A Unified View of TD Algorithms; Introducing Full-Gradient TD and Equi-Gradient Descent TD

Manuel Loth, Philippe Preux

A Unified View of TD Algorithms

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

Manuel Loth * and Philippe Preux

INRIA-Futurs - SequeL
Université de Lille - LIFL
Villeneuve d'Ascq - France

Abstract. This paper addresses the issue of policy evaluation in Markov Decision Processes, using linear function approximation. It provides a unified view of algorithms such as *TD(λ)*, *LSTD(λ)*, *iLSTD*, *residual-gradient TD*. It is asserted that they all consist in minimizing a gradient function and differ by the form of this function and their means of minimizing it. Two new schemes are introduced in that framework: *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 form a new intermediate family with the interesting property of making much better use of the samples than *TD* while keeping a gradient descent scheme, which is useful for complexity issues and optimistic policy iteration.

1 The policy evaluation problem

A *Markov Decision Process* (MDP) describes a dynamical system and an agent. The system is described by its state $s \in \mathcal{S}$. When considering discrete time, the agent can apply at each time step an action $u \in \mathcal{U}$ which drives the system to a state $s' = u(s)$ at the next time step. u is generally non-deterministic.

To each transition is associated a reward $r \in \mathcal{R} \subset \mathbb{R}$. A policy π is a function that associates to any state of the system an action taken by the agent.

Given a discount factor γ, the value function v^π of a policy π associates to any state the expected discounted sum of rewards received when applying π from that state for an infinite time:

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

This paper addresses the evaluation of a policy by approximating the value function as a linear combination of fixed features, and estimating the coefficients from sampled trajectories (sequences of visited states and received rewards when starting from a certain state).

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

All the information on v contained in a trajectory $s_0 \xrightarrow{r_0} s_1 \xrightarrow{r_1} \dots \xrightarrow{r_{n-1}} s_n$ lies in the following system of Bellman equations:

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

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 consists in finding a function that satisfies the most this system (which may include several trajectories). This can be achieved in several ways. In the following, all major methods are described in a single and simple framework:

- define a gradient function μ of the observed transitions and parameters;
- update its value whenever a new transition is observed;
- whenever needed, modify the parameters in order to reduce μ, and then update its value.

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 experimentations made on the *Boyan chain* MDP, which illustrate some of the benefits and drawbacks of each method. Finally, the conclusion discusses the potential advantages of the full gradient scheme for optimistic policy iteration.

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 iteratively by using its current estimate $\hat{v}$ to approximate the right hand side of these equations:

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

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:

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

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 $\sum_{\tau=t}^{T-1} (\lambda\gamma)^{\tau-t}(r_\tau - \hat{v}(s_\tau) + \gamma\hat{v}(s_{\tau+1}))$. This error signal is again used to update $\hat{v}(s_t)$. In the case of linear approximators, the vector of error signals on $\hat{v}(s_0), \ldots, \hat{v}(s_{T-1})$ can be written as $\mathbf{L}(\mathbf{r} - \mathbf{B}\boldsymbol{\Phi}\boldsymbol{\omega}) =$

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

They are projected on the parameter $\boldsymbol{\omega}$ of $\hat{v}$ by $\boldsymbol{\Phi}^\mathsf{T}\mathbf{L}(\mathbf{r} - \mathbf{B}\boldsymbol{\Phi}\boldsymbol{\omega})$. This gives what one can call a fixed-point gradient, which is the sum of these on all trajectories (*ie.* the same expression with adequately extended vectors and matrices).

Another way of doing is to aim at solving the Bellman system, *ie.* minimize $\|\mathbf{r} - \mathbf{B}\boldsymbol{\Phi}\boldsymbol{\omega}\|_2^2$ w.r.t. $\boldsymbol{\omega}$. This gives the Bellman-residual gradient $\boldsymbol{\Phi}^\mathsf{T}\mathbf{B}^\mathsf{T}(\mathbf{r} - \mathbf{B}\boldsymbol{\Phi}\boldsymbol{\omega})$.

The conceptual difference is simple: The fixed-point gradient transforms the errors on transitions (temporal differences) on the approximate value function 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 does a direct projection.

The iterative computation of these gradients proceeds according to the following way: the components of the vector $\mathbf{r} - \mathbf{B}\boldsymbol{\Phi}\boldsymbol{\omega}$ are the successive temporal differences $d_t = r_t - \hat{v}(s_t) + \gamma\hat{v}(s_{t+1})$; the columns of $\boldsymbol{\Phi}^\mathsf{T}\mathbf{L}$ or $\boldsymbol{\Phi}^\mathsf{T}\mathbf{B}^\mathsf{T}$ are referred to as the *eligibility traces* $\mathbf{z_t}$ in the first case – this denomination will be extended here to the second case. 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 $\mathbf{b} = \Phi^\mathsf{T}\mathbf{L}\mathbf{r}$, and $\mathbf{A} = \boldsymbol{\Phi}^\mathsf{T}\mathbf{L}\mathbf{B}\boldsymbol{\Phi}$ in the fixed-point case, or $\mathbf{A} = \boldsymbol{\Phi}^\mathsf{T}\mathbf{B}^\mathsf{T}\mathbf{B}\boldsymbol{\Phi}$ in the Bellman-residual case.

In the following, let us note $\boldsymbol{\delta_\omega}$ the additive term of any update of $\boldsymbol{\omega}$ in the algorithms.

3 TD algorithms

TD(λ) [1], in its purely iterative form, 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 cumulating successive updates before commiting 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 update weights at any chosen time (but necesseraly including the end of the trajectory) by $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha\boldsymbol{\mu}$ followed by $\mu \leftarrow \mathbf{0}$: $\boldsymbol{\mu}$ 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.

To summarize, given the fixed-point gradient function $\mu(\textit{observed transitions}, \boldsymbol{\omega})$, *TD(λ)* updates $\boldsymbol{\mu}$ after each transition (as exposed in previous section), and – whenever wanted– performs a parameter update $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}+\alpha\boldsymbol{\mu}$ followed by $\boldsymbol{\mu} \leftarrow \mathbf{0}$.

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

4 LSTD algorithms

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 lead 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, use of optimistic policy iteration, the possible singularity of $\mathbf{A}$, 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-Morrisson 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.

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

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, which corresponds to forgetting each trajectory after only one gradient descent step on its contribution to the overall gradient $\boldsymbol{\mu}$, the residual of the gradient is kept, and thus the following updates not only perform one gradient descent step on 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 most 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 more correlated component: $\omega_i \leftarrow \omega_i + \alpha\mu_i, \quad$ with $i = \arg\max |\mu_i|$.

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 such 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 (states have a non-zero value only on a subset of the features), 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 *a priori* and uniformely, 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 $\lambda = 0.5$. Here are plotted

- in 1, the RMSE against the number of trajectories, which illustrates the differences between full exploitation of the samples (least-squares and full-gradient methods) and *TD*,
- in 2, the RMSE against the computational time, where 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.

7 Summary and perspectives

The classical algorithms of reinforcement learning have been presented here in a view both practical and enlightning. This view allows a natural introduction of a new intermediate family of algorithms that performs stochastic reduction of the errors, as in TD, but make full use of the samples, as in LSTD. Let alone the time or sample complexity, 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 take them into account, may allow to make a better use of samples than TD while avoiding the issue met by LSTD

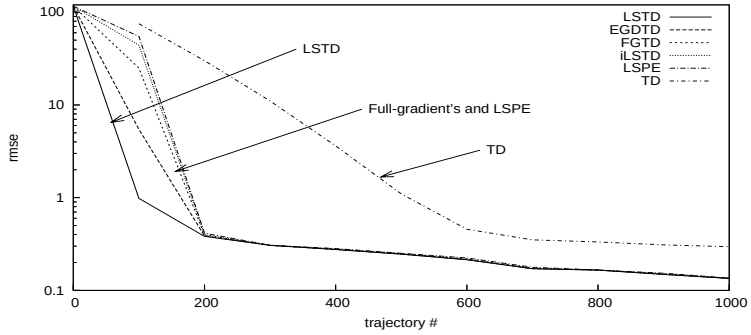

Fig. 1: Root mean squared error against the number of trajectories

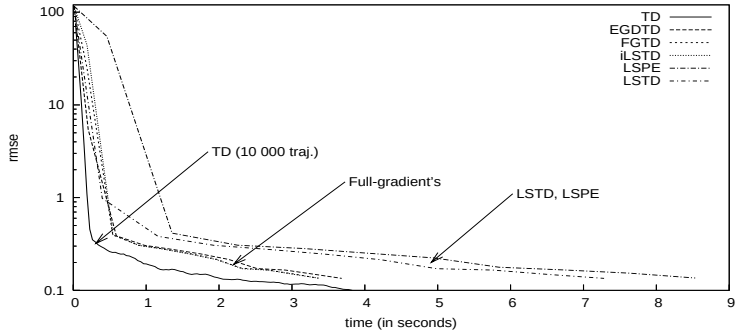

Fig. 2: Root mean squared error against the computational time

in that frame: 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 American Association for Artificial Intelligence (AAAI)*, pages 356–361, 2006.

[6] A. Geramifard, M. Bowling, M. Zinkevich, and R. Sutton. iLSTD: Eligibility traces & convergence analysis. In *Proceeding of Neural Information Processing Systems Conference*, 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. Technical Report LIDS-P-2322, 1996.