Equi-gradient Temporal Difference Learning

Manuel Loth MANUEL.LOTH@ETUDIANT.UNIV-LILLE1.FR
LIFL (UMR CNRS) & INRIA-Futurs, Villeneuve d'Ascq, France

Manuel Davy MANUEL.DAVY@EC-LILLE.FR
LAGIS (UMR CNRS) & INRIA-Futurs, Villeneuve d'Ascq, France

Rémi Coulom REMI.COULOM@UNIV-LILLE3.FR
LIFL (UMR CNRS) & INRIA-Futurs, Villeneuve d'Ascq, France

Philippe Preux PHILIPPE.PREUX@UNIV-LILLE3.FR
LIFL (UMR CNRS) & INRIA-Futurs, Villeneuve d'Ascq, France

Abstract

During the last few years, kernel methods have been a popular area of research in machine learning, providing a number of efficient and popular algorithms like SVMs. They have not been much investigated in the field of continuous reinforcement learning (RL) however, the main reason being the off-line nature of kernel algorithms, whereas RL needs to update estimates iteratively. The most notable contribution yet has been that of (Engel et al., 2005), who uses Gaussian processes. Engel's method is of the same nature as Least-Squares TD, in that it considers every observation from the start of the learning session. It has been shown that this can lead to convergence issues, contrary to TD which performs updates conditioned on the last observations only. We have taken a wide point of view on kernel methods as methods of regression with non-parametric basis-functions-networks, and introduce a variation of TD that uses the recently introduced Least-Angle Regression Stepwise (LARS) method in place of the gradient descent step. In this context, the LARS is viewed and renamed as an *equi-gradient descent*. This allows to approximate the value function with a self-adaptative basis functions network, instead of a fixed, parametric one. Our algorithm combines the low complexity and good convergence properties of parametric TD(λ) with the precision and efficient use of the data of kernel methods.

Preliminary work. Under review for the kernel machines for reinforcement learning workshop, on behalf of the International Conference on Machine Learning (ICML). Do not distribute.

1. Introduction

We address the approximation of the value function for temporal difference methods in reinforcement learning problems. Various tilings methods have been investigated – see (Sutton & Barto, 1998) for instance –, as well as neural networks which have provided very nice results both for discrete problems (Tesauro, 1995), and for continuous problems (Coulom, 2002). In the same time, there has been a frenzy around support vector machines and, more generally, kernel machines, mostly in the field of supervised learning (classification and regression problems). These two threads eventually mixed these last years with the works by (Ormoneit & Sen, 2002; Lagoudakis & Parr, 2003; Ratitch B., 2004; Engel et al., 2005). Using kernel methods for reinforcement learning is not straightforward, though, and several difficulties have to be dealt with:

- putting the kernel methods into the reinforcement learning problem framework
- making the kernel methods able to work on-line
- making them able to do it efficiently, at low computational costs.

In this paper, we propose to use the Least-Angle Regression Stepwise (LARS) algorithm which has been originally proposed in a regression context, without kernelization (Efron et al., 2004). The LARS has then

been kernelized, and it has demonstrated very good performance for feature selection (Guigue, 2005). We formulate the LARS in a slightly more general way, renaming it an *equi-gradient descent*, and use it to learn the value function of a continuous Markov decision problem, within the reinforcement learning framework. Equi-gradient descent has attractive properties to this end, and early experimental results show its effictiveness and efficiency.

This paper is organized as follows: Section 2 defines the notation used throughout the paper; Section 3 presents the equi-gradient descent method for regression. Then, Section 4 presents the original point of this paper, that is, the way we use it in the TD(λ) algorithm; Section 5 then provides an experimental assessment of the LATD(λ). The paper closes on a summary and discussion of future work on this algorithm.

2. TD Learning

Let us consider the following family of continuous Markov decision processes (Sutton & Barto, 1998):

- A dynamical system is observed at discrete fixed time intervals $t \in \{0, 1, \ldots\}$
- It is described by its state x_t which belongs to a continuous space $\mathcal{X}$.
- At each time t, an action u_t chosen in a finite set $\mathcal{U}$ is applied to the system, which state becomes $x_{t+1} = u_t(x_t)$. The dynamics of each action is deterministic and known.
- A reward r_t is associated to each transition $x_{t-1} \xrightarrow{u} x_t$

A policy π associates to each state a chosen action $\pi(x) \in \mathcal{U}$, and the problem of maximizing a discounted sum of rewards to come is considered: finding a policy that maximizes $\sum_{t=1} \gamma^t r_t$ for any start state x_0, with $\gamma \in]0;1]$. The reward function is also deterministic and known.
The value function V^π of a policy π has the following property, known as *Bellman's equation*:

$$\text{if } x \xrightarrow{\pi(x)} x'$$

$$V^\pi(x) = r\big(x, \pi(x)\big) + \gamma V^\pi(x') \qquad (1)$$

An optimal policy π^* is obtained greedily from its value function V^* :

$$\pi^*(x) = \text{argmax}_{u \in \mathcal{U}} r(x,u) + \gamma V^*\big(u(x)\big) \qquad (2)$$

(this is not true for non-optimal policies)

2.1. Optimistic policy iteration

Optimistic policy iteration (Bertsekas, 1996) consists in refining estimates of π^* and V^*, as follows:

---

initialize $\widehat{V}$
the optimistic assumption that $\widehat{V}$ estimates V^* leads to define $\widehat{\pi}$ greedily wrt. $\widehat{V}$, as in Eq. (2):

$$\widehat{\pi}(x) = \text{argmax}_{u \in \mathcal{U}}\ r(x,u) + \gamma \widehat{V}\big(u(x)\big)$$

loop
apply $\widehat{\pi}$ for a number of transitions
update $\widehat{V}$ wrt. the observed errors in Eq. (1)
(the change in $\widehat{V}$ induces a change in $\widehat{\pi}$)
end loop

The errors in Eq. (1) are the temporal differences:

$$d_t = r_t - \widehat{V}(x_{t-1}) + \gamma \widehat{V}(x_t)$$

Let us consider:

- one-episode iterations: an episode consists in applying $\widehat{\pi}$ during $n+1$ time steps (n being set in advance or corresponding to a goal being reached), from a random state x_0 to a final state x_n. Updates of $\widehat{V}$ (and consequently $\widehat{\pi}$) are done after each episode.
- approximation of V by a basis functions network (BFN):

  $$\widehat{V}(x) = \sum_i \beta_i \phi_i(x)$$

  where ϕ_i's, which we will call *features*, are functions $\mathcal{X} \to \mathbb{R}$, and β_i's, which we will call *weights*, are in $\mathbb{R}$. Let us consider both *parametric* BFN where the set of features $\{\phi_i\}$ is set in advance, and *non-parametric* BFN where this set might evolve during learning.

Let us note

- $\boldsymbol{\beta} = (\ldots, \beta_i, \ldots)^\mathsf{T}$
- given an episode $\{x_0, \ldots, x_n\}$,
  - $\mathbf{r} = (r_1, \ldots, r_n)^\mathsf{T}$
  - $\Phi_i = (\phi_i(x_0), \ldots, \phi_i(x_n))^\mathsf{T}$
  - $\boldsymbol{\Phi} = (\ldots, \Phi_i, \ldots)$
  - $\widehat{\mathbf{V}} = (\widehat{V}(x_1), \ldots, \widehat{V}(x_n))^\mathsf{T}$
  - $\mathbf{d} = (d_1, \ldots, d_n)^\mathsf{T} = \mathbf{r} - \mathbf{H}\boldsymbol{\Phi}\boldsymbol{\beta}$

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

2.2. Continuous TD(λ) and parametric BFN

The TD(λ) algorithm (Sutton, 1988) considers that the temporal difference observed at x_t is not directly related to the error made on $\widehat{V}(x_t)$. Instead, it states that this error is gradually involved in all subsequent temporal differences, and thus approximates the error at x_t by:

$$V(x_t) - \widehat{V}(x_t) \simeq \sum_{t'=t}^{n} (\lambda\gamma)^{t'-t} d_{t'} \,,$$

V being the true unknown value function of the current policy.

The estimated error vector on visited states is then:

$$\widehat{\mathbf{err}} = \widehat{\mathbf{V} - \widehat{\mathbf{V}}} = \widehat{\mathbf{V} - \mathbf{\Phi}\boldsymbol{\beta}} = \mathbf{L}(\mathbf{r} - \mathbf{H}\mathbf{\Phi}\boldsymbol{\beta})$$

$$\text{with } \mathbf{L} = \begin{bmatrix} 1 & \lambda\gamma & (\lambda\gamma)^2 & \dots \\ & 1 & \lambda\gamma & \dots \\ & & 1 & \dots \\ \mathbf{0} & & & \ddots \end{bmatrix}$$

In the TD(λ) algorithm, the correction on $\widehat{V}$ consists in changing the weights $\boldsymbol{\beta}$ as follows:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \boldsymbol{\delta}_{\boldsymbol{\beta}}$$

thus correcting the value function as:

$$\widehat{V} \leftarrow \widehat{V} + \mathbf{\Phi}\boldsymbol{\delta}_{\boldsymbol{\beta}}$$

in order to reduce the error, ie. minimize the residual:

$$\mathbf{res} = \widehat{\mathbf{err}} - \mathbf{\Phi}\boldsymbol{\delta}_{\boldsymbol{\beta}}$$

It is worth noting that $\widehat{\mathbf{err}}$ is an approximation of the direction of the error. It is not intended to be maximally reduced, but gives a good direction for a careful step.

The value-gradient method performs a gradient descent step on the squared residual $\frac{1}{2}\|\mathbf{res}\|^2$:

$$\begin{aligned} \boldsymbol{\delta}_{\boldsymbol{\beta}} &= -\alpha \frac{\partial \frac{1}{2}\|\mathbf{res}(\boldsymbol{\delta}_{\boldsymbol{\beta}})\|^2}{\partial \boldsymbol{\delta}_{\boldsymbol{\beta}}} \\ &= -\alpha \frac{\partial \mathbf{res}(\boldsymbol{\delta}_{\boldsymbol{\beta}})}{\partial \boldsymbol{\delta}_{\boldsymbol{\beta}}} \, \mathbf{res}(\mathbf{0}) \\ &= -\alpha \frac{\partial(\widehat{\mathbf{err}} - \mathbf{\Phi}\boldsymbol{\delta}_{\boldsymbol{\beta}})}{\partial \boldsymbol{\delta}_{\boldsymbol{\beta}}} \, (\widehat{\mathbf{err}} - \mathbf{\Phi}\mathbf{0}) \\ &= \alpha \mathbf{\Phi}^{\mathsf{T}} \widehat{\mathbf{err}} \end{aligned}$$

2.3. Continuous TD(λ) and non-parametric BFN

In the non-parametric case, one wishes to reduce the error not only by changing weights on "existing" features, but also by adding new features, conditioned on the observations. A simple example is the use of radial Gaussian features of a given radius σ:

$$\phi_i(x) = e^{-\frac{1}{2}\left(\frac{d(x,x_i)}{\sigma}\right)^2}$$

Each ϕ_i is centered on a state x_i. In the parametric case, $\{x_i\}$ would be a predefined set of states, like the vertices of a mesh (hyper-grid). In the non-parametric case, all visited states are potential centers for a new feature, and the most relevant ones should be picked up. More precisely, the k-th learning step would be:

- after the (k-1)-th step, we have
$$\widehat{V}_{k-1}(x) = \sum_i \beta_i \phi_i(x)$$
- perform an episode with a $\widehat{V}$-greedy policy
- estimate errors made by $\widehat{V}_{k-1}$ on visited states
- build a set $\{\phi'_j\}$ of candidate features related to visited states
- define
$$\begin{aligned} \widehat{V}_k(x) = & \sum_i \beta_i \phi_i(x) \\ & + \sum_i \delta_{\beta_i} \phi_i(x) \\ & + \sum_j \beta'_j \phi'_j(x) \end{aligned}$$

The goal is again to make the corrective term $\sum_i \delta_{\beta_i}\phi_i(x) + \sum_j \beta'_j\phi'_j(x)$ correlated with $\widehat{\mathbf{err}}$: in the same direction but with a partial step.

Here, a gradient descent step is not a good solution anymore, as one wants to use only a small and smart subset of $\{\phi'_j\}$. The method presented in the next section adresses this problem, as it provides a stepwise path from an error to its least-squares minimization. Each step gets closer to the minimization, using more and more features, in an optimal way.

3. Equi-gradient descent

In this section, the general regression problem is considered, independently of the RL framework.

Introduction

For regression problems, kernel methods aim to solve the following problem: given a sample $\{y_1, \dots, y_n\} \in \mathbb{R}^n$ of an unknown function f at points $\{x_1, \dots, x_n\} \in \mathcal{X}^n$ and a kernel function $k : \mathcal{X}^2 \to \mathbb{R}$ find an estimate $\widehat{f}(x) = \sum_{i=1}^n \beta_i k(x_i, x)$ such that $\widehat{y} = \{\widehat{f}(x_1), \dots, \widehat{f}(x_1)\}$ is "close to" $\{y_1, \dots, y_n\}$ and the number of nonzero β_i is kept small, which prevents from over-fitting the data and gives a good complexity and an expressive solution.

Kernel-based regression methods, generally use a symmetric positive-definite kernel (e.g., SVMs), which induces a reproducing kernel Hilert space (RKHS), denoted $\mathcal{H}$. The kernel reproducing property transforms the nonlinear regression problem in $\mathcal{X}$ into a linear regression problem in $\mathcal{H}$. Different from SVMs, Gaussian processes regression uses k in a way that allows to make a bayesian compromise between data-fitting and closeness to an awaited covariance.

The Least-angle regression stepwise (LARS) method, introduced by (Efron et al., 2004) as a variable selection method, and adapted by (Guigue, 2005) to kernel feature selection, uses several kernels on which no positive-definitiveness assumption is required. It can actually be seen as a method to find an estimate $\widehat{f}(x) = \sum_i \beta_i \phi_i(x)$, where ϕ_i's are numerous arbitrary functions. We present it here from a slightly different point of view than (Efron et al., 2004; Guigue, 2005), and relax a normalization condition. This leads us to rename it an *equi-gradient descent*.

Principle

The compromise between data-fitting and the number of nonzero weights can be solved by solving the following problem:

$$\text{minimize } \|\mathbf{y} - \mathbf{\Phi}\boldsymbol{\beta}\|^2 \text{ under the constraint } \sum_i |\boldsymbol{\beta}_i| < \mu$$

The method exactly solves it for all possible values of μ: at each step a solution is provided for a certain μ, which increases monotonously from 0. Each step corresponds to the activation of a new candidate feature (its weight becomes nonzero), or a de-activation of an active feature (its weight is zero).

Let $\mathcal{A} = \{a_1, a_2, \dots\} \in 2^{\mathbb{N}}$ be the active set (that is, the indices of the features $\phi_{a_1}, \phi_{a_2}, \dots$ with nonzero weight), and

$$\Phi_A = \begin{bmatrix} \phi_{a_1}(x_1) & \phi_{a_2}(x_1) & \dots \\ \phi_{a_1}(x_2) & \phi_{a_2}(x_2) & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

The principle is to activate features one by one, using the general ideas of basis pursuit. Precisely, at each step, a new feature is added, and then weights of all active features are modified to optimally reduce the current residual.

At each step, a new residual wrt. the original target is reached, and the next step purpose is, after having included a new feature, to determine the weights corrections $\boldsymbol{\delta_\beta}$ to obtain a new optimally decreasing residual.

To help understand the algorithm, one can view it in a geometrical way (see Fig. 1), in a basis where each axis concerns a learning data x_i, $\boldsymbol{y} = (y_1, \dots, y_n)^\mathsf{T}$ is the target, and

$$\boldsymbol{res}_{(k)} = \begin{pmatrix} y_1 - \widehat{f}_{(k)}(x_1) \\ \vdots \\ y_n - \widehat{f}_{(k)}(x_n) \end{pmatrix}$$

is the residual after the k-th step.

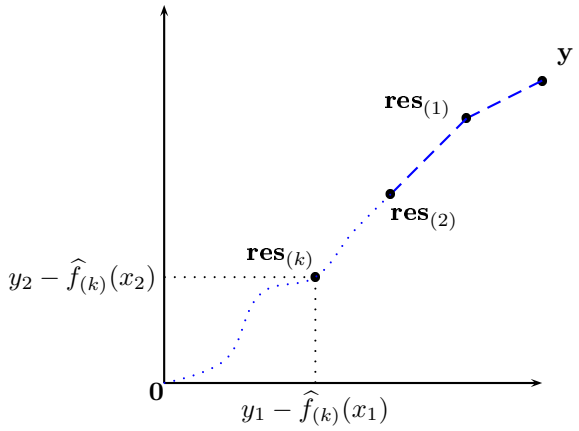

Figure 1. The residual path: we start with a target $\mathbf{y} = (y_1, y_2)$. At each step, a correction of $\widehat{f}$ is made, hence a new $\widehat{\mathbf{y}} = (\widehat{f}(x_1), \widehat{f}(x_2))$ which takes the residual $\mathbf{y} - \widehat{\mathbf{y}}$ closer to its minimum

Profitability

The criterion for including a feature is its *profitability*: the gradient of the squared residual minimization wrt. a weight correction on this feature:

$$-\frac{\partial \|\mathbf{res}_{(k)} - \Phi_i \boldsymbol{\delta_{\beta_i}}\|^2}{\partial \boldsymbol{\delta_{\beta_i}}} = {\Phi_i}^\mathsf{T} \mathbf{res}_{(k)}$$

which represents the correlation of the feature ϕ_i to the current residual, and geometrically a dot-product (see fig. 2).

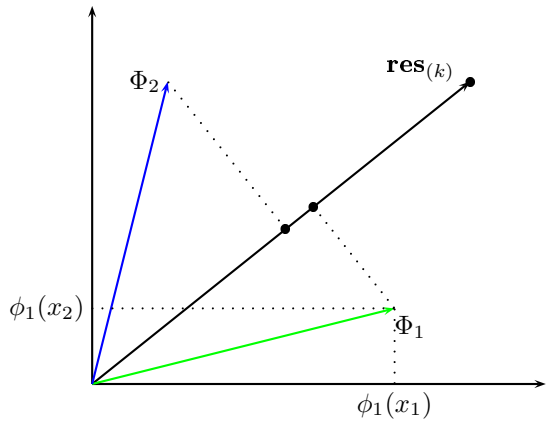

Figure 2. Gradient as a dot-product. ϕ_1 is more correlated to $\mathbf{res}_{(k)}$ than ϕ_2, as $\Phi_1 = \big(\phi_1(x_1), \phi_1(x_2)\big)$ is closer to the residual than Φ_2: it makes a least-angle, and their norms are equal, which is expressed by a higher dot-product.

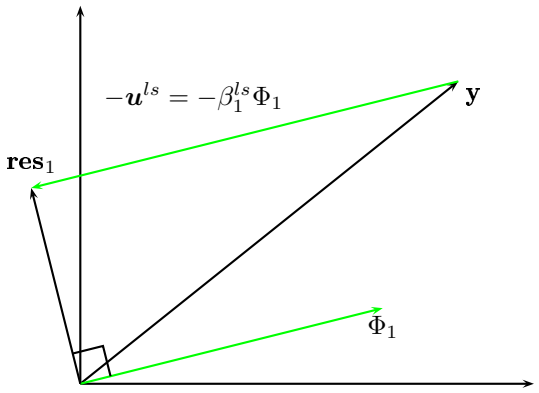

Figure 3. Least-squares minimization: for example if only ϕ_1 is active (first step), the least-squares optimal weight to assign to it is such that $\langle \Phi_1 \cdot \mathbf{res}_{(1)} \rangle = 0$ (they are orthogonal).

OPTIMAL DIRECTION

Let us consider the $(k+1)$-th step. We have a given set of active features. Let us suppose the problem should be solved with these features only. Their weights should be corrected (by $\beta_i \leftarrow \beta_i + \delta_{\beta_i}$) in order to minimize:

$$\|\mathbf{res}_{(k)} - \boldsymbol{\Phi_A}\boldsymbol{\delta_\beta}\|^2$$

To fully minimize this expression, its derivative must be zeroed:

$$\frac{\partial\|\mathbf{res}_{(k)} - \boldsymbol{\Phi_A}\boldsymbol{\delta_\beta}\|^2}{\partial\boldsymbol{\delta_\beta}} = \mathbf{0}$$

$$\Leftrightarrow \boldsymbol{\Phi_A}^\mathsf{T}(\mathbf{res}_{(k)} - \boldsymbol{\Phi_A}\boldsymbol{\delta_\beta}) = \mathbf{0}$$

$$\Leftrightarrow \boldsymbol{\delta_\beta} = (\boldsymbol{\Phi_A}^\mathsf{T}\boldsymbol{\Phi_A})^{-1}\boldsymbol{\Phi_A}^\mathsf{T}\mathbf{res}_{(k)}$$

Let us note this optimal correction of the active weights:

$$\boldsymbol{w} = \boldsymbol{\delta_\beta}^{least-squares} = (\boldsymbol{\Phi_A}^\mathsf{T}\boldsymbol{\Phi_A})^{-1}\boldsymbol{\Phi_A}^\mathsf{T}\mathbf{res}_{(k)}$$

and its application (correction of $\widehat{f}$):

$$\boldsymbol{u} = \boldsymbol{\Phi_A}\boldsymbol{w}$$

An equi-gradient step goes in that *direction*, without necessarily reaching the full least-squares solution:

$$\boldsymbol{\delta_\beta} = \alpha\, \boldsymbol{w}, \quad \text{with } \alpha \in [0,1]$$

thus reducing the residual by:

$$\mathbf{res}_{(k+1)} \leftarrow \boldsymbol{res}_{(k)} - \alpha\boldsymbol{u}$$

The choice for α will be such that after the corresponding change of weights, an inactive feature "deserves" to be included in the active set.

EQUI-CORRELATION

The key of the algorithm is that all active features will remain equi-correlated to the current residual: they all make the same gradient on it, and this correlation is superior to the one of any inactive feature. It uses the following property:
IF the active features are equi-correlated to the residual after the k-th step:

$$|\Phi_{a_1}^\mathsf{T}\mathbf{res}_{(k)}| = |\Phi_{a_2}^\mathsf{T}\mathbf{res}_{(k)}| = \dots$$

$$\text{ie. } \exists c \in \mathbb{R}^+ \text{ such that } \boldsymbol{\Phi_A}^\mathsf{T}\mathbf{res}_{(k)} = c\,\mathbf{sgn}$$

where $\mathbf{sgn}$ is the vector of signs of the correlations

THEN when going in the least-squares direction, they will remain equi-correlated, because:

$$\begin{aligned}
\mathbf{res}_{(k+1)} &= \mathbf{res}_{(k)} - \boldsymbol{\Phi_A}\boldsymbol{\delta_\beta} \\
&= \mathbf{res}_{(k)} - \alpha\boldsymbol{\Phi_A}(\boldsymbol{\Phi_A}^\mathsf{T}\boldsymbol{\Phi_A})^{-1}\boldsymbol{\Phi_A}^\mathsf{T}\mathbf{res}_{(k)} \\
&= \mathbf{res}_{(k)} - \alpha\boldsymbol{\Phi_A}(\boldsymbol{\Phi_A}^\mathsf{T}\boldsymbol{\Phi_A})^{-1}c\,\mathbf{sgn} \\
&= \mathbf{res}_{(k)} - \alpha c\,\boldsymbol{\Phi_A}(\boldsymbol{\Phi_A}^\mathsf{T}\boldsymbol{\Phi_A})^{-1}\mathbf{sgn}
\end{aligned}$$

thus

$$\begin{aligned}
\boldsymbol{\Phi_A}^\top \mathbf{res}_{(k+1)} &= \boldsymbol{\Phi_A}^\top \mathbf{res}_{(k)} \\
&\quad -\alpha c\, \boldsymbol{\Phi_A}^\top \boldsymbol{\Phi_A} (\boldsymbol{\Phi_A}^\top \boldsymbol{\Phi_A})^{-1} \mathbf{sgn} \\
&= c\, \mathbf{sgn} - \alpha c\, \mathbf{sgn} \\
&= (1-\alpha) c\, \mathbf{sgn}
\end{aligned}$$

The invariant of the algorithm is that inactive features are always less correlated to the residual than active ones. So the value of α will be chosen as the smallest such that an inactive feature gets as much correlated to the new residual as the active ones: Metaphorically, the weights are gradually corrected in the least-squares direction until this happens (see Fig. 4).

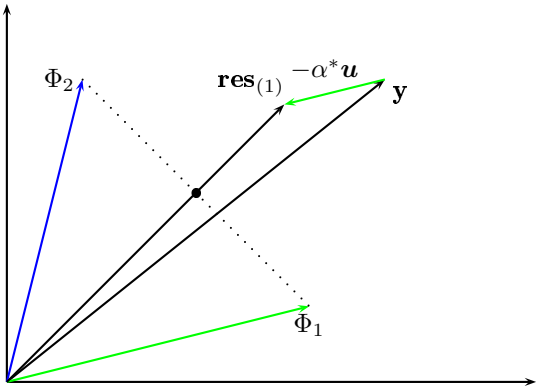

Figure 4. α^* is such that inactive feature ϕ_2 gets as much correlated to the new residual as active feature ϕ_1.

$$\alpha^* = \underset{\alpha \in [0,1]}{\arg\min} \left(\exists i \notin \mathcal{A}, |\Phi_i^\top \mathbf{res}_{k+1}(\alpha)| = (1-\alpha)c \right)$$

We have:

$$\begin{aligned}
&|\Phi_i^\top \mathbf{res}_{k+1}(\alpha)| = (1-\alpha)c \\
&\quad \Leftrightarrow \quad |\Phi_i^\top (\mathbf{res}_{(k)} - \alpha \mathbf{u})| = (1-\alpha)c \\
&\quad \Leftrightarrow \quad \left| \begin{array}{l} \Phi_i^\top \mathbf{res}_{(k)} - \alpha \Phi_i^\top \mathbf{u} = (1-\alpha)c \\ \textit{or} \\ -\Phi_i^\top \mathbf{res}_{(k)} + \alpha \Phi_i^\top \mathbf{u} = (1-\alpha)c \end{array} \right. \\
&\quad \Leftrightarrow \quad \left| \begin{array}{l} \alpha = \frac{c - \Phi_i^\top \mathbf{res}_{(k)}}{c - \Phi_i^\top \mathbf{u}} \\ \textit{or} \\ \alpha = \frac{c + \Phi_i^\top \mathbf{res}_{(k)}}{c + \Phi_i^\top \mathbf{u}} \end{array} \right.
\end{aligned}$$

And so

$$\alpha^* = \min^+ \{ \frac{c - \Phi_i^\top \mathbf{res}_{(k)}}{c - \Phi_i^\top \mathbf{u}} ; \frac{c + \Phi_i^\top \mathbf{res}_{(k)}}{c + \Phi_i^\top \mathbf{u}} \}_{\phi_i \text{ inactive}}$$

By $\min^+$, we mean the smallest positive element of the set:

$$\min^+(\mathcal{S}) = \min\{x \in \mathcal{S}, x \geq 0\}$$

Indeed, α^* must be positive to go in the good direction. The feature ϕ_i corresponding to the chosen α^* will be included in the active set $\mathcal{A}$ in the next step. The sign of its correlation depends on wether the first or the second expression was the smallest positive.

De-activation

Though it may not appear clearly in the algorithm, the weight on an active feature may first increase, and then decrease, as the least-squares direction changes from step to step. This illustrates the optimal use of weights: in the first step, the algorithm is greedy wrt. $\partial \mathbf{err}/\partial \beta_i$, which is the best solution if the threshold μ is very small; in the following, the current-residual greediness leads to discard these greedy choices in favor of more weight-costly but more error-fitting choices.

Another invariant is that the weight of every active features should keep its sign, which is the same as its correlation with the residual when it was included. This relates to the fact that $\mathrm{sgn}(\beta_j) = \partial|\beta_j|/\partial\beta_j$. So when it changes during a step, it should be removed from the active set after having taken a step back so as to strictly zero it, by letting:

$$\alpha' = \frac{-\beta_j}{\Phi_j \boldsymbol{u}}$$

Instead of noting afterwards that a change of sign occured and making this step back, α^* should be chosen as the smallest that wether activates a new feature or zeroes the weight of an active feature.

After its de-activation, a feature should remain available for a possible re-activation.

Normalization

A normalization of the variables/features is imposed in (Efron et al., 2004; Guigue, 2005). It consists in asserting that

$$\|\Phi_1\|_2 \simeq \|\Phi_2\|_2 \simeq \ldots$$

$$\text{with } \|\Phi_i\|_2 = \sqrt{\sum_t \phi_i(x_t)^2}$$

It is important in the idea of *selection* where one looks for a ranking of the variables/features, but not crucial in the idea of gradient descent.

The criterion of profitability (correlation) defined above, being a gradient wrt. the weights selects features that both "go" in a good direction (geometrically

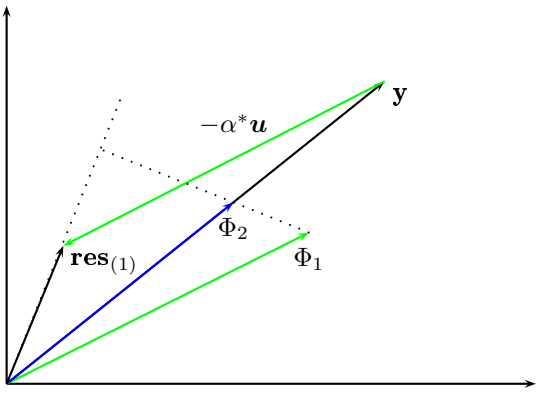

Figure 5. The first step selects ϕ_1, because the norm of Φ_1 is greater and putting a small weight on it will be more benefitable than on ϕ_2. The second and last step ($\rightarrow \mathbf{res}_{(2)} = \mathbf{0}$) will take β_1 to 0, and give $\widehat{f}(x) = \beta_2\phi_2(x)$ as it should.

form a small angle with the residual) and have a large norm. The reason is that the norms of the weights are penalized, and a feature with a larger norm requires less weight to produce an effect of the same amount. Given two features, if one has a larger angle and a larger norm, it may be selected first. However, the other will eventually be activated and the first degenerated, as illustrated in Fig. 5. But to fairly compare the candidates, they should have a similar norm.

Given candidate features ϕ_i, this can be achieved by scaling them ($\phi_i \leftarrow a_i\phi_i, \quad a_i \in \mathbb{R}$) so that their norms on the training set $\|\Phi_i\|$ are equal. One can also scale them wrt. their norms on the whole dataset ($\int_{\mathcal{X}} \phi_i(x)^2 dx$). This is more robust in certain cases, as it allows to compare features that are similar on the data space but may have different ranges of effect on a local training set: the ones whose local effect is lower are penalized, which is fair, as using them would unconsiderably change the estimate outside the local training set.

Bandwiths

One of the tendancy and advantage of equi-gradient descent is to first select features with a large bandwidth, ie. which reduce the residual on more learning points. This tendancy depends on the normalization. If the norms are equal, large bandwidth features have a notable effect on more points, but the effect on each of these points is lower than that of a small bandwidth feature. Nonetheless, the largest one tends to be selected first. If largest bandwidths come with largest norms, they will be used longer (further) before using more local and precise features.

Stopping

The algorithm converges to the least-squares solution, but its interest is the intermediate solutions, that get closer to the target while using more and more of the candidate features, in an optimal way. The question of when it should stop is left open since it depends on the problem and various considerations on the dataset, training set, estimated function, complexity of the solution... It is typically a matter of bias/variance trade-off. It can rely on the number of active features, the value of their correlation to the residual, the squared norm of the residual compared to the original target, ...

Algorithm

The previous subsections have explained the principle and properties of the equi-gradient descent. This algorithm *is* the Least-Angle Regression Stepwise, but generalized to features - and not only variables or kernels, and in which the normalization condition is relaxed. Thus the term *least-angle* is no longer justified: the highest gradient corresponds to a least angle only if all norms are equal.

To summarize, the algorithm consists in reducing the error (the residual gradually goes to its minimum) along a regularization path. Along that path, it only alters those weights that have the highest influence on the current residual. This alteration is made in the optimal direction considering only these weights.

Given a data space $\mathcal{X}$
Input:
- a vector of training points $\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T} \in \mathcal{X}^n$
- a target vector $\boldsymbol{y} = (y_1, \ldots, y_n)^\mathsf{T} \in \mathbb{R}^n$
- a set of candidate features $\{\phi_i : \mathcal{X} \rightarrow \mathbb{R}\}_{i\in\{1,\ldots,m\}}$ having equal or similar norm on $\mathcal{X}$ ($\int \phi^2$)

Output *at each step*:
- $\mathcal{A} = \{a_1, \ldots, a_l\} \subset \{1, \ldots, m\}$ and $\{\beta_{a_1}, \ldots, \beta_{a_l}\}$ such that $\|\boldsymbol{y} - \sum \beta_{a_j}\phi_{a_j}\|$ optimally decreases at each step as $\sum |\beta_i|$ increases

for $i = 1$ to m **do**
$$\Phi_i \leftarrow \begin{pmatrix} \phi_i(x_1) \\ \vdots \\ \phi_i(x_n) \end{pmatrix}$$
end for
$\boldsymbol{res} \leftarrow \boldsymbol{y}$
$i^* \leftarrow \text{argmax } |\langle \Phi_i \cdot \boldsymbol{res} \rangle|$
$c \leftarrow |\langle \Phi_{i^*} \cdot \boldsymbol{res} \rangle|$
$sgn \leftarrow \text{sgn}(\langle \Phi_{i^*} \cdot \boldsymbol{res} \rangle)$

$\mathcal{A} \leftarrow \{i^*\}$
$\boldsymbol{\beta} \leftarrow (0)$
$\boldsymbol{\Phi}_A \leftarrow [\Phi_{i^*}]$
$\boldsymbol{sgn} \leftarrow (sgn)$

repeat
  $\boldsymbol{w} \leftarrow (\boldsymbol{\Phi_A}^\mathsf{T}\boldsymbol{\Phi_A})^{-1} c\, \boldsymbol{sgn}$
  $\boldsymbol{u} \leftarrow \boldsymbol{\Phi_A}\boldsymbol{w}$
  $(\alpha_+, i_+) \leftarrow (\min^+, \mathrm{argmin}^+)_{i \notin \mathcal{A}} \dfrac{c - \Phi_i^\mathsf{T}\boldsymbol{res}}{c - \Phi_i^\mathsf{T}\mathbf{u}}$
  $(\alpha_-, i_-) \leftarrow (\min^+, \mathrm{argmin}^+)_{i \notin \mathcal{A}} \dfrac{c + \Phi_i^\mathsf{T}\boldsymbol{res}}{c + \Phi_i^\mathsf{T}\mathbf{u}}$
  $(\alpha_0, j) \leftarrow (\min^+, \mathrm{argmin}^+)_{a_j \in \mathcal{A}} \dfrac{-\beta_{a_j}}{w_j}$
  $\alpha^* = \min(\alpha_+, \alpha_-, \alpha_0)$
  $c \leftarrow (1 - \alpha^*)c$
  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha^* \boldsymbol{w}$
  $\boldsymbol{res} \leftarrow \boldsymbol{res} - \alpha^* \boldsymbol{u}$
  if $\alpha^* = \alpha_0$ **then**
    remove j-th element of $\mathcal{A}, \boldsymbol{\Phi}_A, \boldsymbol{sgn}, \boldsymbol{\beta}$
  else
    if $\alpha^* = \alpha_+$ **then**
      $i^* \leftarrow i_+; \quad sgn \leftarrow +1$
    else
      $i^* \leftarrow i_-; \quad sgn \leftarrow -1$
    end if
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{i^*\}$
    $\boldsymbol{\Phi}_A \leftarrow \left[\begin{array}{c|c} \boldsymbol{\Phi}_A & \Phi_{i^*} \end{array}\right]$
    $\boldsymbol{sgn} \leftarrow \begin{pmatrix} \boldsymbol{sgn} \\ sgn \end{pmatrix}$
    $\boldsymbol{\beta} \leftarrow \begin{pmatrix} \boldsymbol{\beta} \\ 0 \end{pmatrix}$
  end if
until stop_criterion$(\mathcal{A}, \boldsymbol{\beta}, c, \boldsymbol{res}, \ldots)$

Note: $(\boldsymbol{\Phi_A}^\mathsf{T}\boldsymbol{\Phi_A})^{-1}$ can be computed recursively, using block matrix inversion.

4. Equi-gradient TD(λ)

The integration of the equi-gradient descent in continuous TD(λ) has been explained in Section 2, but will be made more precise here.

The goal is to approximate V as a linear combination of a few relevant features related to the visited states. A natural example is to associate to each visited state several Gaussian kernels of various bandwidth centered on that state (more generally, radial features).

- At step k of the policy iteration, V is approximated as a linear combination of previously selected features:
$$\widehat{V} = \sum \beta_i \phi_i$$
- A $\widehat{V}$-greedy episode is done.
- The error made by $\widehat{V}$ on visited states is estimated.
- A set of new features $\{\phi'_j\}$ is mapped from the visited states.
- The error is considered as a target for a corrective term
$$\sum \delta_{\beta i} \phi_i + \sum \beta'_j \phi' j$$
  - This target should not be reached, for it is approximate and over-estimated; it just gives a good direction.
  - The number of nonzero β'_j should be small.
- This corrective term is determined by making an equi-gradient descent. The stopping criterion, given that a partial reduction of the error should be made, can be its reduction by a given percentage (eg. $\|\mathbf{res}\|^2 = 0.8\|\mathbf{err}\|^2$).
- nonzero δ_{β_i}'s come as a modification of previous weights β_i; nonzero β'_j's add new features and associated weights to $\widehat{V}$.

Two variations can be added to this general algorithm: setting a preference on existing features ϕ_i, and the possibility of removing them.

Preference on existing features

As explained previously, the norm of a feature on the training set influences the equi-gradient descent: one going in a slightly less good direction, but having a higher norm than another might be selected first. This is a good thing in the above algorithm, as an existing feature centered on a state far from the trajectory of the episode should not be used: it would take a high weight and dangerously modify $\widehat{V}$ outside the trajectory.

However, it would be benefitable to put a preference on existing *nearby* features, as long as they provide a good correction. This can be achieved by over-estimating them:

The candidate features proposed to the equi-gradient descent will be

- new features $\{\phi'_j\}$ centered on the visited states.
- existing features scaled so their norm on the data space are greater: $\{\phi'_i = \tau\phi_i\}$, with $\tau > 1$.

Thus, on existing features, the weights considered and penalized by the equi-gradient descent (egd) are lower than the ones eventually used:

$$\beta_i\phi_i = (\frac{\beta_i}{\tau})(\tau\phi_i)$$

and the equi-gradient descent will tend to use these features easily in the first steps. The weights β'_i computed by the egd on scaled existing features $\phi'_i = \tau\phi_i$ should then be scaled "back" by

$$\beta_i = \tau\beta'_i$$

De-activation of existing features

When the weight on an existing feature is zeroed, it is clearly of no use anymore. It should then be removed, as well as new features can be activated and then removed in the course of a single episode correction. This can be achieved in a simple way: equi-gradient algorithm, though stated with initial weights at zero, can be generalized to arbitrary ones. In this case, the vector $\boldsymbol{\beta}$ of active weights is built with these initial values: when activating an existing feature, instead of adding a 0 element to it the initial value is used. Then the α_0 test considers the overall value of the weight.

5. Experimental results

Experiments were run on the inverted pendulum problem, as described in (Coulom, 2002). 100 independent learning sessions of 300 episodes were run each time, the episodes consisting in 40 transitions of $0.1s$. The evolution of the quality of the value function was estimated by running 100 off-learning episodes and adding received rewards. The same initial states were used for each experiment. The results are presented in Fig.'s 6-10, and are detailed below.

Influence of the preference parameter τ

We first used a Gaussian kernel of variance 0.15 for each state, and compared choices for the preference parameter τ (scale factor on existing features to promote them to the egd). It has the expected influence on the number of actives features: as the τ increases, the number of active features naturally decreases, which is helpful to tune the computation time: in fact, the algorithm is linear in the number of active features.

The convergence speed and quality are overall similar, but it should be noted that the quality is slightly better with preference $\tau = 1.3$, see Fig. 6.

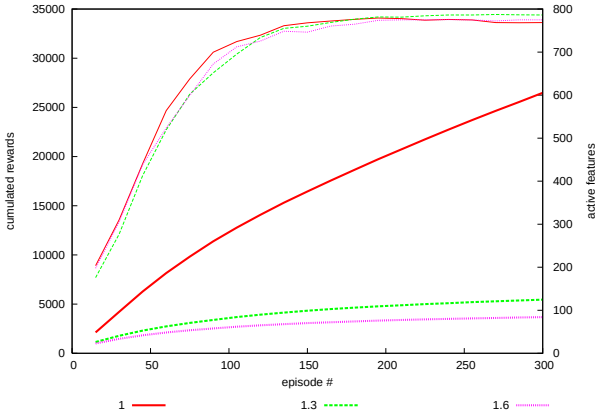

Figure 6. Influence of τ on the number of active features (3 thick lines), and on the cumulated reward (3 thin lines), for three values of τ. We use a single Gaussian kernel of variance 0.15.

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. 7.

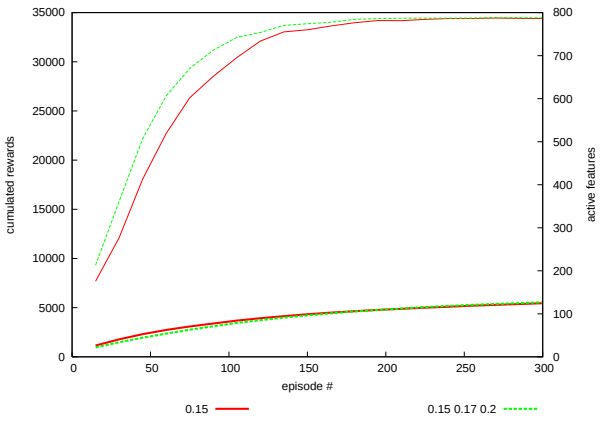

Figure 7. This plot compares the performance of the algorithm using a single kernel function (in red), and using 3 kernel functions (in green). All the kernels are Gaussian; they differ from their variance.

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 GPTD, 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. 8.

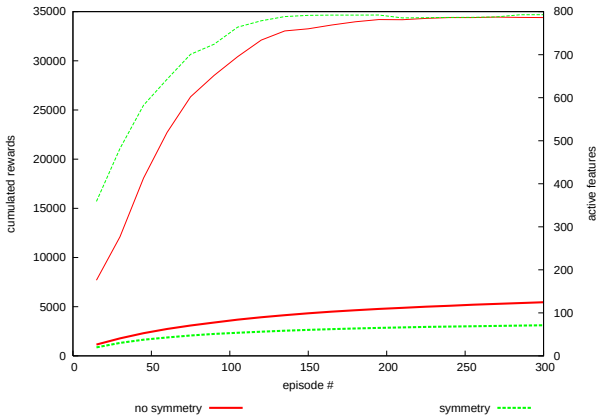

Figure 8. This plots exhibits the difference that is observed when one uses the symmetry of the problem (in green), or not (in red). The algorithm uses a single Gaussian kernel function of variance 0.15.

COMPARISON WITH OTHER METHODS

We compared the results obtained by the following algorithms:

- continuous TD(λ) with gradient descent on a parametric BFN: 15×15 grid of Gaussian bases.
- Gaussian Process TD, with a Gaussian kernel of variance 0.15 and a limit of 200 bases.
- Equi-gradient TD(λ) with the same single kernel and use of symmetry.

GPTD shows a fast convergence but the asymptotic quality is very variable. TD(λ) is robust, but the convergence is slow and the asymptotic quality is not optimal. Equi-gradient TD(λ) shows both fast convergence and robustness, and reaches better qualities. See Fig. 9

COMPLEXITY

Considering the complexity of an episode-update:

- TD(λ) is linear in the number of features.
- GPTD is quadratic in the number of features.

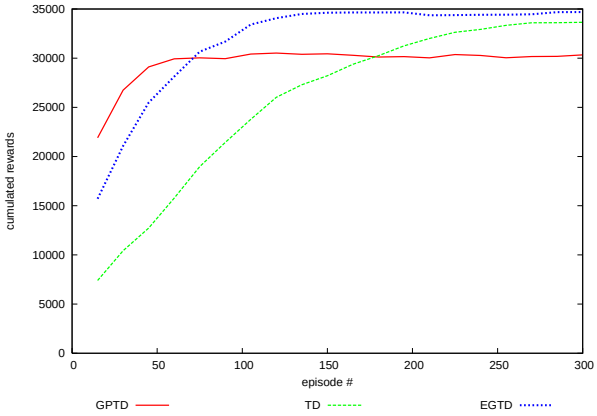

Figure 9. We plot the cumulated rewards of three different algorithms along the episodes. We can see the very good performance of the EGTD.

- Equi-gradient TD(λ) is linear in the number of candidates and quadratic in the number of selected features (which are a few).

When comparing convergences wrt. computation time, the preference goes even more to Equi-gradient TD(λ), as it starts with no features and keeps the number small. See Fig. 10.

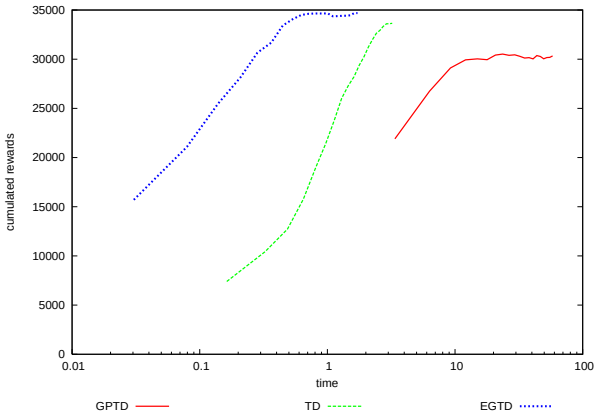

Figure 10. We plot the cumulated rewards against the amount of computational times for the three algorithms compared in Fig. 9 (notice the time log-scale). We see that the EGTD obtains its good performance while using the less computation time.

CAR ON THE HILL

Experiments were also quickly run on the *car on the hill* problem, but to this day only a few tests were made, with no formal estimation of the quality. It seems to show that equi-gradient TD converges faster again, and is less sensible than TD to the choice of λ.

6. Summary and future work

We formulated a variant of TD(λ) in which the gradient descent is replaced by what we called an equi-gradient descent. The first takes an approximatively good 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(λ) 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. Further experiments will be made on problems of higher dimension: acrobot, swimmer (Coulom, 2002), octopuss arm (Engel, 2005), etc. Future work will include

- going further in non-parametricity, by automating the construction of the set of candidates and the stopping criterion for the equi-gradient descent.
- exploring ways of altering existing features by a gradient backpropagation, and possibly mix with multi-layer perceptron.
- studying other schemes than episode-updates.

References

Bertsekas, D. (1996). *Neuro-dynamic programming.* Athena Scientific.

Coulom, R. (2002). *Reinforcement learning using neural networks, with applications to motor control.* Doctoral dissertation, Institut National Polytechnique de Grenoble.

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least-angle regression. *Annals of statistics, 32*, 407–499.

Engel, Y. (2005). *Algorithms and representations for reinforcement learning.* Doctoral dissertation, Hebrew University.

Engel, Y., Mannor, S., & Meir, R. (2005). Gaussian process with reinforcement learning. *Proc. of the International Conf. on Machine Learning (ICML).*

Guigue, V. (2005). *Méthodes à noyaux pour la représentation et la discrimination de signaux non-stationnaires.* Doctoral dissertation, Institut National des Sciences Appliquées de Rouen.

Lagoudakis, M., & Parr, R. (2003). Reinforcement learning as classification: Leveraging large margin classifiers. *Proc. of the International Conf. on Machine Learning (ICML).*

Ormoneit, D., & Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning, 49*, 161–178.

Ratitch B., P. D. (2004). Sparse distributed memories for on-line value-based reinforcement learning. *Proc. of the European Conf. on Machine Learning (ECML).*

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning, 3*, 9–44.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: an introduction.* MIT Press.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Comm. of the ACM, 38*, 58–68.