathbf{v}}'(x) = \sum_i \omega_i\phi_i(x) + \sum_i \delta\omega_i\phi_i(x)$. The linear parameters (weights) remain the same (in the sense that they apply to the same feature set) throughout the successive gradient descent steps.

The key idea is to replace the gradient descent step performed after each trajectory by an equi-gradient descent.

The direct application of this would be to perform after each trajectory an equi-gradient descent that approximates $\widehat{\mathbf{res}} = \widehat{\mathbf{v} - \mathbf{\Phi\omega}}$ by $\mathbf{\Phi'\omega'}$, the features in $\mathbf{\Phi'}$ being a kernel centered on a selection of p visited states. $\hat{\mathbf{v}}(x) = \sum_{i=1}^m \omega_i\phi_i(x)$ would then be updated to $\hat{\mathbf{v}}'(x) = \sum_{i=1}^m \omega_i\phi_i(x) + \sum_{i=m+1}^{m+p} \omega_i\phi_i(x)$ (new features being indexed). This scheme could be used with any kernel method, and more generally any regression method. The trivial drawback is that although feature selection may be accurate in each regression, no pertinent selection is made throughout the whole algorithm: features just add up through the –possibly very long– sequence of trajectories.

To provide "global" feature selection, the first key is to include features that occur in the current estimate $\hat{v}$ in the set of candidates for approximating the residual. The approximation of the residual can then use these features (as linear TD(λ) does) as well as new features centered on visited states. This can be done straightforwardly with equi-gradient descent, for it considers any arbitrary set of candidates features.

Global sparsity would still not be satisfyingly ensured, because new features would remain more attractive than the others, as they are centered on the considered states. Again, the flexibility of equi-gradient descent provides a solution: since it penalizes the use of features through the sum of their weights, one can penalize the new features more than the others by including coefficients in the penalization term. The LASSO problem is rewritten as:

$$\boldsymbol{\omega}^* = \arg\min \quad \|\mathbf{y} - \mathbf{\Phi\omega}\|_2^2 + \lambda \sum_{i=1}^m \alpha_i |\omega_i|$$

The change induced in the equi-gradient algorithm is just to replace the signs $s = \pm 1$ by $\alpha_i s$.

The last point is that when the equi-gradient descent gets the final weight of a feature already occuring in $\hat{v}$ to zero, it should be taken into account by explicitely de-activating it.

Concerning the stopping criterion, performing only one step does not seem interesting, as it would just select one feature each time. One natural possibility is to stop when the residual has been reduced by a given ratio. Empirically, a ratio of 0.8, ie. $|\mathbf{L}(\mathbf{r} - \mathbf{B\Phi\omega}) - \mathbf{\Phi'\omega'}| < 0.8|\mathbf{L}(\mathbf{r} - \mathbf{B\Phi\omega})|$, seems to ensure good and stable performances.

A kernel TD(λ) algorithm is summarized in Alg. 2. It could be used with any kernel method as long as it allows to include the kernel centered on arbitary points as additional candidates and to penalize them less than the "natural" ones.

Algorithm 2: Equi-gradient kernel TD(λ)

```
/* v̂ is defined by a set of weighted
   features {(ω_i, φ_i)}                    */
```
$\hat{v} \leftarrow \{\}$

repeat

perform a trajectory $s_0 \xrightarrow{r_1} s_1 \dots s_n$ using a greedy policy

perform an equi-gradient descent with

- target $\mathbf{L}(\mathbf{r} - \mathbf{B}\hat{\mathbf{v}})$
- candidate features the ones used in $\hat{\mathbf{v}}$ (penalized with $\alpha < 1$) and $\{k(s_i, \cdot)\}$ penalized with $\alpha = 1$
- stopping criterion: the reduction of the residual by a given ratio

include the result in $\hat{v}$

- remove features with weight taken to 0
- modify weights of features used in the previous estimate and modified by the descent
- add new weighted features

until $\hat{v}$ *stationary*

C. Kernel residual-gradient TD

As reminded above, TD(λ) uses averages of temporal differences to estimate a fixed residual. The Bellman equations relating sampled states are used to estimate independant targets on each state and then forgotten. One can see this as an empirical way to mix all 2^n Bellman correlations between states in order to estimate v (as opposed to value iteration which only focuses on correlations between states and their successor).

An alternate way is to directly aim at solving the system of Bellman equations, which expresses all correlations. This is the scheme used in LSTD and in *residual-gradient TD* ([8]), as opposed to the original *value-gradient* TD(λ). Let us consider a simple example, with a 3 states trajectory. The Bellman system is:

$$\begin{cases} r_1 + v_0 - \gamma v_1 = 0 \\ r_2 + v_1 - \gamma v_2 = 0 \end{cases}$$

Value iteration (TD(0)) approximately solves this by

$$\begin{cases} r_1 + v_0 - \gamma \hat{v}_1 = 0 \\ r_2 + v_1 - \gamma \hat{v}_2 = 0 \end{cases}$$

TD(λ) adds the combined equation:

$$\begin{cases} r_1 + v_0 - \gamma v_1 = 0 \\ r_1 + \gamma r_2 + v_0 - \gamma^2 v_2 = 0 \\ r_2 + v_1 - \gamma v_2 = 0 \end{cases}$$

It then approximates the system by

$$\begin{cases} r_1 + v_0 - \gamma \hat{v}_1 = 0 \\ r_1 + \gamma r_2 + v_0 - \gamma^2 \hat{v}_2 = 0 \\ r_2 + v_1 - \gamma \hat{v}_2 = 0 \end{cases}$$

and finally estimates v_0 as a linear combination of the solutions of the first two equations.

The residual-gradient TD aims at solving the system directly. The problem of its uncompleteness, which is somehow the motivation for the TD(λ) scheme, is escaped by the implicit regularization in gradient descent, which implicitely treats the current estimate $\hat{v}$ as a Bayesian prior for the next.

The practical difference is that, as one attempts to directly minimize $\|\mathbf{r} - \mathbf{B}\hat{\mathbf{v}}_\theta\|_2^2$, the gradient descent steps consists in $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \frac{\partial \mathbf{B}\hat{\mathbf{v}}_\theta}{\partial \boldsymbol{\theta}} (\mathbf{r} - \mathbf{B}\hat{\mathbf{v}}_\theta)$. A possible drawback is that this being an approximation of

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \frac{\partial \mathbf{B}\hat{\mathbf{v}}_{\theta'}}{\partial \boldsymbol{\theta}'} (\mathbf{r} - \mathbf{B}\hat{\mathbf{v}}_{\theta'})$$

it may be less accurate than the one made in TD(λ) because the Bellman operator increases the variance.

Using the same principle as in kernel TD(λ), one can perform after each episode an equi-gradient descent on the following LASSO:

$$\text{minimize } \|\mathbf{r} - \mathbf{B}(\boldsymbol{\Phi}\boldsymbol{\omega} + \boldsymbol{\Phi}'\boldsymbol{\omega}')\|_2^2 + \lambda \sum_i \alpha_i |\omega_i'|$$

where $\boldsymbol{\Phi}'$ includes both the features in $\boldsymbol{\Phi}$ and the new ones centered on sampled states, and the α_i's penalize new features more than the other ones.

IV. Benefits

The most obvious benefits of the algorithms exposed above are the ones associated with kernel methods: they provide advantages of linear approximators while being non-parametric, thus less subject to the curse of dimensionality. Other benefits appear that are related to the genericity of the LASSO formulation and the precision of its resolution by equi-gradient descent.

A. precision

Gradient descent minimizes

$$\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\omega}\|_2^2 + \lambda \sum_{i=1}^{m} (\omega_i)^2$$

by means of numerous small steps where the derivate of the loss function is considered constant. In TD(λ), only one step is performed per trajectory or, turning this another way, the residual is witnessed on new points after each step. This explains the so-called "waste of samples" in this algorithm.

In equi-gradient descent, the exact derivate is used, benefiting from its piecewise linearity, and the lengths of the steps are determined exactly. This allows to go further in the updates in a safe way, and is especially appealing when using residual gradient.

B. genericity

Unlike traditional kernel regression methods, equi-gradient descent does not rely on the Mercer or positive-definiteness properties of a single kernel. It just considers an arbitrary finite set φ of candidate features. This does not mean it misses some properties of such a kernel: when used on the feature

set $\{k(x_0,\cdot),\ldots,k(x_n,\cdot)\}$, it performs the same task: find a suitable compromise between data-fitting and sparsity, this task being achieved in very similar ways.

This allows the use of multiple kernels, typically Gaussian kernels with various bandwidths. One can also use kernels on projections of the state space on some of its dimensions: if $s=(s^{(x)},s^{(y)})^\mathsf{T}\in\mathbb{R}^2$, φ can include $k(s_0^{(x)},\cdot),k(s_0^{(y)},\cdot),\ldots$.

Another interesting possibility is to use kernels that exploits the symmetry in the dynamics and value function of the MDP: if the state space $\mathcal{S}=\mathbb{R}^d$ and it is known that $\forall s, v(s)=v(-s)$, a kernel $k_{sym}(x,x')=k(x,x')+k(x,-x')$ extends the neighborhood of a point w.r.t. the approximated function to the neighborhood of its opposite. Such a kernel causes serious perturbations around 0 in a grid-based TD(λ) or in Gaussian Process TD, which is not the case with equi-gradient TD. Note that there is no other satisfying way of exploiting such a symmetry; projecting on half the state space does not preserve continuity across the separating hyperplan.

V. Experiments

Experiments were run on the inverted pendulum problem, as described in [3]. 100 independent learning sessions of 300 trajectories were run each time, the trajectories consisting in 40 transitions of $0.1s$. The evolution of the quality of the value function was estimated by running 100 off-learning trajectories and adding received rewards. The same initial states were used for each experiment. The results are presented in Fig.'s 1-5, and are detailed below.

A. kernel-TD(λ)

influence of the penalization coefficient α: We first used a Gaussian kernel of variance 0.15 (the state space being normalized), and compared choices for the penalization coefficient α of features already used in the previous estimate. It has the expected influence on the number of actives features: as α decreases, the number of active features naturally decreases, which is helpful to tune the computation time.

The convergence speed and quality are overall similar, but it should be noted that the quality is slightly better with $\alpha=0.67$, see Fig. 1.

multi kernels: We then used 3 Gaussian kernels of variances 0.15, 0.17 and 0.2. Convergence is a bit faster, and the number of active features does not change, see Fig. 2.

symmetry: We then exploited the central symmetry of the problem $\big(V(x)=V(-x)\big)$ by using symmetric features: Instead of $\phi_i(x)=k(x_i,x)$, we set $\phi_i(x)=k(x_i,x)+k(x_i,-x)$. This trick is not applicable in either TD(λ) with gradient descent nor Gaussian Process TD, as it creates divergences around 0. With Equi-gradient TD, some momentaneous divergences appear when using multi kernels, but it is robust with a single kernel. The benefits are a faster convergence and the use of less features, as can be seen in Fig. 3.

comparison with TD(λ) on a grid: We compared the results obtained by the following algorithms:

- parametric TD(λ) with gradient descent on a 15×15 grid of Gaussian bases.

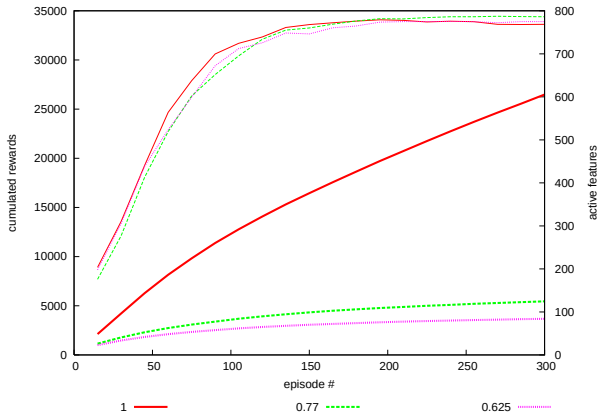

Fig. 1. Influence of α on the number of active features (3 thick lines), and on the cumulated reward (3 thin lines), for three values of α. We use a single Gaussian kernel of variance 0.15.

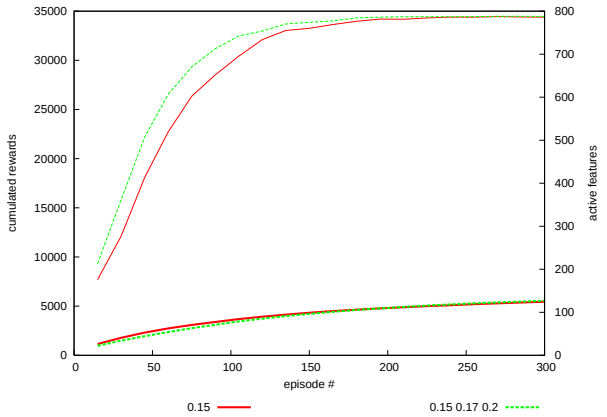

Fig. 2. This plot compares the performance of kernel-TD(λ) using a single kernel function (in red), and using 3 kernel functions (in green). All kernels are Gaussian; they differ from their variance.

- Equi-gradient kernel TD(λ) with the same single kernel and use of symmetry.

The later shows both fast convergence and better policies. See Fig. 4

B. Kernel residual gradient TD(λ)

Experiments were run to compare *kernel TD(λ)* with the recently developped *kernel residual gradient TD*. Three observations are noteworthy:

- The convergence on the pendulum problem is even faster,
- the stopping ratio is best set around 0.9 to avoid some momentaneaous divergences in the policy,
- contrary to the value-gradient version, the evolution of the number of features shows a fine asymptotic shape when running a large number of trajectories, as shown in Fig. 5.

VI. Summary and future work

We have formulated variants of TD algorithms in which the gradient descents are replaced by what we called an equi-gradient descent. The first takes an approximatively good

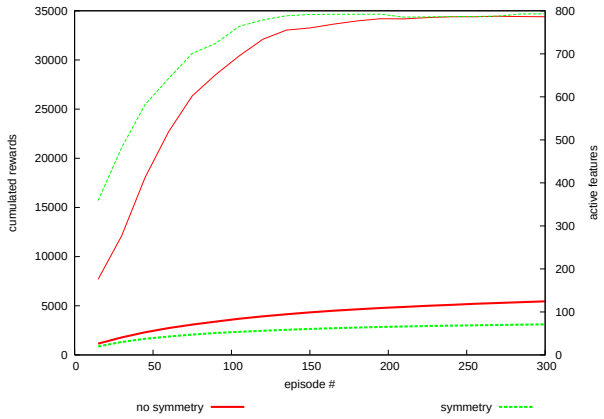

Fig. 3. This plots exhibits the difference that is observed when one uses the symmetry of the problem (in green), or not (in red). The algorithm uses a single Gaussian kernel function of variance 0.15.

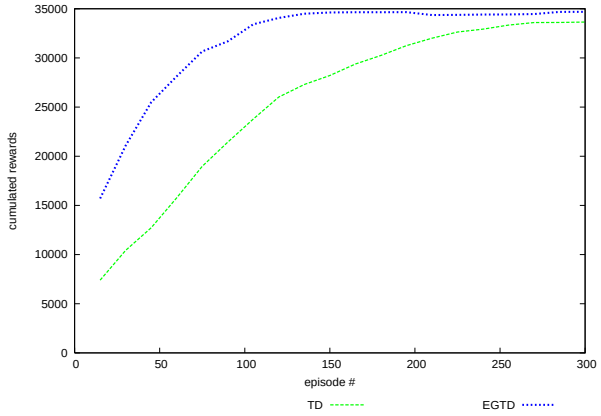

Fig. 4. Evolution of cumulated rewards of grid-based TD(λ) and kernel-TD(λ). One can see the very good performance of the later.

direction on all parameters, whereas the second takes a succession of optimal directions on the most correlated parameters, including more and more of them on the way. This allows to use TD on a non-parametric basis function network, where bases are smartly selected in a large set of candidates, with various centers, shapes and ranges of effect.

Good results in terms of quality, fast convergence, and computation complexity have been obtained on the inverted pendulum problem.

Future work will focus on ways to adapt these algorithms to high-dimensional problems, including

- the use of kernels defined on all possible selections of the original variables/dimensions. This means an exponential growth of the number of candidate features, which may be handled by a possible stochastic version of the EGD algorithm.
- the definition of fine exploration schemes. Exploration may be improved by following the evolution of the policy throughout the equi-gradient steps.

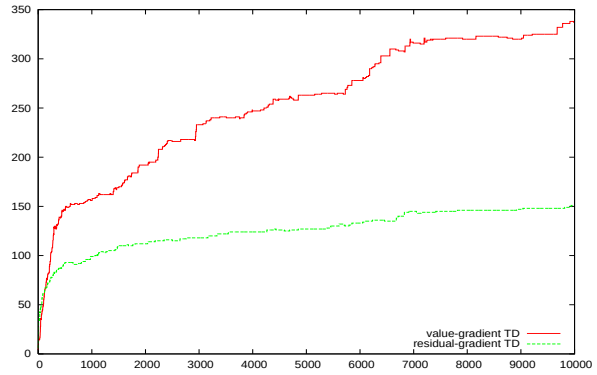

Fig. 5. Evolution of the number of features throughout the learning trajectories in both kernel TD(0.5) and kernel value-gradient TD.

ACKNOWLEDGEMENTS

The first author gratefully acknowledges the support from *INRIA-Futurs* and *Region Nord - Pas-de-Calais*.

REFERENCES

[1] Chris Atkeson, Andrew Moore, and Stefan Schaal. Locally weighted learning for control. *AI Review*, 11:75–113, April 1997.

[2] J. Boyan. Least-squares temporal difference learning. In *Proc. 16th International Conference on Machine Learning*, pages 49–56. Morgan Kaufmann, San Francisco, CA, 1999.

[3] R. Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[4] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least-angle regression. *Annals of statistics*, 32(2):407–499, 2004.

[5] Y. Engel. *Algorithms and Representations for Reinforcement Learning*. PhD thesis, Hebrew University, April 2005.

[6] F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural computation*, 10(6):1455–1480, 1998.

[7] V. Guigue. *Méthodes à noyaux pour la représentation et la discrimination de signaux non-stationnaires*. PhD thesis, Institut National des Sciences Appliquées de Rouen, 2005.

[8] Leemon C. Baird III. Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, pages 30–37, 1995.

[9] Andrew Moore and Chris Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21, 1995.

[10] Remi Munos and Andrew Moore. Variable resolution discretizations for high-accuracy solutions of optimal control problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, Stockholm*, pages 1348–1355, 1999.

[11] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, (2):213–225, 1992.

[12] B. Ratitch and D. Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *Proc. of the 15^{th} European Conference on Machine Learning*, 2004.

[13] R. Sutton. Generalization in reinforcement learning: successful examples using sparse coarse coding. In *Proc. NIPS*, pages 1038–1044, 1996.

[14] G. Tesauro. Temporal difference learning and TD-Gammon. *Comm. of the ACM*, 38:58–68, 1995.

[15] C.K. Tham. *Modular on-line function approximation for scaling up reinforcement learning*. PhD thesis, Cambridge University, October 1994.

[16] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal Statistics*, 58(1):267–288, 1996.

[17] A. Tikhonov and V. Arsénin. *Solutions of ill-posed problems*. W.H. Winston, Washington D.C., 1977.