ESANN'2007 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2007, d-side publi., ISBN 2-930307-07-2.

A Unified View of TD Algorithms

Introducing *Full-gradient TD* and *Equi-gradient descent TD*

Manuel Loth[1] * and Philippe Preux[1] and Manuel Davy[2]

1- INRIA-Futurs - SequeL & Université de Lille / CNRS LIFL
Villeneuve d'Ascq - France

2- INRIA-Futurs - SequeL & École Centrale de Lille / CNRS LAGIS

Abstract. This paper addresses policy evaluation in MDP. It provides a unified view of algorithms such as *TD(λ)*, *LSTD(λ)*, *iLSTD*, and *residual-gradient TD*. We assert that they all consist of minimizing a gradient function and differ in the form of this function and their means of minimizing it. Building on this unified view, two new schemes are introduced: *Full-gradient TD* which uses a generalization of the principle introduced in *iLSTD*, and *EGD TD* which reduces the gradient by successive *equi-gradient descents*. These three algorithms share the worthy property of using much more efficiently the samples than *TD*, while keeping the good properties of gradient descent schemes.

1 The policy evaluation problem

A *Markov Decision Process* (MDP) [1] describes a dynamical system in which an agent has to learn a behavior so as to reach a given goal, in an optimal way. In this paper, the state of the system $s \in \mathcal{S}$ may be either discrete, or continuous. The agent applies an action $u \in \mathcal{U}$ at each time step $t \in \mathcal{N}$. This drives the system to a state $s' = u(s)$ at the next time step, where u is generally non-deterministic. A reward $r \in \mathcal{R} \subset \mathbb{R}$ is associated to each transition. A policy π defines the behavior of the agent on the system: it is a mapping from $\mathcal{S}$ to $\mathcal{U}$. One objective is to find a policy that maximizes the rewards. To this end, we need to evaluate the performance of any policy, and define its *value function* v^π:

$$v^\pi(s_0) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r(s_t \xrightarrow{\pi(s_t)} s_{t+1})\right)$$

where $0 < \gamma \leq 1$ is a discount factor[1].

Given a trajectory $s_0 \xrightarrow{r_0} s_1 \xrightarrow{r_1} \dots \xrightarrow{r_{T-1}} s_T$ (where $r_t = r(s_t \xrightarrow{\pi(s_t)} s_{t+1})$), all the information about v lies in the following set of Bellman equations[2]:

$$\begin{cases} v(s_0) & = & r_0 + \gamma v(s_1) \\ \dots & & \\ v(s_{T-1}) & = & r_{T-1} + \gamma v(s_T) \end{cases}$$

*ML gratefully acknowledges the support from *Region Nord - Pas-de-Calais* and *INRIA* (PhD grant).

[1] We will now use the notation v for v^π.

[2] The equalities are abusive when the actions are not deterministic, but averaging these equations converges to valid equations as the number of samples tends to infinity.

The policy evaluation problem thus consists of finding a function $\hat{v}$ that satisfies Bellman's equations as good as possible, using one or several trajectories. In several previous works, this has been achieved by searching $\hat{v}$ as a linear combination of basis functions (*features*): $\hat{v}(s) = \sum_{i=1}^{n} \omega_i \phi_i(s)$. Here, we propose to cast several important previous methods in a unified and simple framework, as follows:

- define a n-vector-valued gradient function $\boldsymbol{\mu}$ which depends on the observed transitions and on $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n)^\mathsf{T}$. This function evaluates the contribution of each ω_i to some error measure (see details below).
- iterate the following two steps:
  - update $\boldsymbol{\mu}$ using new transitions
  - modify $\boldsymbol{\omega}$ in order to reduce $\boldsymbol{\mu}$, and re-evaluate $\boldsymbol{\mu}$.

This paper is organized as follows: Section 2 discusses the two currently used gradient functions and their meaning. Section 3 presents the TD algorithms – *TD(λ)* [1] and *residual-gradient TD* [2] – in that framework. Section 4 shows that *LSTD(λ)* [3] and *LSPE(λ)* [4] and their *Bellman-residual* versions share the same kind of derivation. Section 5 discusses a third family of algorithms that use an intermediate update scheme (*full gradient*). It includes *iLSTD* [5, 6] and two algorithms introduced in this paper: *Full-TD* and *Equi-gradient descent TD*. Section 6 presents experiments made on the *Boyan chain* MDP, which illustrate some of the benefits and drawbacks of each method. Complete proofs of the equivalences of these formulations with the original ones and derivation of the equi-gradient descent algorithm are exposed in [7, 8].

2 Fixed-point gradient vs. Bellman-residual gradient

The *TD(0)* algorithm estimates v^π iteratively by using its current estimate $\hat{v}$ to approximate the right hand side of the Bellman equations:

$$\begin{aligned} v(s_t) = r_t + \gamma v(s_{t+1}) \quad &\Rightarrow \quad v(s_t) \simeq r_t + \gamma \hat{v}(s_{t+1}) \\ &\Rightarrow \quad v(s_t) - \hat{v}(s_t) \simeq r_t - \hat{v}(s_t) + \gamma \hat{v}(s_{t+1}) \end{aligned}$$

and consequently updating $\hat{v}(s_t) \leftarrow \hat{v}(s_t) + \alpha \left(r_t - \hat{v}(s_t) + \gamma \hat{v}(s_{t+1})\right)$

TD(λ) averages such approximations of $v(s_t)$ on all "dynamic programming ranks". It can be seen as expanding the system to all implicit equations:

$$\begin{cases} v(s_0) = r_0 + \gamma v(s_1) = r_0 + \gamma(r_1 + \gamma v(s_2)) = \ldots = r_0 + \gamma(r_1 + \gamma(r_2 + \ldots + \gamma v(s_T))) \\ v(s_1) = r_1 + \gamma v(s_2) = \ldots \\ \ldots \end{cases}$$

and again replacing v by $\hat{v}$ in the right hand sides. The different estimations of $v(s_t)$ are averaged using coefficients determined by a value $\lambda \in [0, 1]$, which leads to estimating $v(s_t) - \hat{v}(s_t)$ by $\sum_{\tau=t}^{n-1} (\lambda\gamma)^{\tau-t}(r_\tau - \hat{v}(s_\tau) + \gamma \hat{v}(s_{\tau+1}))$. This error signal is again used to update $\hat{v}(s_t)$. As far as we consider $\hat{v}$ to be a linear combination of basis functions, the vector of error signals on $\hat{v}(s_0), \ldots, \hat{v}(s_{T-1})$

can be written as $\mathbf{L}(\mathbf{r} - \mathbf{B}\mathbf{\Phi}\boldsymbol{\omega}) =$

$$\begin{pmatrix} 1 & \lambda\gamma & (\lambda\gamma)^2 & \dots \\ & 1 & \lambda\gamma & \dots \\ \mathbf{0} & & \ddots & \end{pmatrix} \left[\begin{pmatrix} r_0 \\ \vdots \\ r_{T-1} \end{pmatrix} - \begin{pmatrix} 1 & -\gamma & & \mathbf{0} \\ & 1 & -\gamma & \\ \mathbf{0} & & \ddots & \end{pmatrix} \begin{pmatrix} \phi_1(s_0) & \dots & \phi_m(s_0) \\ \vdots & & \vdots \\ \phi_1(s_T) & \dots & \phi_m(s_T) \end{pmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_m \end{pmatrix} \right]$$

Let us note $\mathbf{v} = (v(s_0), \dots, v(s_{T-1}))^\mathsf{T}$ and $\hat{\mathbf{v}} = (\hat{v}(s_0), \dots, \hat{v}(s_{T-1}))^\mathsf{T}$. Considering $\mathbf{L}(\mathbf{r} - \mathbf{B}\mathbf{\Phi}\boldsymbol{\omega})$ as a good estimate of $\mathbf{v} - \hat{\mathbf{v}} = \mathbf{v} - \mathbf{\Phi}\boldsymbol{\omega}$, one can minimize $\frac{1}{2}\|\mathbf{v} - \hat{\mathbf{v}}\|^2$ by minimizing the gradient $\nabla_{\boldsymbol{\omega}}(\mathbf{v} - \hat{\mathbf{v}})\ (\mathbf{v} - \hat{\mathbf{v}})$ estimated by $-\mathbf{\Phi}^\mathsf{T}\mathbf{L}(\mathbf{r} - \mathbf{B}\mathbf{\Phi}\boldsymbol{\omega})$ (only $\mathbf{v} - \hat{\mathbf{v}}$ is replaced by its approximation, not its gradient). This gives what one may call a "fixed-point gradient".

Another way of doing is to aim at solving the Bellman system, *ie.* minimize $\frac{1}{2}\|\mathbf{r} - \mathbf{B}\mathbf{\Phi}\boldsymbol{\omega}\|^2$ w.r.t. $\boldsymbol{\omega}$. This gives the Bellman-residual gradient $\nabla_{\boldsymbol{\omega}}(\mathbf{r} - \mathbf{B}\mathbf{\Phi}\boldsymbol{\omega})\ (\mathbf{r} - \mathbf{B}\mathbf{\Phi}\boldsymbol{\omega}) = -\mathbf{\Phi}^\mathsf{T}\mathbf{B}^\mathsf{T}(\mathbf{r} - \mathbf{B}\mathbf{\Phi}\boldsymbol{\omega})$.

The conceptual difference is simple: the fixed-point gradient transforms the errors on transitions (temporal differences) into estimated errors on $\hat{v}$ itself (*ie.* errors on single states) by a multi-rank dynamic programming scheme, and then projects these estimated errors on the parameter $\boldsymbol{\omega}$, whereas the Bellman-residual gradient performs a direct projection.

We will consider these gradients ($\boldsymbol{\mu}$) on all observed transitions (including several trajectories), by adequately expanding all vectors and matrices. The iterative computation of these gradients proceeds according to the following way: the components of the vector $\mathbf{r} - \mathbf{B}\mathbf{\Phi}\boldsymbol{\omega}$ are the successive temporal differences $d_t = r_t - \hat{v}(s_t) + \gamma\hat{v}(s_{t+1})$; the columns of $\mathbf{\Phi}^\mathsf{T}\mathbf{L}$ (resp. $\mathbf{\Phi}^\mathsf{T}\mathbf{B}^\mathsf{T}$) are referred to as the *eligibility traces* $\mathbf{z_t}$. Each new sampled transition modifies the gradient $\boldsymbol{\mu}$ by $\boldsymbol{\mu_t} \leftarrow \boldsymbol{\mu_{t-1}} + d_t\mathbf{z_t}$, $\mathbf{z_t}$ itself being computed iteratively.

These gradients, as well as $\hat{v}$, are linear in $\boldsymbol{\omega}$: $\boldsymbol{\mu} = \mathbf{A}\boldsymbol{\omega} - \mathbf{b}$, with $\mathbf{b} = \mathbf{\Phi}^\mathsf{T}\mathbf{L}\mathbf{r}$, and $\mathbf{A} = \mathbf{\Phi}^\mathsf{T}\mathbf{L}\mathbf{B}\mathbf{\Phi}$ (resp. $\mathbf{\Phi}^\mathsf{T}\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{\Phi}$).

In the following where ways of modifying $\boldsymbol{\omega}$ given the value of $\boldsymbol{\mu}$ are discussed, let us note $\boldsymbol{\delta}_{\boldsymbol{\omega}}$ the additive term of any modification: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \boldsymbol{\delta}_{\boldsymbol{\omega}}$.

3 TD algorithms

In its purely iterative form, *TD(λ)* [1] performs the following update after each transition: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha d_t\mathbf{z_t}$. Equivalently, the updates can be performed only after each trajectory, which is more consistent with its definition. Depending on one's view (related to the backward/forward views discussed in [1]), the first scheme can be considered as the natural one and the second as accumulating successive updates before committting it at the end, or the second one can be seen as more natural (given the explanation in the previous section) and the first one as a partial update given the partial computation of $\boldsymbol{\mu}$. Note that here, $\boldsymbol{\mu}$ only concerns the current trajectory: the updates performed in *TD(λ)* only take into account the last trajectory.

Let us take a neutral point of view and state that the algorithm considers the gradient on the current trajectory and updates weights at any chosen time

(but necessarily including the end of the trajectory) by $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha\boldsymbol{\mu}$ followed by $\boldsymbol{\mu} \leftarrow \mathbf{0}$: $\boldsymbol{\mu}$ is computed iteratively, and each time a partial computation has been used, it is "thrown away". At the end of each trajectory, the associated gradient has been used for one update $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha\boldsymbol{\mu}$ and is then forgotten.

The *residual-gradient TD* algorithm [2] is actually the same algorithm, only using the Bellman-residual gradient.

4 LSTD algorithms

When $\hat{v}$ is approximated by alinear combination of features, it has been shown in [9] that $\boldsymbol{\omega}$ converges in *TD(λ)* to $\boldsymbol{\omega}^*$ such that $\boldsymbol{\mu}(\boldsymbol{\omega}^*) = \mathbf{A}\boldsymbol{\omega}^* - \mathbf{b} = \mathbf{0}$. This leads to the *LSTD(λ)* algorithm [3] which, given sampled trajectories, directly computes $\boldsymbol{\omega}^* = \mathbf{A}^{-1}\mathbf{b}$.

For various motivations like numerical stability, or the use of optimistic policy iteration, or the possible singularity of $\mathbf{A}$, or a smooth processing time, or getting a specific point of view on the algorithm, the computation can be performed iteratively. The algorithm can then be described as follows:

- for each new transition, update $\boldsymbol{\mu}$ as exposed in section 2, and update $\mathbf{A}^{-1}$ (using Shermann-Morrison formula),
- whenever wanted, reduce $\boldsymbol{\mu}$ by updating $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \mathbf{A}^{-1}\boldsymbol{\mu}$. $\boldsymbol{\omega}$ is then the exact solution of $\boldsymbol{\mu}(\textit{samples so far}, \boldsymbol{\omega}) = \mathbf{0}$ and $\boldsymbol{\mu}$ is updated to $\mathbf{0}$.

Again, the same algorithm can be applied using the Bellman-residual gradient.

In [4] is introduced a similar algorithm, namely *Least Squares Policy Evaluation*. The difference resides in the update $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \left(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\mu}$, and the consequent update $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \mathbf{A}\boldsymbol{\delta}_{\boldsymbol{\omega}}$.

5 Full-gradient algorithms

Three algorithms are presented in this section that all rely on the same idea: reduce $\boldsymbol{\mu}$ (again at any time) in a gradient descent way, but maintain its "real" value: instead of zeroing it after each update[3], the residual of the gradient is kept. Then, the next updates not only perform one gradient descent step using the current trajectory, but also continue this process for the previous ones.

The first natural algorithm is introduced here as *Full-gradient TD* and consists in replacing $\boldsymbol{\mu} \leftarrow \mathbf{0}$ by $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \mathbf{A}\boldsymbol{\delta}\boldsymbol{\omega}$ in the *TD* algorithm.

The *iLSTD* algorithm was introduced in [5, 6] (as well as the notation $\boldsymbol{\mu}$). Although it is presented as a variation of *LSTD* (hence its name), it is more closely related to gradient descent than to the exact least-squares solving scheme. With the "any-time update" generalization used throughout this article, it can be described as a full-gradient TD in which $\boldsymbol{\omega}$ is updated only on its most correlated component: $\omega_i \leftarrow \omega_i + \alpha\mu_i, \quad \text{with } i = \arg\max |\mu_i|$.

[3] which corresponds to forget each trajectory after only one gradient descent step on its contribution to the overall gradient $\boldsymbol{\mu}$

Finally, the equi-gradient descent (EGD) TD, introduced here, consists in taking EGD [8] steps as an update scheme. In a few words, EGD also consists in modifying only the most correlated parameter ω_i, but α is chosen so that after this update, another parameter ω_j becomes equi-correlated. The next update is $\begin{pmatrix} \omega_i \\ \omega_j \end{pmatrix} \leftarrow \begin{pmatrix} \omega_i \\ \omega_j \end{pmatrix} + \alpha_2 \begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix}^{-1} \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$, and so on. The constraint is that to allow the exact computations of the step lengths, $\boldsymbol{\mu}$ must not be modified (by new samples) in between those steps. So a typical update schedule is to perform a certain number of steps at the end of each trajectory, preferably to one or a few steps after each transition.

The benefit exposed in the first paragraph comes at the cost of maintaining the matrix $\mathbf{A}$, which has the same order of complexity as maintaining $\mathbf{A}^{-1}$ in *LSTD*, but is still about half less complex. However, as exposed in [5], if the features are sparse, the complexity of the two last algorithms can be lowered, unlike in *LSTD*.

EGD TD presents the crucial benefit of not having to tune the α update parameter of gradient descent schemes. Instead of setting the lengths of descent steps beforehand and uniformly, and cross-validate them, they are computed on the fly given the data.

6 Experiments

Experiments were run on a 100 states Boyan chain MDP [3]. Details are exposed in [7]. The fixed-point gradient was used, with $\lambda = 0.5$.

- In fig. 1, the RMSE is plotted against the number of trajectories to illustrate the differences between full exploitation of the samples (least-squares and full-gradient methods) and *TD*,
- In fig. 2, the RMSE is plotted against the computational time: the three families are clearly clustered. Note that the sparsity of the features has not been taken into account, and *EGD TD* and *iLSTD* can perform much better on that point, as experimented in [5] for the latter.

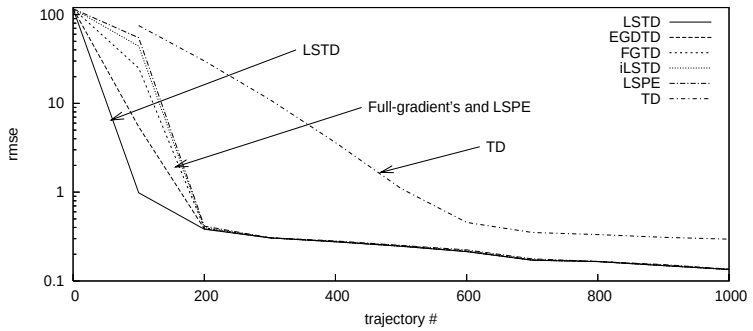

Fig. 1: Root mean squared error against the number of trajectories

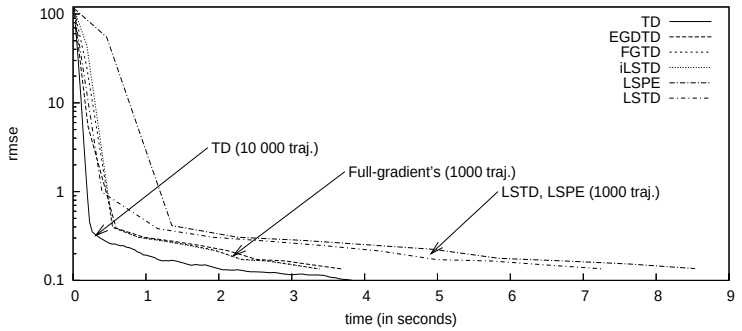

Fig. 2: Root mean squared error against the computational time

7 Summary and perspectives

Classical algorithms of reinforcement learning have been presented here in a view both practical and enlightening. This view allows a natural introduction of a new intermediate family of algorithms that performs stochastic reduction of the errors, as in TD, yet makes full use of the samples, as in LSTD. Let alone the time and sample complexities, these methods open interesting perspectives in the frame of optimistic policy iteration. Indeed, the principle of neither forgetting samples after a small update, nor directly fully taking them into account, may allow to make a better use of samples than TD while avoiding the issue met by LSTD: making too much case of samples coming from previous policies. This can be achieved by scaling $\boldsymbol{\mu}$ by a discount factor after each trajectory (for example), which amounts to reducing only a given ratio of it.

References

[1] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. MIT Press, 1998.

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

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

[4] A. Nediç and D. P. Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110, 2003.

[5] A. Geramifard, M. Bowling, and R. Sutton. Incremental least-squares temporal difference learning. In *Proceeding of AAAI*, pages 356–361, 2006.

[6] A. Geramifard, M. Bowling, M. Zinkevich, and R. Sutton. iLSTD: Eligibility traces & convergence analysis. In *Proceeding of NIPS*, 2006 to appear.

[7] M. Loth. A unified view of td algorithms – introducing full-gradient td and equi-gradient descent td. Technical report, INRIA-Futurs, 2006, to appear.

[8] M. Loth. Equi-gradient descent. Technical report, INRIA-Futurs, 2006, to appear.

[9] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Trans. on Automatic Control*, 42(5):674–690, May 1997.