The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)

Deep Conservative Policy Iteration

Nino Vieillard, Olivier Pietquin, Matthieu Geist

Google Research, Brain Team

Abstract

Conservative Policy Iteration (CPI) is a founding algorithm of Approximate Dynamic Programming (ADP). Its core principle is to stabilize greediness through stochastic mixtures of consecutive policies. It comes with strong theoretical guarantees, and inspired approaches in deep Reinforcement Learning (RL). However, CPI itself has rarely been implemented, never with neural networks, and only experimented on toy problems. In this paper, we show how CPI can be practically combined with deep RL with discrete actions, in an off-policy manner. We also introduce adaptive mixture rates inspired by the theory. We experiment thoroughly the resulting algorithm on the simple Cartpole problem, and validate the proposed method on a representative subset of Atari games. Overall, this work suggests that revisiting classic ADP may lead to improved and more stable deep RL algorithms.

1 Introduction

We consider the Reinforcement Learning (RL) problem with discrete actions, formalized with Markov Decision Processes (MDP) (Puterman 1994). Approximate Dynamic Programming (ADP) is a standard approach to practically solve MDPs when the state space is large. In this case, a popular – and rather successful – approach is to approximate the value function and/or the policy with function approximation, using techniques ranging from linear parametrization to deep neural networks. Recently, several algorithms inspired by ADP have shown unprecedented results on hard control tasks by using deep neural networks, that provide a great power of approximation. A lot of these algorithms can be seen as instances or variations of ADP algorithms, notably Value Iteration (VI) and Policy Iteration (PI). For example, Deep Q-Network (DQN) (Mnih et al. 2015) can be related to VI, while Soft Actor-Critic (SAC) (Haarnoja et al. 2018) or Trust Region Policy Optimization (TRPO) (Schulman et al. 2015) can be related to PI.

Conservative Policy Iteration (CPI) is a classic extension of PI introduced by Kakade and Langford (2002). Its main principle is to relax the improvement step in PI by being conservative with respect to the previous policies: instead of computing a sequence of deterministic greedy policies (as in PI), CPI computes a sequence of stochastic policies that are mixtures between consecutive greedy policies. While CPI has inspired some recent algorithms, such as TRPO (Schulman et al. 2015), it has never been implemented as such in practice, nor experimented on large challenging environments. In this paper, we propose a way to derive a practical algorithm from CPI, using neural networks as approximation scheme and relaxing the on-policy nature of CPI into off-policy learning through a VI-like scheme. We call the resulting algorithm Deep Conservative Policy Iteration, or DCPI (even if it is VI-based, to highlight the connection to CPI). It is specifically a conservative variation of DQN, but the proposed approach could be in principle applied to any pure-critic algorithm, notably the many variations of DQN.

After a short background, we develop the approximation steps that allow us to go from CPI to DCPI (Sec. 3), and give a detailed description of DCPI (Sec. 4). We then discuss some adaptive mixture rates in Sec. 5, inspired by the theory, and present experimental results on Cartpole and Atari environments in Sec. 6. The Appendix can be found in the long version of this paper (Vieillard, Pietquin, and Geist 2019).

2 Background and notations

We classically frame RL with an infinite horizon discounted MDP, a tuple $\{\mathcal{S}, \mathcal{A}, P, r, \gamma\}$ where $\mathcal{S}$ is the state space[1], $\mathcal{A}$ the finite action space, $P \in \Delta_{\mathcal{S}}^{\mathcal{S}\times\mathcal{A}}$ the Markovian transition kernel, $r \in [-R, R]^{\mathcal{S}\times\mathcal{A}}$ a bounded reward function, and $\gamma \in (0, 1)$ a discount factor. A stochastic policy π associates to each state s a distribution over actions $\pi(\cdot|s)$. We write $P_\pi(s'|s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[P(s'|s, a)]$ for the stochastic kernel associated to π, and $r_\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s, a)]$ the expected discounting reward for starting in s and following π. The value $v_\pi \in \mathbb{R}^{\mathcal{S}}$ of a policy is, for all $s \in \mathcal{S}$,

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

where $\mathbb{E}_\pi$ designates the expected value over all trajectories produced by π. The value function of a policy is the unique

[1] We assume it finite, for the ease of notations, but what we present extends to the continuous case.

fixed point of the Bellman evaluation operator associated to this policy, defined for each $v \in \mathbb{R}^{\mathcal{S}}$ as $T_\pi v = r_\pi + \gamma P_\pi v$. From this operator, one can define the Bellman optimality operator for each $v \in \mathbb{R}^{\mathcal{S}}$, $T_\star v = \max_\pi T_\pi v$. $T_\star$ admits as its unique fixed point the optimal value $v_\star$. A policy is said to be greedy w.r.t. to a value function v if $T_\pi v = T_\star v$, the set of all such policies is written $\mathcal{G}v$. A policy $\pi_\star$ is optimal with value $v_{\pi_\star} = v_\star$ when $\pi_\star \in \mathcal{G}v_\star$. To any policy π, we also associate the quality function q_π, for each $(s,a) \in \mathcal{S} \times \mathcal{A}$

$$q_\pi(s,a) = r(s,a) + \mathbb{E}_{s' \sim P(\cdot|s,a)}[v_\pi(s')],$$

which behaves similarly to the value function in the sense that $T_\pi q_\pi = q_\pi$ and $T_\star q_{\pi_\star} = q_{\pi_\star} = q_\star$ (with a slight abuse of notation). We can also define the set of policies that are greedy w.r.t. any function $q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ that we write $\mathcal{G}q = \operatorname{argmax}_a q(\cdot, a)$. It is useful in practice because a policy can be greedy to a q-function even if the model (the transition kernel) is unknown.

Finally, the advantage of a policy π, A_π, is defined as $A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$, and we write $d_{\pi,\mu} = (1-\gamma)\mu(I - \gamma P_\pi)^{-1}$ the discounted cumulative occupancy measure induced by π when starting from a distribution μ of states (distributions being written as row vectors).

3 Relaxing CPI

In this section, we describe the process that leads from CPI, a mainly theoretical dynamic programming algorithm, to a variant that can be combined with deep networks in an off-policy manner.

3.1 Ideal CPI

We first turn to the description of the CPI algorithm. We start by introducing the classic Approximate Policy Iteration (API) (Bertsekas and Tsitsiklis 1996), an iterative scheme that takes as input a distribution μ of states, and that computes at each iteration k a new policy

$$\pi_{k+1} = \mathcal{G}q_k,$$

where q_k is an approximation of q_{π_k} computed with states sampled from μ. An error on the greedy step $\mathcal{G}$ can be considered, but this error only appears when considering a infinite action space or when the greedy policy is approximated (for example with a cost-sensitive classifier). Here, we consider a finite action space, the greediness with respect to a q-function is exact.

CPI was first proposed by Kakade and Langford (2002). At each iteration k, CPI uses a mixture coefficient α_k to compute a stochastic mixture of all the previous greedy policies,

$$\pi_{k+1} = (1-\alpha_{k+1})\pi_k + \alpha_{k+1}\mathcal{G}q_k, \tag{1}$$

where q_k is still an approximation of q_{π_k}. This algorithm comes with strong theoretical guarantees, in particular the mixture rate can be chosen so that Eq. (1) guarantees an improvement of the expected value of the policy value, as shown by Kakade and Langford (2002) and Pirotta et al. (2013). In these works, the error on the value function estimation is supposed bounded, and the mixture rate depends on this bound. These theoretical guarantees rely on the fact that at each iteration k, the approximations are computed on the distribution $d_{\pi_k,\mu}$, where μ is the starting distribution of states, something far from being practical and making CPI inherently on-policy. CPI and its extension Safe Policy Iteration (SPI) (Pirotta et al. 2013) have only been experimented on tabular toy problems, with at most linear function approximation, in a very controlled manner (Pirotta et al. 2013; Scherrer 2014).

We will next introduce approximations that allow for an actual implementation using deep learning in an off-policy setting, but keeping the essence of CPI, that is regularizing the greediness. The question of the choice of the mixture rate will be studied later.

3.2 Approximating towards practicality

Approximating the value First, as said before, the value function has to be approximated. As the distributions $d_{\pi_k,\mu}$ are impractical, one classically computes an estimate q_k of the quality function q_{π_k}, with states sampled from a fixed state distribution or gathered during learning. The quality function can be estimated either by rollouts – but this is quite sample inefficient – or for example by using an algorithm such as LSTD (Bradtke and Barto 1996) – but that would require a linear parametrization. In any case, we can consider an error ϵ_k on this approximation, resulting in the scheme

$$\begin{cases} q_k = q_{\pi_k} + \epsilon_k \\ \pi_{k+1} = (1-\alpha_{k+1})\pi_k + \alpha_{k+1}\mathcal{G}q_k. \end{cases} \tag{2}$$

Temporal differences A classic approach is Temporal Difference (TD) learning, that estimates $q_k(s,a)$ by performing a regression on targets of the form $r(s,a) + \gamma \sum_{a' \in \mathcal{A}} \pi_k(a'|s') q_{k-1}(s',a')$. This can be written formally as computing $q_{k+1} = T_{\pi_k} q_{k-1} + \epsilon_k$. Practically, one can consider doing m-steps returns (Sutton 1988), which from an abstract perspective is $q_k = T^m_{\pi_k} q_{k-1}$, as done in Modified Policy Iteration (MPI) (Puterman and Shin 1978), or even Approximate MPI (Scherrer et al. 2015). This results in the scheme

$$\begin{cases} q_k = T^m_{\pi_k} q_{k-1} + \epsilon_k \\ \pi_{k+1} = (1-\alpha_{k+1})\pi_k + \alpha_{k+1}\mathcal{G}q_k. \end{cases}$$

Note that with $m = \infty$, it falls back to Eq. (2), and with $m = 1$, it becomes similar to VI, where the greediness has been regularized. Specifically, with $m = 1$ and $\alpha_k = 1$, this reduces to AVI (Approximate VI). In addition to allow using TD-learning, this also allows to learn in an off-policy manner (without off-policy correction if $m = 1$, as we work with state-action value functions).

Approximating the mixture Computing π_k would require remembering every q_i computed for $i \in [0,k]$, and this is not feasible in practice. Instead, we approximate the mixture, which adds a new source of errors. This can be done, for example, by minimizing an expected Kullback-Leibler (KL) divergence between a parametrization of π_{k+1} and the mixture. It can be written formally as

$$\begin{cases} q_k = T^m_{\pi_k} q_{k-1} + \epsilon_k \\ \pi_{k+1} = (1-\alpha_{k+1})\pi_k + \alpha_{k+1}\mathcal{G}q_k + \epsilon'_{k+1}. \end{cases} \tag{3}$$

This approximate dynamic programming scheme can then be instantiated into an off-policy Deep RL algorithm; we detail this process in Section 4.

3.3 Theoretical insights

The scheme depicted on Eq. (3) no longer enjoys the theoretical guarantees of CPI, as we relax some of its components (for example, partial policy evaluation or more freedom on how samples are gathered for learning). We give a partial analysis of this relaxed scheme in the Appendix, and here, we discuss its main results. Without errors ($\epsilon_k = \epsilon'_k = 0$), we show in the Appendix that v_{π_k} will converges linearly to v_*. With $\alpha_k = 1$ (this corresponds to MPI), the scheme benefits from a γ-contraction and it leads to a bound $\|v_* - v_{\pi_k}\|_\infty = \mathcal{O}(\gamma^k)$. With $\alpha_k < 1$, we obtain an η_k-contraction with $\eta_k = 1-\alpha_k(1-\gamma)$. If α_k does not go too fast towards zero, this would also lead to linear convergence. Indeed, using the fact that $\ln(1-x) \leq -x$ for $x \in (0,1)$,

$$\prod_{i=1}^{k} \eta_k = \exp \sum_{i=1}^{k} \ln(1-\alpha_i(1-\gamma)) \leq \exp(-(1-\gamma)\sum_{i=1}^{k} \alpha_i).$$

Therefore, this would lead to a bound $\|v_* - v_{\pi_k}\|_\infty = \mathcal{O}(\prod_{i=1}^{k} \eta_i) = \mathcal{O}(\exp(-(1-\gamma)\sum_{i=1}^{k} \alpha_i))$. If we still have a linear convergence, it is slower as long as $\alpha_k < 1$, which was to be expected without approximation error. However, at least this scheme does not break convergence.

With errors, we conjecture that we would obtain a bound close to the one of AMPI (Scherrer et al. 2015, Thm. 7), maybe with a larger propagation of errors (much like the convergence is slower, in the exact case), and so worse than the original bound of CPI (Kakade and Langford 2002; Scherrer 2014) (notably, with bigger concentrability coefficient). This is to be expected, the bound of CPI relies heavily on using $m = \infty$, on how the approximation error is plugged in the approximate dynamic scheme, and on using the $d_{\pi,\mu}$ distribution to sample transitions for learning approximations, three things that we relax. Yet, we still think that relaxing greediness is worth experimentally speaking, and that much remains to be done regarding its theoretical understanding.

4 Deep CPI

We now turn to the actual practical algorithm, DCPI. The basic idea is to define an instance of the update in Eq. (3) where the value function and the policy are parametrized via neural networks. We will focus on the case $m = 1$ (a regularized VI scheme), so we can apply the evaluation operator to the estimated q-function in an off-policy fashion without correction. It could be extended to the case $m > 1$ by simply using an off-policy correction method such as importance sampling. Note that focusing on $m = 1$ makes our algorithm a regularized VI-scheme and not a PI-scheme, but we keep the name DCPI to highlight the connection to CPI.

We parametrize the q-function and the policy by two *online* networks q_θ and π_ω, where θ and ω denote the weights of the respective networks. In a similar way to DQN, we define two *target* networks, q^- and π^-, whose weights are respectively θ^- and ω^-. DCPI introduces stochastic approximation by acting in an online way, meaning that transitions $(s, a, r, s') \in \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S}$ from the environment are collected during training. Transitions are stored in a FIFO replay buffer $\mathcal{B}$.

We write the two updates from Eq. (3) as optimization problems. The evaluation step consists of a regression problem, trying to minimize a quadratic error between q_θ and an approximation of $T^m_{\pi_\omega} q^-$. Recall that we now use $m = 1$. From this, denoting $\hat{\mathbb{E}}$ the empirical mean over a finite set, we can define a regression loss function $\mathcal{L}_q(\theta)$ for the value weights as

$$\hat{\mathbb{E}}\left[\left(r + \gamma \sum_{a' \in \mathcal{A}} \pi^-(a'|s') q^-(s', a') - q_\theta(s, a)\right)^2\right], \quad (4)$$

where the empirical average is computed over all transitions $(s, a, r, s') \in \mathcal{B}$ (recall that there is no need for these transitions to be sampled according to π^-, as we learn off-policy). The improvement step requires approximating a distribution over actions for each state. One way to do that is to minimize the expected value over the states of the expected KL divergence between the online policy network and the stochastic mixture. This leads to a loss function $\mathcal{L}_\pi(\omega)$ on the policy weights,

$$\hat{\mathbb{E}}\left[\mathrm{KL}\left((1-\alpha)\pi^-(\cdot|s) + \alpha\mathcal{G}(q_\theta)(\cdot|s) \| \pi_\omega(\cdot|s)\right)\right], \quad (5)$$

where the empirical average is computed over all states $(s, \dots) \in \mathcal{B}$. We minimize both $\mathcal{L}_q$ and $\mathcal{L}_\pi$ with a fixed number of steps of batch-SGD (or a variant), and update the target networks with the weights of the online networks. Each gradient step is performed after a fixed number (the interaction period F) of transitions are collected from the environment. Note that the use of a replay buffer makes our algorithm off-policy: the samples used to evaluate π_w originate independently from older policies. During training we sample transitions with $\pi_{\omega,\varepsilon}$, the policy which chooses a random action uniformly on $\mathcal{A}$ with probability ε and follows π_ω with probability $1 - \varepsilon$ (recall that π_ω is itself stochastic). A detailed pseudo-code is given in Algorithm 1.

Connection to DQN Despite its actor-critic look, DCPI can simply be seen as a variation of DQN. Indeed, note that with $\alpha = 1$, if π_ω is exactly computed (*i.e.* if $\pi_w = \mathcal{G}q_\theta$), DCPI reduces to DQN.

5 Choosing the mixture rate

Algorithm 1 does not give a way to choose the mixture rate, and this section studies different manners to do it. The natural idea is to choose a constant rate which experimentally (see Section 6.1) seems to improve stability, but comes at a great cost in terms of sample efficiency. Another possibility is to choose a decaying rate, for example with a hyperbolic schedule, or – and that is what we focus on – choosing an adaptive rate inspired from the literature on CPI.

Algorithm 1 DCPI

Require: $K \in \mathbb{N}^\star$ the number of steps, $C \in \mathbb{N}^\star$ the update period, $F \in \mathbb{N}^\star$ the interaction period

1: Initialize θ, ω at random
2: $\mathcal{B} = \{\}$
3: $\theta^- = \theta, \omega^- = \omega$
4: **for** $k = 1$ **to** K **do**
5: Collect a transition $t = (s, a, r, s')$ from $\pi_{\omega,\varepsilon}$
6: $\mathcal{B} \leftarrow \mathcal{B} \cup \{t\}$
7: **if** $k \mod F == 0$ **then**
8: On a random batch of transitions $B_{q,k} \subset \mathcal{B}$, update θ with one step of SGD of $\mathcal{L}_q$, see (4)
9: On a random batch of transitions $B_{\pi,k} \subset \mathcal{B}$, update ω with one step of SGD of $\mathcal{L}_\pi$, see (5)
10: **end if**
11: **if** $k \mod C == 0$ **then**
12: $\omega^- \leftarrow \omega$, $\theta^- \leftarrow \theta$
13: **end if**
14: **end for**
15: **return** π_ω

CPI adaptive rate Kakade and Langford (2002) provide a rate for CPI that guarantees an improvement of the policies, by choosing $\alpha = \frac{(1-\gamma)A^{\bar{\pi}}_{\pi,\mu}}{4R}$. Here, we write $\bar{\pi} = \mathcal{G}q_\pi$ the greedy policy with respect to q_π, and $A^{\bar{\pi}}_{\pi,\mu}$ the advantage of the greedy policy ($\bar{\pi}$) over the previous one (π), that is $A^{\bar{\pi}}_{\pi,\mu} = \sum_{s\in\mathcal{S}} d_{\pi,\mu}(s) A^{\bar{\pi}}_\pi(s)$ with $A^{\bar{\pi}}_\pi(s) = \sum_{a\in\mathcal{A}} \bar{\pi}(a|s) A_\pi(s,a)$. Recall that R is the maximum possible reward. We can estimate close quantities over a batch $B \subset \mathcal{B}$ at step k in the sense of Algorithm 1. We compute $\hat{A}_k(s) = \max_{a\in\mathcal{A}} q_\theta(s,a) - \sum_{a\in\mathcal{A}} \pi_\omega(a|s) q_\theta(s,a)$ as an estimate of $A^{\bar{\pi}}_\pi(s)$, and $\hat{\mathbb{A}}_k = \hat{\mathbb{E}}_{(s,\dots)\in B}[\hat{A}_k(s)]$ as an estimate of $A^{\bar{\pi}}_{\pi,\mu}$. The term $R/(1-\gamma)$ can be approximated by an estimate $\hat{Q}_k$ of $\|q_\pi\|_\infty$, which is consistent with corollary 3.6 of Pirotta et al. (2013). We compute it over a batch with $\hat{Q}_k = \max_{(s,a,\dots)\in B} |q_\theta(s,a)|$. For simplicity and to add a degree of freedom, we replace the constant factor $1/4$ by an hyperparameter α_0 that allows us to directly control the amplitude of our mixture rate. To compensate the fact that we compute our approximation over (potentially small) batches, we use a moving average m_k and a moving maximum Q^+_k. This leads to

$$\begin{cases} m_k = \beta_1 m_{k-1} + (1-\beta_1)\hat{\mathbb{A}}_k \\ Q^+_k = \max\left(\beta_2 Q^+_{k-1}, \hat{Q}_k\right) \end{cases}, \quad \alpha^{cpi}_k = \alpha_0 \frac{m_k}{Q^+_k}, \tag{6}$$

with $\beta_1, \beta_2 \in (0,1)$ typically close to 1.

SPI adaptive rate Pirotta et al. (2013) propose an improvement of CPI, Safe Policy Iteration. They provide a better bound on the policy improvement based on the mixture rate $\alpha = \frac{(1-\gamma)^2 A^{\bar{\pi}}_{\pi,\mu}}{\gamma\|\bar{\pi}-\pi\|_\infty \Delta A^{\bar{\pi}}_\pi}$, with $\Delta A^{\bar{\pi}}_\pi = \max_{s\in\mathcal{S}} A^{\bar{\pi}}_\pi(s) - \min_{s\in\mathcal{S}} A^{\bar{\pi}}_\pi(s)$, and with $\|\bar{\pi}-\pi\|_\infty = \max_{s\in\mathcal{S}} \sum_a |\pi(a|s) - \pi^-(a|s)|$ the maximum total variation between policies. We can approximate these quantities with the same methods used to obtain Eq. (6). Using the value $\hat{A}_k$ described previously, we compute an estimate of $\Delta A^{\bar{\pi}}_\pi$ by subtracting $\hat{A}_{k,\min} = \min_{(s,\dots)\in B} \hat{A}_k(s)$ to $\hat{A}_{k,\max} = \max_{(s,\dots)\in B} \hat{A}_k(s)$. Note that in addition to the previous approximations, we also include the total policy variation in the α_0 hyperparameter, as $\|\bar{\pi}-\pi\|_\infty \leq 2$. Using moving approximations, we obtain

$$\begin{cases} M^+_k = \max\left(\beta_2 M^+_{k-1}, \hat{A}_{k,\max}\right) \\ M^-_k = \min\left(M^-_{k-1}/\beta_2, \hat{A}_{k,\min}\right) \end{cases}, \qquad \alpha^{spi}_k = \alpha_0 \frac{m_k}{M^+_k - M^-_k}. \tag{7}$$

Bounding SPI The SPI mixture rate from Eq. (7) gives a rate that is not bounded. To keep our rate below 1, we propose a simple variation

$$\alpha^{adx}_k = \alpha_0 \frac{m_k}{M^+_k}. \tag{8}$$

From the fact that $\hat{A}_k(s)$ are positive numbers, it is immediate that α^{adx} is a "little more conservative" version of α^{spi}, with $\alpha^{adx} \leq \alpha^{spi}$ and $\alpha^{adx} \leq 1$. In fact, the advantage function can be linked to the functional gradient of the expected value function, respectively to the policy (see Scherrer and Geist (2014) who interpret CPI as a policy gradient boosting approach) and this rate is similar to the one the Adamax (Kingma and Ba 2015) algorithm would give (up to the fact that our rate is global, not component-wise) – hence the name.

About the batch The adaptive rate is computed using a batch of transitions from the replay buffer, and an important question is *which* batch to choose. In Algorithm 1, two different batches of transitions are defined: $B_{q,k}$ a batch of transitions used to estimate q_θ, and $B_{\pi,k}$ used to estimate π_ω. Our approach is, as the rate needs to adapt with respect to the current policy, to use $B_{\pi,k}$ to compute the rate. That means that, at iteration k in Algorithm 1, α_k and $\hat{\nabla}_\omega \mathcal{L}_\pi$ (the approximation of the gradient of $\mathcal{L}_\pi$ computed at line 9 of Algorithm 1) are computed with the same batch of transitions.

6 Experiments

In this section, we experimentally study DCPI on several environments. The method we propose is general, and could be used to regularize any pure-critic algorithm, by adding an actor to it. For this experimentation, we consider DCPI as a variation of DQN, and take DQN as our baseline. In principle, our method could extend to other frameworks, such as Rainbow (Hessel et al. 2018) or Implicit Quantile Networks (IQN) (Dabney et al. 2018), which are extensions to DQN. We start this experiment by an intensive test on Cartpole, a light environment that allows us to exhibit various behaviours of DCPI, such as stability over random seeds, convergence speed, or efficiency of the proposed mixture rate. We then conduct an experiment on Atari, to observe the effects of scaling up.

6.1 Cartpole

Cartpole is a classic control problem introduced by Barto, Sutton, and Anderson (1983). In this setup, the agent needs to balance a vertical pole by controlling its base (the cart) along one dimension, by applying a force on the cart of -1 or $+1$. We use the version of Cartpole implemented in OpenAI Gym (Brockman et al. 2016), with a maximum steps limit raised to 500 steps instead of a more classic 200, to make the task harder and get more accurate observations. The agent gets a reward of $+1$ while the pole is in the air, and 0 when it touches the ground.

Although CartPole is considered an "easy" problem in RL, it is cheap to run in computation time, so we use it as a test-bed to perform studies on the influence of our hyperparameters. Such studies would be prohibitive in cost on larger environments such as Atari. Our approach is to modify the DQN algorithm without changing its parameters so as to analyze how our framework modifies its learning behaviour. Our baseline is the DQN provided in the Dopamine library (Castro et al. 2018), and we use the hyperparameters provided here for Cartpole. Notably, we used the same network architecture for the q-network and the policy network and two identical Adam optimizers; we compute a gradient step every $F = 4$ interactions with the environment, and update the target networks every $C = 100$ interactions. Full parameters are reported in the Appendix. Our first observation is that this version of DQN is not very efficient on this problem, as it greatly lacks stability, be it over random seeds or over time (see Figure 1). This instability could probably be tempered by a better tuning of hyper parameters, but our goal is to verify the stabilizing effects of CPI, so we keep them as is.

Our method introduces three new hyperparameters: α_0, β_1, and β_2, described precisely in Section 5. To choose β_1 and β_2, we consider that our estimate of the advantage should be stable between two updates of the target networks. As this update occurs every 100 steps, and the size of the window for our moving average is $1/(1-\beta_1)$, this leads us to choose $\beta_1 = 0.99$. To increase stability, we choose a slower moving average in the denominator with $\beta_2 = 0.9999$. The ratio $(1-\beta_1)/(1-\beta_2) = 100$ is classic, it is for example consistent with the defaults parameters of Adam (Kingma and Ba 2015). We did a parameter search over α_0, with values ranging from $1e-3$ to 1, and tested the α^{cpi} and α^{adx} heuristics for an adaptive rate described in Section 5, Eqs. (6) and (8), in addition to a constant rate. The results for α^{spi} are similar to α^{adx}, and provided in the Appendix.

Results presented in Figure 1 and 2 are computed as follows: every 1000 training steps, an *iteration* in this context, we report the averaged undiscounted score per episode over these 1000 steps. The results are averaged over 50 different random seeds: the thick line indicates the empirical mean, while the semi-transparent areas denote the standard deviation of the score over the seeds.

Results with a constant rate (see Figure 1) show a strong increase of stability with small mixture rates ($\alpha_0 = 0.001$), with a cost in speed. With a higher learning rate, we obtain a faster convergence, but we loose stability. This introduces a speed/stability dilemma, and using adaptive rates (see Figure 2) allows us to get the best of both worlds. In a good case

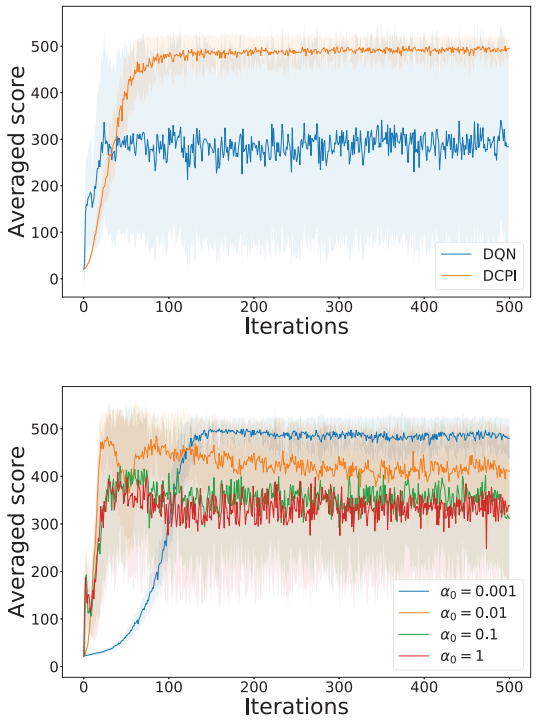

Figure 1: **Top:** comparison of the averaged training scores of DCPI with CPI rate and $\alpha_0 = 0.1$ (orange) against DQN (blue). **Bottom:** DCPI on Cartpole with constant rates for 4 values of α_0.

– CPI adaptive rate with $\alpha_0 = 0.1$, see Figure 1 (top) – we can keep the stability of the small constant mixture rates, while benefiting from a relatively fast convergence, and here DCPI shows a clear improvement on DQN on stability and average performance: DCPI is able to stabilize at an average score of 480 (on a maximum of 500) with a low standard deviation around 20, while DQN stabilizes around 300, with a standard deviation of approximately 200. Remarkably, even for $\alpha = 1$ (see Figure 1), *i.e.* when the stochastic mixture is not conservative and the regularization only comes from approximating the greediness, DCPI yields a slight improvement on stability over DQN. This can be seen as the distillation of the greedy policy, and is here less effective than a mixture scheme.

6.2 Atari

Atari is a challenging discrete-actions control environment, introduced by Bellemare et al. (2013) consisting of 57 games. We used sticky actions to introduce stochasticity as recommended by Machado et al. (2018). In a similar way to our Cartpole experiments, we used the DQN implementation from the Dopamine library as our baseline, keeping the parameters given in this library – much more optimized than the one for Cartpole. We compare against DQN's baseline score given in Dopamine. The parameters are detailed in the Appendix. In particular, the states stored in the replay buffer consist of stacks of 4 consecutive observed frames. With the

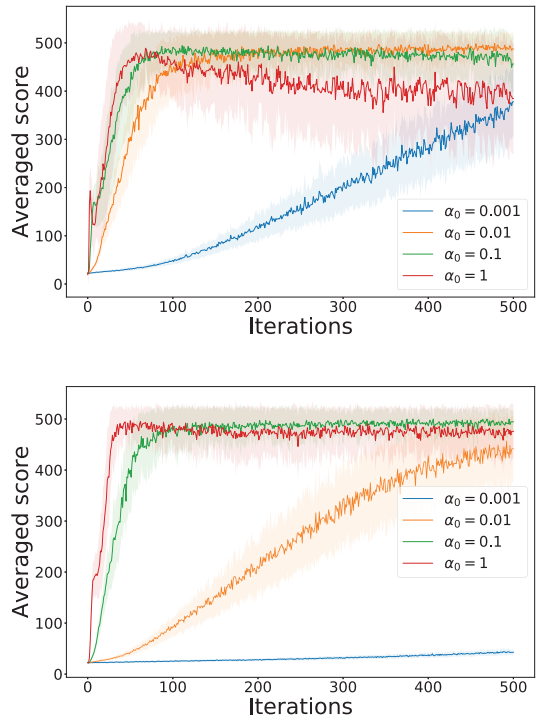

Figure 2: **Top:** DCPI on Cartpole with Adamax rates for 4 values of α_0. **Bottom:** DCPI on Cartpole with CPI rates for 4 values of α_0.

same arguments as in Section 6.1 we chose $\beta_1 = 0.9999$ and $\beta_2 = 0.999999$. After a small hyperparameter search on a few games (Pong, Asterix and Space Invaders), we chose $\alpha_0 = 1$ and the Adamax mixture rate (see Eq. (8)). Provided results are computed in a similar manner to the ones from Cartpole, except that here, an *iteration* represents 250000 environment steps. The results are averaged over 5 different random seeds.

For Atari, we also found empirically that interacting with the policy $\pi_{q,\varepsilon}$ that is ε-greedy with respect to q_θ improved performance over playing with π_ω. This is taken into account in the provided results. This can be seen as an optimistic controller regarding the stochastic policy π_ω, as both policies π_ω and $\pi_{q,\varepsilon}$ converge to the same behavior in the exact case. It can also be seen as a regularization of the Bellman optimality operator used in DQN, without changes to the way samples are gathered.

We tested DCPI on a representative subset of 20 Atari games, chosen from the categories described in (Ostrovski et al. 2017, Appendix A), excluding the hardest exploration games with sparse rewards – our algorithm has no ambition to help with exploration. All results are provided in the Appendix. DCPI yields a clear improvement on performance on a large majority of those games, outperforming DQN on 15 games over 20. Note that choosing a lower rate α_0 could increase stability and final performance, but also lower convergence speed. We chose to use rather aggressive adaptive rates on Atari due to constraints on computing time.

As a matter of illustration, Figure 3 provides three games where DCPI attains a higher score than DQN: Seaquest, Frostbite, and Breakout. All other games are reported in the Appendix. We also report on Figure 4 a comparison summary of DQN vs DCPI on all considered games. We used the Area Under the Curve (AUC) metric. For each game, we compute the sum of all averaged returns obtained during training, respectively S_{dcpi} and S_{dqn}, and we report the values for $(S_{dcpi} - S_{dqn})/|S_{dqn}|$.

7 Related work and discussion

The proposed approach is related to actor-critics in general, being itself an actor-critic. It is notably related to TRPO (Schulman et al. 2015), that introduced a KL penalty on the greedy step as an alternative to the stochastic mixture of CPI. This is indeed very useful for continuous actions, but probably unnecessary for discrete actions, the case considered here. Moreover, TRPO is an on-policy algorithm, while the proposed DCPI approach is off-policy. This explains that we do not consider it as a baseline in Section 6, but it would have been probably less sample efficient. As far as we know, there is no DQN-like TRPO algorithm, thus comparing our mixture-based DQN to one that KL-regularizes greediness would have required introducing a new algorithm.

The principle of regularizing greediness in actor-critics is quite widespread, be it with a KL divergence constraint (TRPO), a clipping of policies ratio (PPO, Schulman et al. (2017)), entropy regularization (SAC), or even following policy gradient, for example. The common point of these approaches is that they focus on continuous action spaces. In the discrete case, considering a stochastic mixture is quite natural, acknowledging that its extension to the continuous case is not easy.

Performance-wise, the experiments on Cartpole show a clear improvement for DCPI over DQN: DCPI is able to reach a higher score in average, with a lower variance and a lower sensitivity to the random seed. These experiments validate the stabilizing power of CPI and its expected behaviour with respect to the mixture rate, and the consistency of the considered adaptive rates. On Atari, even if results are game-dependent, we observe an improvement on the majority of the games. Note that the improvement in score is quite clear (the score is more than doubled on some games, like Seaquest or Asterix), but the learning is not stabilized as it is in Cart-Pole. As mentioned in Section 6.2, using a smaller (constant) mixture rate could stabilize learning and in the end increase performance, at a cost in terms of sample efficiency. This would be a problem for a single-threaded agent, like DQN, but it could improve the results of a multi-threaded agent, like R2D2 (Kapturowski et al. 2018). We also recall that default used hyperparameters where better tuned for Atari than for Cartpole, and that this might also influence our empirical results. DCPI could be more efficient by better tuning its own parameters.

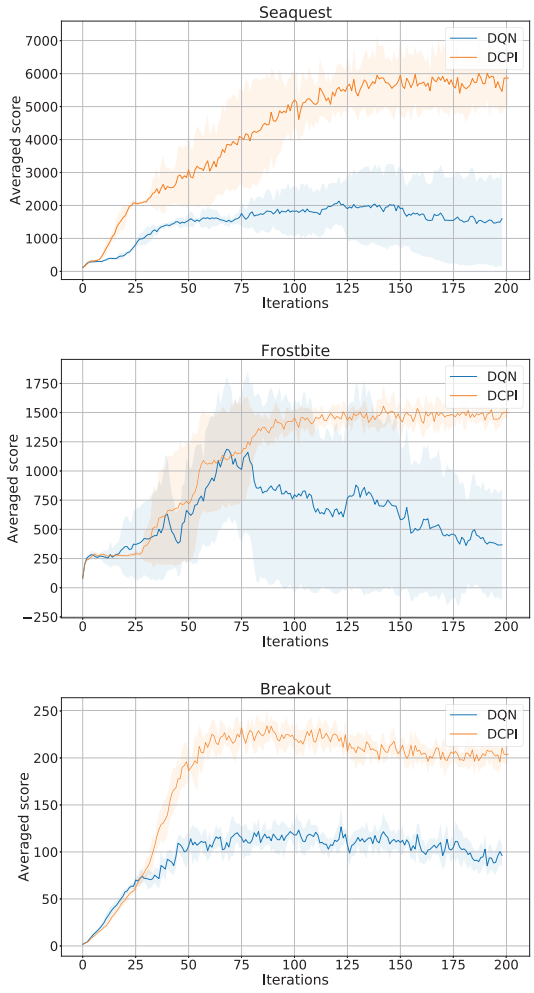

Figure 3: Averaged training scores of DCPI (orange) and DQN (blue) on three of the considered games (Seaquest, Frostbite and Breakout).

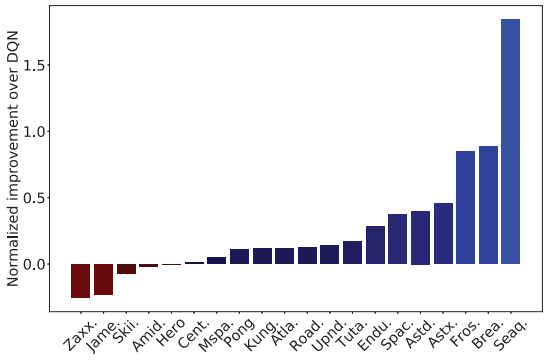

Figure 4: Normalized AUC improvement of DCPI over DQN on a subset of Atari games.

8 Conclusion

We introduced a new deep RL algorithm derived from CPI, DCPI, and this way gave a general method to regularize any pure-critic algorithm by adding a conservative actor to it, based on an approximate stochastic mixture. We gave in Section 3 a detailed depiction of the different approximation steps we used, resulting in the end in a practical algorithm, that we evaluated on several benchmarks. We also proposed different ways to compute adaptive mixture rates for DCPI by approximating optimal rates from the literature. Our experimental results shown, on Cartpole and on most considered Atari games, that DCPI can indeed improve the performance and the stability of learning, often at the cost of slower learning, introducing a speed/stability dilemma. We plan to investigate more adaptive rates, in order to get an even better trade-off and to be less sensitive to the new hyperparameter, and to combine the proposed approach with other variations of DQN, notably based on distributional RL, such as C51 (Bellemare, Dabney, and Munos 2017) or IQN (Dabney et al. 2018).

References

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics* 834–846.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 449–458. JMLR. org.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA.

Bradtke, S. J., and Barto, A. G. 1996. Linear least-squares algorithms for temporal difference learning. *Machine learning* 22(1-3):33–57.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym, 2016. *arXiv preprint arXiv:1606.01540*.

Castro, P. S.; Moitra, S.; Gelada, C.; Kumar, S.; and Bellemare, M. G. 2018. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*.

Dabney, W.; Ostrovski, G.; Silver, D.; and Munos, R. 2018. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, 1096–1105. PMLR.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*, 1861–1870.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep

reinforcement learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

Kakade, S., and Langford, J. 2002. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, volume 2, 267–274.

Kapturowski, S.; Ostrovski, G.; Quan, J.; Munos, R.; and Dabney, W. 2018. Recurrent experience replay in distributed reinforcement learning. In *Proceedings of the International Conference on Learning Representation*.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations*.

Machado, M. C.; Bellemare, M. G.; Talvitie, E.; Veness, J.; Hausknecht, M.; and Bowling, M. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61:523–562.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Ostrovski, G.; Bellemare, M. G.; van den Oord, A.; and Munos, R. 2017. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning*, 2721–2730.

Pirotta, M.; Restelli, M.; Pecorino, A.; and Calandriello, D. 2013. Safe policy iteration. In *Proceedings of the 30th International Conference on Machine Learning*, 307–315.

Puterman, M. L., and Shin, M. C. 1978. Modified policy iteration algorithms for discounted markov decision problems. *Management Science* 24(11):1127–1137.

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

Scherrer, B., and Geist, M. 2014. Local policy search in a convex space and conservative policy iteration as boosted policy search. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 35–50. Springer.

Scherrer, B.; Ghavamzadeh, M.; Gabillon, V.; Lesner, B.; and Geist, M. 2015. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research* 16:1629–1676.

Scherrer, B. 2014. Approximate policy iteration schemes: a comparison. In *Proceedings of the 31st International Conference on Machine Learning*, 1314–1322.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 1889–1897.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3(1):9–44.

Vieillard, N.; Pietquin, O.; and Geist, M. 2019. Deep Conservative Policy Iteration. *arXiv preprint arXiv:1906.09784*.