Momentum in Reinforcement Learning

Nino Vieillard[1,2] **Bruno Scherrer**[2] **Olivier Pietquin**[1] **Matthieu Geist**[1]
[1]Google Research, Brain Team
[2]Université de Lorraine, CNRS, Inria, IECL, F-54000 Nancy, France

Abstract

We adapt the optimization's concept of momentum to reinforcement learning. Seeing the state-action value functions as an analog to the gradients in optimization, we interpret momentum as an average of consecutive q-functions. We derive Momentum Value Iteration (MoVI), a variation of Value iteration that incorporates this momentum idea. Our analysis shows that this allows MoVI to average errors over successive iterations. We show that the proposed approach can be readily extended to deep learning. Specifically,we propose a simple improvement on DQN based on MoVI, and experiment it on Atari games.

1 Introduction

Reinforcement Learning (RL) is largely based on Approximate Dynamic Programming (ADP), that provides algorithms to solve Markov Decision Processes (MDP, Puterman [1994]) under approximation. In the exact case, where there is no approximation, classic algorithms such as Value Iteration (VI) or Policy Iteration (PI) are guaranteed to converge to the optimal solution, that is find an optimal policy that dominates every policy in terms of value. These algorithms rely on solving fixed-point problems: in VI, one tries to reach the fixed point of the Bellman optimality operator by an iterative method. We focus on VI for the rest of the paper, but the principle we propose can be extended beyond this. Approximate Value Iteration (AVI) is a VI scheme with approximation errors. It is well known [Bertsekas and Tsitsiklis, 1996] that if the errors do not vanish, AVI does not converge. To get some intuition, consider a sequence of policies being greedy according to the optimal q-function, with an additional state-action dependant noise. The resulting sequence of policies will be unstable and suboptimal, even with centered and bounded noise. Dealing with errors is however crucial to RL, as we hope to tackle problems with large states spaces that require function approximation. Indeed, many recent RL successes are algorithms that instantiate ADP schemes with neural networks for function approximation. Deep Q-Networks (DQN, Mnih et al. [2015]) for example, can be seen as an extension of AVI with neural networks.

In optimization, a common strategy to stabilize the descent direction, known as momentum, is to average the successive gradients instead of considering the last one. In reinforcement learning, the state-action value function can be seen informally as a kind of gradient, as it gives an improvement direction for the policy. Hence, we propose to bring the concept of momentum to reinforcement learning by basically averaging q-values in a DP scheme.

We introduce Momentum Value Iteration (MoVI) in Section 4. It is Value Iteration, up to the fact that the policy, instead of being greedy with respect to the last state-action value function, is greedy with respect to an average of the past value functions. We analyze the propagation of errors of this scheme. In AVI, the performance bound will depend on a weighted sum of the norms of the errors at each iteration. For MoVI, we show that this depends on the norms of the cumulative errors of previous iteration. This means that it allows for a compensation of errors along different iterations, and even convergence in the case of zero-mean and bounded noises, under some assumption. This compensation property is shared by a few algorithms that will be discussed in Section 6. We also show that MoVI can be successfully combined with powerful function approximation by proposing Momentum-DQN in Section 5, an extension of MoVI with neural networks based on DQN. It provides a strong performance improvement over DQN on the standard Arcade Learning Environment (ALE) benchmark [Bellemare et al., 2013]. All stated results are proven in the appendix.

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

2 Background

Markov Decision Processes. We consider the RL setting where an agent interacts with an environment modeled as an infinite discounted horizon MDP. An MDP is a quintuple $\{\mathcal{S}, \mathcal{A}, P, r, \gamma\}$, where $\mathcal{S}$ is a finite[1] state space, $\mathcal{A}$ a finite action space, $P \in \Delta_{\mathcal{S}}^{\mathcal{S}\times\mathcal{A}}$ is a Markovian transition kernel (writing Δ_X the simplex over the set X), $r \in [-r_{\max}, r_{\max}]^{\mathcal{S}\times\mathcal{A}}$ a reward function and $\gamma \in (0,1)$ the discount factor. A policy π maps the state space to distributions over actions $\pi(\cdot|s)$. We define the q-value q_π of a policy π as, for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$,

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s, a_0 = a\right],$$

where $\mathbb{E}_\pi$ denotes the expected value over all trajectories $(s_1, a_1, s_2, a_2, \dots)$ produced by π. The value is bounded by $q_{\max} = r_{\max}/(1-\gamma)$. Let us define the transition kernel operator associated to π as, for each $q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ and for each $(s,a) \in \mathcal{S}\times\mathcal{A}$, as $[P_\pi q](s,a) = \mathbb{E}_{s'\sim P(\cdot|s,a), a'\sim\pi(\cdot|s')}[q(s',a')]$. The q-function of a policy is the fixed point of its Bellman evaluation operator, defined for each $q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ as $T_\pi q = r + \gamma P_\pi q$. An optimal policy π_* is such that for any other policy π, we have that, for each $(s,a) \in \mathcal{S}\times\mathcal{A}$, $q_{\pi_*}(s,a) \geq q_\pi(s,a)$. The Bellman optimality operator is defined as $T_* q = \max_\pi T_\pi q$, and we have that q_* is the unique fixed point of T_*. A policy is said to be greedy with respect to $q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ if $T_* q = T_\pi q$. We denote the set of these policies $\mathcal{G}(q)$. Note that such a policy can be computed without accessing to the model (the transition kernel).

Finally, for $\mu \in \Delta_{\mathcal{S}\times\mathcal{A}}$ we write $d_{\pi,\mu} = (1-\gamma)\mu(I - \gamma P_\pi)^{-1}$ the discounted cumulative occupancy measure induced by π when starting from the distribution μ (distributions being written as row vectors). We define the μ-weighted ℓ_p-norm as, for each $q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, $\|q\|_{p,\mu} = \left(\mathbb{E}_{(s,a)\sim\mu}\left[|q(s,a)|^p\right]\right)^{\frac{1}{p}}$.

Approximate Value Iteration. Approximate Dynamic Programming provides algorithms to solve an MDP under some errors. One classic algorithm is Approximate Value Iteration. It looks directly for the fixed point of T_* with an iterative process

$$\begin{cases} \pi_{k+1} \in \mathcal{G}(q_k) \\ q_{k+1} = T_{\pi_{k+1}} q_k + \epsilon_{k+1}. \end{cases} \qquad \text{(AVI)}$$

Notice that here, $T_{\pi_{k+1}} q_k = T_* q_k$. In this scheme, we call the first line the greedy step, and the second line the partial evaluation step. AVI satisfies the following bound for the quality of the policy π_k

$$\|q_* - q_{\pi_k}\|_\infty \leq 2\gamma^k q_{\max} + \frac{2\gamma \max_{j<k} \|\epsilon_j\|_\infty}{(1-\gamma)^2}. \qquad (1)$$

This explains why AVI is not resistant to errors: $\max_{j<k} \|\epsilon_j\|_\infty$ can be high even if each ϵ_k is zero-mean.

[1]This is for ease and clarity of exposition, the proposed algorithm and analysis can be extended to continuous state spaces.

3 Momentum Value Iteration

In the context of optimization, momentum aims at stabilizing gradient ascent (or descent) methods. Consider we want to maximize a concave function f whose gradient is not known analytically, and we use a classic (stochastic) gradient ascent algorithm. This algorithm iterates from a value x_0 by computing an approximation g_k of $\nabla f(x_k)$, and updating $x_{k+1} = x_k + \eta g_k$. One can then use momentum [Qian, 1999] to stabilize the process through a smoothing function $h_k = \rho h_k + g_k$, with $\rho \in \mathbb{R}$, and an update $x_{k+1} = x_k + \eta h_k$. This can stabilize the ascent as the gradient may vary greatly from step to step.

In the context of ADP, the q-function intuitively gives the direction that guides the policy, in the same way that the gradient is the improvement direction of a variable. In particular, we can rewrite the greedy step (in AVI) as $\pi_k(\cdot|s) \in \operatorname{argmax}_{\pi(\cdot|s)\in\Delta_\mathcal{A}} \langle q_k(s,\cdot), \pi(\cdot|s)\rangle$, thus seeing this step as finding the policy being state-wise the most colinear with q_k. This is also reminiscent of the direction finding subproblem of Frank and Wolfe [1956]. Consequently, the greedy step can be seen as an analog of the update in gradient ascent (the policy π is analog to the variable x), the differences being *(i)* that q_k in AVI is not a gradient, but the result of an iterative process, $q_k = T_{\pi_k} q_{k-1}$, and *(ii)* that the policy is not updated, but replaced.

This analogy is thus quite limited (q_k is not really a gradient, there is no optimized function, the policy is replaced rather than updated). However, it is sufficient to adapt the momentum idea to AVI, by replacing the q-function in the improvement step by a smoothing of the q-functions, $h_k = \rho h_{k-1} + q_k$. We can then notice that $\mathcal{G}(h_k) = \mathcal{G}(\frac{h_k}{1+\rho})$, allowing us to compute a moving average instead of a smoothing, $h_k = \beta_k h_{k-1} + (1-\beta_k) q_k$, which leads to the following ADP scheme, initialized with $h_0 = q_0$,

$$\begin{cases} \pi_{k+1} = \mathcal{G}(h_k) \\ q_{k+1} = T_{\pi_{k+1}} q_k + \epsilon_{k+1} \\ h_{k+1} = \beta_{k+1} h_k + (1-\beta_{k+1}) q_{k+1}. \end{cases} \qquad \text{(MoVI)} \quad (2)$$

We call this scheme Momentum Value Iteration (MoVI), we analyze it in the following section.

4 Analysis

For the analysis, we consider a specific case of the scheme in Equation (2), with an empirical mean rather than an iteration-dependant moving average. This amounts to define $\beta_k = \frac{k}{k+1}$ in Eq. (2). We study the propagation of errors of MoVI, to see how it is impacted by the introduction of momentum, compared to a classic AVI scheme (see Eq. (1)).

4.1 Error propagation analysis

First, let us define some useful notations. We denote $P_{j:i} = P_{\pi_j} P_{\pi_{j-1}} \dots P_{\pi_i}$ if $1 \le i \le j$, $P_{j:i} = I$ otherwise, where π_j is the policy computed by MoVI at iteration j. We then define the negative cumulative error $E_k = -\sum_{j=1}^{k} \epsilon_j$, and the weighted negative cumulative error $E'_{k,j} = -\sum_{i=1}^{k-j} P_{i+j:i+1}(I - \gamma P_{\pi_i})\epsilon_i$.

To study the efficiency of the algorithm, the natural quantity to bound is the loss $q_* - q_{\pi_k} \ge 0$, the difference between the value of the optimal policy and the (true) value of the policy computed by MoVI.

Theorem 1. *After $k+1$ iterations of MoVI, we have*

$$q_* - q_{\pi_{k+1}} \le \frac{1}{k+1}\Bigg[(I - \gamma P_{\pi_*})^{-1}(E_{k+1} + q_{k+1} - q_0)$$
$$-(I - \gamma P_{\pi_{k+1}})^{-1}\left(\sum_{j=0}^{k-1} \gamma^j E'_{k,j} + \sum_{j=0}^{k} \gamma^j P_{j:1}(T_{\pi_1} q_0 - q_0)\right)\Bigg].$$

To understand and then discuss this result, we provide a bound of a μ-weighted ℓ_1-norm of the loss: the norm is what one would want to control in a practical setting. Notice that we could similarly derive a bound for the μ-weighted ℓ_p-norm.

Corollary 1. *Let μ be the distribution of interest, and ν the sampling distribution. We introduce the following concentrability coefficient (the fraction being componentwise)*

$$C = \max_\pi \left\| \frac{d_{\pi,\mu}}{\nu} \right\|_\infty$$

Suppose that we initialize $h_0 = q_0 = 0$. At iteration $k+1$ of MoVI, we have

$$\|q_* - q_{\pi_{k+1}}\|_{1,\mu} \le \frac{C}{(k+1)(1-\gamma)}\Bigg(\|E_{k+1}\|_{1,\nu} + \sum_{j=0}^{k-1} \gamma^j \|E'_{k,j}\|_{1,\nu} + 2q_{\max}\Bigg).$$

Theorem 1 shows that $q_* - q_{\pi_k}$ depends on two error terms, E_k and a γ-discounted sum of $E'_{k,j}$. The first term corresponds to a sum of errors, that can then compensate, which is not the case in AVI (see Equation (1)). The normalization by $\frac{1}{k+1}$ reduces the variance of this term, and that can lead to convergence under some assumptions (see Section 4.2). However, the second term is more cumbersome. The terms $E'_{k,j}$ depend on sums of error weighted by composed kernels $P_{i:j}$. Would these kernels be arbitrary, this could lead to further variance reduction. However, the corresponding average is done over the state-action space in addition to over iterations, and the kernels are dependent of the error they weight, this dependency being hard to quantify. We further discuss this next.

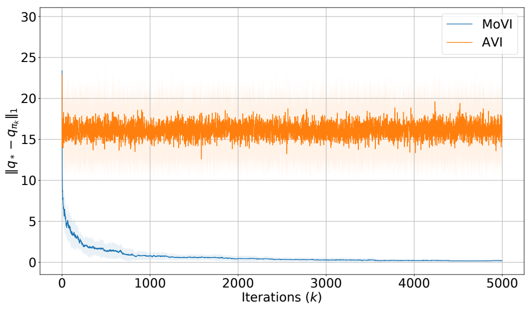

Figure 1: Illustration of the convergence of MoVI. We represent the empirical mean and standard deviation of the error over 100 MDPs.

Still, the algorithm can converge in practice, and we illustrate its behaviour on a simple case. We give ourselves a tabular representation of a randomly generated MDP, with access to a generative model. The approximation comes from the fact that the Bellman operator is sampled at each iteration (instead of being evaluated exactly); we compare it to AVI in the same scenario. We report the average error between q_{π_k} and q_* in Figure 1. This experiments illustrate how AVI oscillates with high error, while MoVI converges to q_*.

We note that our proof technique should hold with a constant β too (moving average instead of average). In this case, instead of having an average error ($k^{-1}E_k$), we would have a moving average of the (weighted) errors. This would not vanish asymptotically, even with zero-mean bounded noises ϵ_k, but this would still reduce the variance, and improve upon the AVI bound.

4.2 About the sample complexity

To better understand MoVI, we analyze its sample complexity in a simple case, Sampled-MoVI. In this setting, we have access to a generative model of the MDP and we give ourselves a tabular representation of the MDP. At each iteration of Sampled-MoVI, for each $(s,a) \in \mathcal{S} \times \mathcal{A}$, we sample a state $s' \sim P(\cdot|s,a)$ and perform the update from Equation (2) with only this state. We denote $\hat{T}_{\pi_k}$ the resulting sampled Bellman operator, $\hat{T}_{\pi_k} q(s,a) = r(s,a) + \gamma q(s', \pi_k(s))$. The error at iteration k is then, for each (s,a), $\epsilon_k(s,a) = \hat{T}_{\pi_k} q_{k-1}(s,a) - T_{\pi_k} q_{k-1}(s,a)$. It is thus zero-mean and centered. We provide a detailed pseudo-code in the Appendix.

We are interested in controlling the distance of our policy to the optimal policy, precisely in the norm $\|q_* - q_{\pi_k}\|_\infty$ at iteration k of Sampled-MoVI. We have, as a direct consequence of Thm. 1, that

$$\|q_* - q_{\pi_{k+1}}\|_\infty \leq \frac{1}{(k+1)(1-\gamma)} \Bigg(\|E_{k+1}\|_\infty + \sum_{j=0}^{k-1} \gamma^j \|E'_{k,j}\|_\infty + 2q_{\max} \Bigg). \quad (3)$$

Informally, using an Hoeffding argument, we have $k^{-1}\|E_k\|_\infty = \mathcal{O}(k^{-\frac{1}{2}})$. However, bounding a term $\max_{j \leq k} \|E'_{k,j}\|_\infty$ is more involved. This could typically be done using the Maximal Azuma-Hoeffding inequality. Yet, this requires the errors to be centered and bounded. In our case, the sequence of estimation errors $\{\epsilon_1(s,a), \dots \epsilon_k(s,a)\}$ is a martingale difference sequence with respect to the natural filtration $\mathcal{F}_{k-1}$ (generated by the sequence of states sampled from the generative model), that is $\mathbb{E}[\epsilon_k(s,a)|\mathcal{F}_{k-1}] = 0$. This is sufficient for controlling the term E_k, but the terms $E'_{k,j}$ are more difficult. Indeed, there, the errors are multiplied by a series of transition kernel matrices. For an arbitrary kernel P, independent of ϵ_k, we would have $\mathbb{E}[P\epsilon_k(s,a)|\mathcal{F}_{k-1}] = P\mathbb{E}[\epsilon_k(s,a)|\mathcal{F}_{k-1}] = 0$. Unfortunately $P_{\pi_{k+1}}$ depends on π_{k+1}, which is greedy with respect to h_k, which is computed using q_k and so depends on ϵ_k. Thus, the independence cannot be assessed. To control the error in Sampled-MoVI, we consequently make the following assumption.

Assumption 1. $\forall i,j \geq 1$, $\mathbb{E}\left[P_{j+i:j+1}\epsilon_j|\mathcal{F}_{j-1}\right] = 0$.

This assumption may seem very strong, as the dependency is hard to quantify. However, we have that $\pi_{k+1} \in \mathcal{G}(h_k) = \mathcal{G}(\frac{k}{k+1}h_{k-1} + \frac{1}{k+1}T_{\pi_k}q_{k-1} + \frac{1}{k+1}\epsilon_k)$. Thus, the influence of ϵ_k on π_k diminishes with time. Indeed, assuming that $\mathbb{E}\left[P_{j+i:j+1}\epsilon_j|\mathcal{F}_{j-1}\right] = o(\frac{1}{\sqrt{j}})$ should be enough to ensure convergence, but at a lower speed. We study numerically this assumption in Section 7.

Proposition 1. *Suppose Asm. 1 holds. After k iterations of Sampled MoVI, with probability at least* $1-\delta$

$$\|q_* - q_{\pi_k}\|_\infty \leq \frac{2r_{\max}}{(1-\gamma)^2}\left[\frac{1}{k} + \frac{3}{(1-\gamma)}\sqrt{\frac{2\ln\frac{4|\mathcal{S}||\mathcal{A}|}{\delta}}{k}}\right].$$

This result only holds under the strong Asm. 1. Under this setting (tabular representation, generative model), there exist algorithms with faster convergence [Wainwright, 2019]. However, they are not easily extandable beyond this setting, contrary to MoVI that can be easily turned into a practical large scale deep RL algorithm.

5 Momentum DQN

We now propose an extension of MoVI to Momentum-DQN, introducing stochastic approximation and using deep neural networks for function approximation. We base ourselves on Deep Q-Networks (DQN , Mnih et al. [2015]), using the same algorithmic structure. We propose an off-policy algorithm, using a replay buffer as in DQN: we can apply the Bellman evaluation operator to the estimated q-function in an off-policy manner.

We parametrize the q-function by an *online* network Q_θ of weights θ, and we keep a copy of these weights in a *target* network Q^- of weights θ^-. We additionally define the averaging network H_ϕ of weights ϕ, and their target counterparts H^- and ϕ^-. Momentum-DQN interacts in an online way with an environment collecting transitions $\{s,a,r,s'\} \in \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S}$, that are stored in a FIFO replay buffer $\mathcal{B}$. In DQN, the algorithm performs gradient descent to approximate the partial evaluation step by regressing an approximation of $T_* Q^-$, and periodically copies the weights of the online networks to the target networks. The loss minimized at each step is almost the same as in DQN, replacing an approximation of $T_* Q_k$ by an approximation of the evaluation operator of the greedy policy with respect to the averaging network, $T_{\mathcal{G}(H_\phi)}Q^-$. We define a regression target for Q_θ as

$$\hat{Q}(r,s') = r + \gamma Q^-(s', \operatorname{argmax} H^-(s',\cdot)),$$

and a regression loss

$$\mathcal{L}_q(\theta) = \hat{\mathbb{E}}_{\mathcal{B}}\left[\left(\hat{Q}(r,s') - Q_\theta(s,a)\right)^2\right], \quad (4)$$

with $\hat{\mathbb{E}}$ the empirical loss over a finite set. Then, we define a regression loss for the averaging network as an approximation of Equation (2). We use the general scheme from Equation (2) with a possibly variable mixture rate β_k. The regression target $\hat{H}$ for the averaging network is computed as

$$\hat{H}(s,a,r,s') = \beta_k H^-(s,a) + (1-\beta_k)\hat{Q}(r,s'),$$

which leads to a regression loss

$$\mathcal{L}_H(\phi) = \hat{\mathbb{E}}_{\mathcal{B}}\left[\left(\hat{H}(s,a,r,s') - H_\phi(s,a)\right)^2\right]. \quad (5)$$

Momentum-DQN interacts with the environment with the policy $\mathcal{G}_{e_k}(H)$ that is e_k-greedy with respect to H, the averaging network (e_k depends on k because we use a classic decreasing schedule for the exploration). During training, it minimizes losses $\mathcal{L}_q$ and $\mathcal{L}_h$ with stochastic gradient descent (or a variant), and update the target weights with the online weights every C gradient steps. A detailed pseudo-code is given in Algorithm 1, and we evaluate this algorithm in Section 7.2.

On the mixture rate. We aim at considering a rate close to the one of MoVI, $\beta_k = \frac{k}{k+1}$. Due to stochastic approximation, an iteration of Momentum-DQN does not match one iteration of MoVI, rather we should wait for several target updates before considering we have performed such an iteration. Consequently, we consider a rate such that $\beta_k = \frac{\lfloor k/\kappa \rfloor}{\lfloor k/\kappa \rfloor + 1}$, with κ a rate update period (an hyperparameter), that is the number of environment steps between each change of β.

Algorithm 1 Momentum-DQN

```
Require: $K \in \mathbb{N}^*$ the number of steps, $C \in \mathbb{N}^*$ the
    update period, $F \in \mathbb{N}^*$ the interaction period, $\kappa \in \mathbb{N}^*$
    the rate update period.
  Initialize $\theta$, $\phi$ at random
  $\mathcal{B} = \{\}$
  $\theta^- = \theta, \phi^- = \phi$
  for $k = 1$ to $K$ do
    Collect a transition $t = (s, a, r, s')$ from $\mathcal{G}_{e_k}(H_\phi)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{t\}$
    if $k \mod F == 0$ then
      $\beta_k = \frac{\lfloor k/\kappa \rfloor}{\lfloor k/\kappa \rfloor + 1}$
      On a random batch of transitions $B_{q,k} \subset \mathcal{B}$,
      update $\theta$ with one step of SGD of $\mathcal{L}_q$, see (4)
      On a random batch of transitions $B_{h,k} \subset \mathcal{B}$,
      update $\phi$ with one step of SGD of $\mathcal{L}_h$, see (5)
    end if
    if $k \mod C == 0$ then
      $\theta^- \leftarrow \theta$, $\phi^- \leftarrow \phi$
    end if
  end for
  return $\mathcal{G}(H_\phi)$
```

6 Related work and discussion

The closest approaches to MoVI are Speedy Q-Learning (SQL) [Azar et al., 2011] and Dynamic Policy Programming (DPP) [Azar et al., 2012] (generalized by Kozuno et al. [2019] as Conservative VI, with similar guarantees). Both approaches are extensions of AVI that also benefit from a similar compensation of errors along iterations. As fat as we know, they are the sole algorithms with this kind of guarantee. We first discuss extensively the links to SQL and DPP, before mentioning other (less) related works.

Algorithmic comparison. First, let us consider DPP, in the DPP-RL version[2] [Azar et al., 2012, Algorithm 2]. Define the scalar product on $\mathcal{A}$ for all policy π and q-value q as $\langle \pi, q \rangle(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q(s,a)$. DPP estimates a quantity $\psi_k \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$, as

$$\psi_k = \psi_{k-1} + T_{\pi_k} \psi_{k-1} - \langle \pi_k, \psi_{k-1} \rangle + \epsilon_k, \qquad (6)$$

with $\pi_k \in \mathcal{G}(\psi_k)$. Without error, $\psi_k(s,a)$ converges to $q_*(s,a)$ when a is the optimal action in state s, and to $-\infty$ otherwise. This makes difficult an extension of DPP to a function approximation setting (unbounded function).

Secondly, SQL updates a q-value q_k as

$$q_k = q_{k-1} + \frac{1}{k}(T_* q_{k-2} - q_{k-1}) + \frac{k-1}{k}(T_* q_{k-1} - T_* q_{k-2}). \qquad (7)$$

We then re-write SQL as an update on similar quantities as DPP. Let us define $\psi_k = k q_k$, and consider the policy $\pi_k = \mathcal{G}(q_k) = \mathcal{G}(\psi_k)$. SQL is then equivalent to

$$\psi_k = \psi_{k-1} + T_{\pi_k} \psi_{k-1} - \gamma P_{\pi_{k-1}} \psi_{k-2} + \epsilon_k. \qquad (8)$$

Finally, we also position MoVI in this setting. Here, we define ψ_k as $\psi_k = (k+1) h_k = \sum_{i=0}^{k} q_j$. We consider the sequence of policies $\pi_k = \mathcal{G}(h_k) = \mathcal{G}(\psi_k)$. With some work (detailed in Appendix) we can rewrite Equation (2) as an update on ψ_k as

$$\psi_k = \psi_{k-1} + T_{\pi_k} \psi_{k-1} - \gamma P_{\pi_k} \psi_{k-2} + \epsilon_k. \qquad (9)$$

Comparing MoVI, SQL and DPP through the prism of Eqs. (9), (6) and (8), we observe that these three schemes are similar. They all share the first part of their update in common, and differ only in the subtraction term – that allows for error compensation. This term is $\gamma P_{\pi_k} \psi_{k-2}$ in MoVI, which is replaced by a $\gamma P_{\pi_{k-1}} \psi_{k-2} = T_* \psi_{k-2}$ in SQL, and by $\langle \pi_k, \psi_{k-1} \rangle$ in DPP. This writing eases comparison, but we highlight that it is not how algorithms are defined initially, and implemented, except for DPP (SQL and MoVI do not require estimating an unbounded function).

Performance bounds. We now compare performance bounds of various algorithm. SQL and DPP both propagates averaged errors instead of errors, as they both satisfy[3]

$$\|q_* - q_{\pi_k}\|_\infty \le \frac{2\gamma}{k(1-\gamma)} \left(\sum_{j=1}^{k} \gamma^{k-j} \|E_j\|_\infty + \frac{8\gamma q_{\max}}{1-\gamma} \right),$$

[2]DPP considers general softmax policies, of which greedy policies are a special case, that correspond to DPP-RL.

[3]The bounds in the original papers differ slightly by their multiplicative constants, the one provided here is true for both.

This is to be compared to the bound for MoVI given in Eq. (3). MoVI, DPP and SQl enjoys similar bounds, the main difference being in the nature of the error terms. Both SQL and DPP depend on a term of the from $\sum_{j=0}^{k} \gamma^{k-j} \|E_k\|_\infty$, that is a discounted sum of the norm of averaged errors. On the other hand, in MoVI, we have the dependency in $\sum_{j=0}^{k} \gamma^{k-j} \left\| E'_{k,j} \right\|_\infty$, so not averaged errors, but averaged weighted errors. In the generative model setting, the bound is less favourable for MoVI. Indeed, in this case, the errors are zero-mean, so the dependence on their average in DPP and SQL is a strong advantage. However, we empirically show that MoVI behaves similarly to SQL and DPP in this case (see Sec. 7.1). In a more general case (ϵ_k corresponding to a regression error), none of the bounds can be easily instantiated, because the quantity we can hope to control is $\|\epsilon_k\|_2, \mu$, not $\|E_k\|_{2,\mu}$. This, we will check the algorithms' behaviors empirically.

From a practical point of view, neither SQL or DPP have been originally implemented in RL on large scale problems. A deep version of a variation of DPP[4] have been proposed by Tsurumine et al. [2017], but it is only applied on a small number of samples. The principal issue of a practical DPP is that it has to estimate ψ_k, a quantity that is asymptotically unbounded. It could then be applied on short training environments, when this value is updated a relatively small number of times, and stays numerically stable. However, on environments like the ALE, where one needs to compute millions of environments steps, DPP is likely to diverge, and fail due to numerical issues. In Section 7.2, we provide a experiment in a larger setting. We extened MoVI to deep learning, and, for the sake of somparison, we propose deep versions of SQL and DPP. These two last algorithms are variations of DQN that make use of updates in Equations (7) and (6) to define DQN-like regression targets. We could not obtain satisfying results with both of these implementations. Experimental results and details are given in Section (7) and in the Appendix.

Other related methods. MoVI shares also algorithmic similarities with other algorithms, Softened LSPI [Pérolat et al., 2016] and Politex [Lazic et al., 2019]. Pérolat et al. [2016] consider the zero-sum games setting, and propose a Policy Iteration (PI)-based algorithm. It relates to MoVI in the sense that it averages the q-values of consecutive policies. Politex is also a PI-scheme, where the policy is a softmax of the sum of all q-values. These two algorithms share the idea of averaging the q-values, but are derived from different principles. Pérolat et al. [2016] build their algorithm as a quasi-Newton method on the Bellman residual and rely heavily on linear parameterization, while Politex build upon prediction with expert advice, and deals with the average reward criterion, instead of the discounted one. Moreover, none of these two approaches offer the kind of guarantee about the propagation of averaged errors that DPP, SQL or MoVI have.

7 Experiments

In this Section, we present experimental results from MoVI and Momentum-DQN. First, we consider small random MDPs (Garnets), to check empirically Asm. 1 and to compare to DDP and SQL on a tabular setting, with access to a generative model. Then, we experiment Momentum-DQN on a subset of Atari games, and compare to DQN (a natural baseline) as well as deep versions of DPP and SQL. Further experimental details are provided in the appendix.

7.1 Garnets

A Garnet [Archibald et al., 1995, Bhatnagar et al., 2009] is an abstract MDP. It is built from three parameters (N_S, N_A, N_B). N_S and N_A are respectively the number of states and actions. The parameter N_B is the branching factor, the maximum number of states accessible from any other state. The transition probabilities $P(s'|s,a)$ are then computed as follows. For each state-action couple (s,a), N_B states $(s_1, \dots s_{N_B})$ are drawn uniformly without replacement. Then, $N_B - 1$ number are drawn uniformly in $(0,1)$ and sorted as $(p_0 = 0, p_1, \dots p_{N_B-1}, p_{N_B} = 1)$. The transition probabilities are assigned as $P(s_k|s,a) = p_k - p_{k-1}$ for each $1 \leq k \leq N_B$. The reward function is drawn uniformly in $(-1,1)^{N_S}$.

Assumption check. First, we want to check that Asm. 1 is reasonable. Given a step j of the algorithm and a size l, we compute an empirical estimate of $\mathbb{E}[P_{j+l:j+1}\epsilon_j]$. With Garnets, we have access to the transition kernel, so we can compute the error at step j, $\epsilon_j(s,a) = \hat{T}_{\pi_j} q_j(s,a) - T_{\pi_j} q_j(s,a)$. Given a fixed Garnet, we first compute the value q_j with MoVI. Then, on a number N of runs, we re-start MoVI from the same q_j, re-run the algorithm for l steps from there, and compute the values $P_{j+l:j+1,n}\epsilon_{j,n}(s,a)$, with $n \in [|1;N|]$. We get an estimate $\bar{\epsilon}_{l,N}$ of $\|\mathbb{E}[P_{j+l:j+1}\epsilon_j|\mathcal{F}_{j-1}]\|_\infty$,

$$\bar{\epsilon}_{l,N} = \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \left| \frac{1}{N} \sum_{n=1}^{N} P_{j+l:j+1,n}\epsilon_{j,n}(s,a) \right|.$$

We want to check that $\bar{\epsilon}_N \to 0$ when $N \to \infty$. For several values of l, We compute $\bar{\epsilon}_{l,N}$ for N between 0

[4] Specifically, it is the update described by Azar et al. [2012, Eq. (24)], that also lead to an asymptotically unbounded function, and thus to numerical instability.

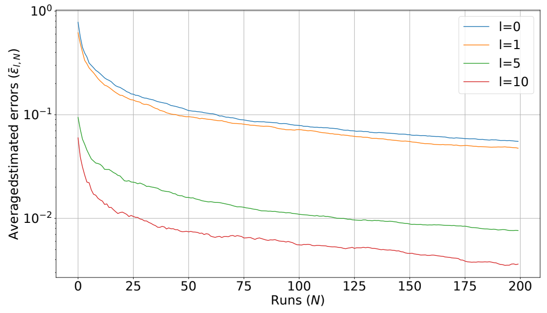

Figure 2: Evolution of the empirical weighted average error $\bar{\epsilon}_{l,N}$ with N (log scale) for different values of l. We need a convergence towards 0 for our assumption to be numerically verified, which seems to be the case.

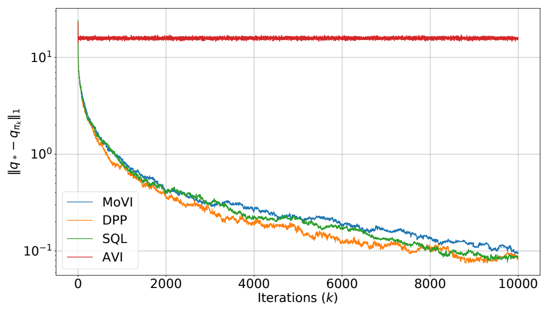

Figure 3: Error on the policy value of different ADP schemes. Each curve represents $\|q_* - q_{\pi_k}\|_1$, where π_k results from AVI, MoVI, SQL or DPP.

and 200, and average these results over 100 garnets. We report the evolution of the means of $\bar{\epsilon}_{l,N}$ (over Garnets) in Figure 2. We observe that the limit of $\bar{\epsilon}_{l,N}$ seems to be 0 for each l, which experimentally validates our assumption. With $l = 0$, we get the "natural" norm of errors (not multiplied by any matrix). We see here that, for every tested $l > 0$, the norm is lower than for $l = 0$, meaning that the policies kernels do not have a negative impact on the expected value, but seem to further reduce variance.

Algorithms comparison. We compare VI, MoVI, SQL and DPP on random Garnets, using the sampled version with a generative model described in Section 4.2. We run each algorithm on 100 Garnets, an we report the norm of the empirical error on the uniform distribution $\|q_* - q_{\pi_k}\|_1$. We can compute the exact value of π_k with access to the model. The four algorithms are compared in Figure 3. We observe an almost identical behaviour for MoVI, DPP, and SQL. They all converge towards v_* at roughly the same speed, while AVI oscillates around a sub-optimal policy.

7.2 Atari

Atari is a standard discrete-actions environment introduced by Bellemare et al. [2013] with a high dimensional state space. We use this environment to validate our Momentum-DQN architecture. Our baseline is DQN as it is implemented in the Dopamine library [Castro et al., 2018]. We used the same architecture and the same hyperparameters as DQN, and notably we used sticky action with a rate of 0.25 to introduce stochasticity as recommended by Machado et al. [2018], and our state consists in the stacking of the 4 last frames. Every 4 steps in the environment, we perform a gradient update on θ and ϕ. Every C=25000 environment steps, we update the target networks. We report the average undiscounted score obtained during learning on the last 250000 steps (named an iteration). On the figures, the thick line show this average score averaged on 5 random seeds, while the semi-transparent parts denote the standard deviation with respect to the seeds.

We evaluate Momentum-DQN on a subset of 20 Atari games. This games are selected to represent the categories from Ostrovski et al. [2017, Appendix A], excluding the hardest exploration ones – we have no claim in helping DQN in this setting. Here, we used a schedule of β_k as defined in Section 5, with $\kappa = 2500000$ that we tuned on a small subset of game (Asterix, Zaxxon, and Jamesbond). As an example, we give the comparison of Momentum-DQN and DQN on the game SpaceInvaders in Figure 4. In figure 5, we report the normalized improvement of Momentum-DQN over DQN using the Area Under the Curve (AUC) metric. These results show a clear improvement using Momentum. Momentum-DQN outperforms DQN on 16 games out of 20, with an average normalized improvement of 45%. It only under-performs DQN on three games by a low margin, while the improvement goes up to 200% for the game Seaquest. In the Appendix, we report the score obtained for the 20 games, along with experiments testing the influence of various β_k schedules.

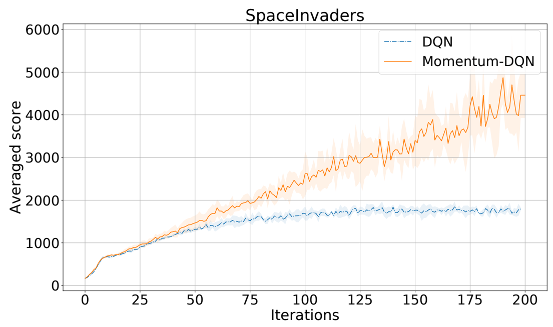

Figure 4: Scores obtained on SpaceInvaders by DQN (dashed-dotted, blue) and Momentum-DQN (orange).

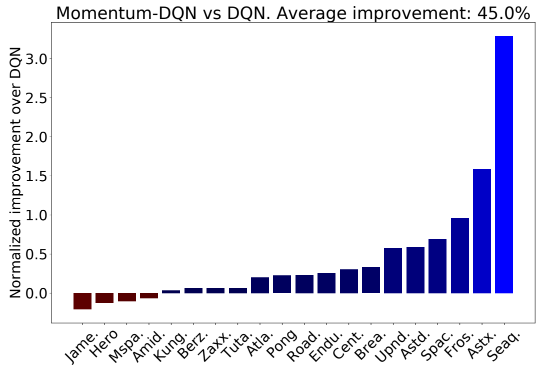

Figure 5: Normalized improvement of Momentum-DQN over DQN. We obtain an almost constant improvement on these 20 games.

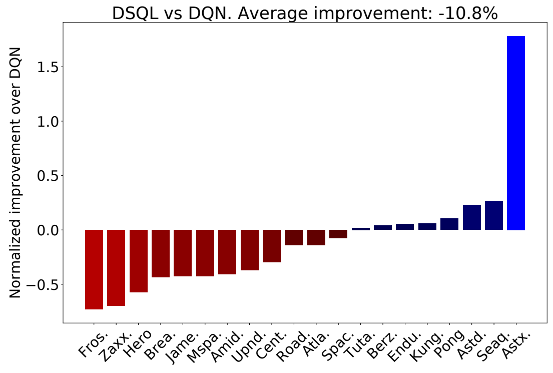

Figure 6: Normalized improvement of DSQL over DQN.

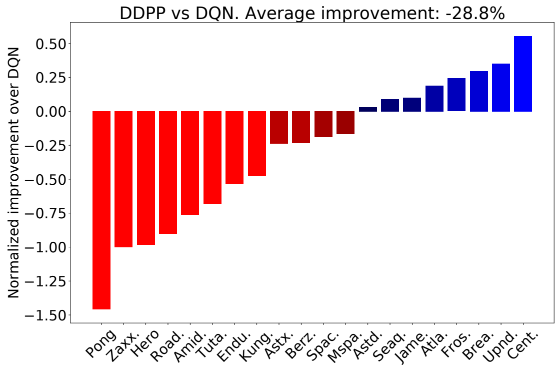

Figure 7: Normalized improvement of DDPP over DQN.

Deep-SQL and Deep-DPP. We implemented Deep versions of SQL and DPP (respectively DSQL and DDPP), that we tested on Atari, also based on the architecture and hyperparameters of Dopamine's DQN. For both algorithms, we derive an update rule based on the ADP scheme, using the same parametrization as DQN (we report specific equations in Appendix). We were however not able to obtain satisfying – i.e. competitive with DQN – scores with these algorithms. We report the experimental results of DDPP and DSQL versus DQN in Figures 6 and 7. We used the same parameters as for Momentum-DQN, in particular the same β_k schedule for DSQL. On these two graphs, we see that both DSQL and DDPP underperform DQN on most of the games.

For DDPP, the reason is quite simple, as the Q-network has to estimate a value that diverges to $-\infty$, causing heavy numerical issues, and the algorithms fails on most of the games after a few iterations. It is less clear why DSQL underperforms DQN. Our hypothesis is that Momentum-DQN enjoys a separate network that approximate the average of the q-values, while DSQL needs to compute its update from consecutive target. However, when using deep networks and stochastic approximation, the consecutive target networks cannot securely be associated to consecutive q-values computed in ADP, making the update in DSQL less reliable.

8 Conclusion

We introduced a new ADP scheme, MoVI, inspired by Momentum in gradient ascent. To adapt Momentum to RL, we made an analogy between the q-values in DP schemes and the gradient in gradient ascent methods, interpreting Momentum in RL as an averaging of consecutive q-function. We provided an anlysis of MoVI, showing that the Momentum brings compensation of errors to AVI. We also derived a partial analysis of the sample complexity when instantiated in the tabular case. These results are similar to what are to our knowledge the closest algorithms to MoVI, SQL and DPP. Our bound involves a more complicated averaging of errors, extensively discussed. Yet, we have shown that all algorithmic schemes behave similarly in toy problems. We advocated that MoVI is better suited for deep learning extensions and proposed Momentum-DQN, as well as natural deep extensions of DPP and SQL. With experiments on a representative subset of Atari games, we have shown that, contrary to DDPP and DSQL, momentum-DQN brings a clear improvement over DQN. Note that in principle, Momentum could be applied to any RL algorithm that estimates a value: a value-based algorithm like C51 [Bellemare et al., 2017], or an actor-critic (for example, SAC [Haarnoja et al., 2018]). It could also be extended straightforwardly to continuous action settings, replacing the critic by the average of successive critics. We plan to extend the idea of Momentum to other RL algorithms in future works.

References

TW Archibald, KIM McKinnon, and LC Thomas. On the generation of markov decision processes. *Journal of the Operational Research Society*, 46(3):354–361, 1995.

Mohammad G Azar, Mohammad Ghavamzadeh, Hilbert J Kappen, and Rémi Munos. Speedy q-learning. In *Advances in Neural Information Processing System (NeurIPS)*, pages 2411–2419, 2011.

Mohammad Gheshlaghi Azar, Vicenç Gómez, and Hilbert J Kappen. Dynamic policy programming. *Journal of Machine Learning Research*, 13(Nov):3207–3245, 2012.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 449–458, 2017.

Dimitri P Bertsekas and John N Tsitsiklis. *Neuro dynamic programming.* Athena Scientific Belmont, MA, 1996.

Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870, 2018.

Tadashi Kozuno, Eiji Uchibe, and Kenji Doya. Theoretical analysis of efficiency and robustness of softmax and gap-increasing operators in reinforcement learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2995–3003, 2019.

Nevena Lazic, Yasin Abbasi-Yadkori, Kush Bhatia, Gellert Weisz, Peter Bartlett, and Csaba Szepesvari. Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning (ICML)*, pages 3692–3702, 2019.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *International Conference on Machine Learning (ICML)*, pages 2721–2730, 2017.

Julien Pérolat, Bilal Piot, Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. Softened approximate policy iteration for markov games. In *International Conference on Machine Learning (ICML)*, pages 1860–1868, 2016.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 1994.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

Yoshihisa Tsurumine, Yunduan Cui, Eiji Uchibe, and Takamitsu Matsubara. Deep dynamic policy programming for robot control with raw images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1545–1550. IEEE, 2017.

Martin J Wainwright. Variance-reduced q-learning is minimax optimal. *arXiv preprint arXiv:1906.04697*, 2019.