Exploiting the Sign of the Advantage Function to Learn Deterministic Policies in Continuous Domains

Matthieu Zimmer and **Paul Weng**
Shanghai Jiao Tong University, UM-SJTU Joint Institute, Shanghai, China
{matthieu.zimmer, paul.weng}@sjtu.edu.cn

Abstract

In the context of learning deterministic policies in continuous domains, we revisit an approach, which was first proposed in Continuous Actor Critic Learning Automaton (CACLA) and later extended in Neural Fitted Actor Critic (NFAC). This approach is based on a policy update different from that of deterministic policy gradient (DPG). Previous work has observed its excellent performance empirically, but a theoretical justification is lacking. To fill this gap, we provide a theoretical explanation to motivate this unorthodox policy update by relating it to another update and making explicit the objective function of the latter. We furthermore discuss in depth the properties of these updates to get a deeper understanding of the overall approach. In addition, we extend it and propose a new trust region algorithm, Penalized NFAC (PeNFAC). Finally, we experimentally demonstrate in several classic control problems that it surpasses the state-of-the-art algorithms to learn deterministic policies.

1 Introduction

Model-free reinforcement learning combined with neural networks achieved several recent successes over a large range of domains [Mnih *et al.*, 2015; Lillicrap *et al.*, 2016; Schulman *et al.*, 2017]. Yet those methods are still difficult to apply without any expert knowledge, lack robustness and are very sensitive to hyperparameter optimization [Henderson *et al.*, 2018; Colas *et al.*, 2018].

In this context, we focus in this paper on improving methods that learn deterministic policies. Such policies have three main advantages during the learning phase: 1) they usually require less interactive data because fewer parameters need to be learned, 2) their performances are less costly to estimate during testing phases because randomness only comes from the environment (as opposed to randomized policies), and 3) they are also less sensitive to the premature convergence problem, because they cannot directly control exploration. Moreover, deterministic policies are preferred in some domains (e.g., robotics), because we do not want the agent to act stochastically after the learning phase.

In continuous state and action space domains, solution methods require function approximation. Neural control architectures are excellent representations for policies because they can handle continuous domains, are easily scalable, and have a high degree of expressiveness. The weights of such neural networks are usually updated with a policy gradient method. As vanilla policy gradient suffers from high variance, it is generally implemented in an actor-critic architecture where an estimated value function helps to reduce the variance at the cost of introducing some bias [Konda and Tsitsiklis, 1999]. In this architecture, the parameters (e.g., weights of neural networks) of the policy (i.e., actor) and its value function (i.e., critic) are updated simultaneously.

The basic version of an actor-critic architecture for learning deterministic policies in continuous domains is the deterministic policy gradient (DPG) method [Silver *et al.*, 2014]. Learning the value function is crucial but also difficult, which is why several extensions of DPG have been proposed. Deep Deterministic Policy Gradient (DDPG) [Lillicrap *et al.*, 2016] brings batch normalization [Ioffe and Szegedy, 2015], target networks and replay buffer [Mnih *et al.*, 2015] to DPG and is one of the most used actor-critic methods for learning continuous deterministic policies. However, it has several limitations: 1) the critic learns the state-action value function (Q function), which is difficult to estimate, 2) it relies on the fact that non-biased estimates of the gradient of the Q function are accessible, which is not the case in the model-free setting, 3) it does not use compatible functions: the policy gradient might be poorly estimated.

In this work, we focus on an alternative method that estimates the state value function (V function) instead of the Q function to learn continuous deterministic policies. Van Hasselt and Wiering [2007] were the first to propose to reinforce the policy toward an action with a positive temporal difference. They experimentally showed that using such a method, in an incremental actor-critic algorithm, called Continuous Actor Critic Learning Automaton (CACLA), provided better results than both the stochastic and the deterministic policy gradients[1] in the *Mountain Car* and the *Acrobot* environments. Zimmer *et al.* [2016; 2018] validated those results in higher-dimensional environments, *Half-Cheetah* and *Hu-*

[1]In their paper the deterministic policy gradient algorithm was called ADHDP [Prokhorov and Wunsch, 1997].

manoid in Open Dynamic Engine [Smith, 2005], and proposed several extensions with the Neural Fitted Actor Critic (NFAC) algorithm. However, no theoretical explanation for their good performance, nor a clear discussion about which objective function those methods optimize were given. Providing such an explanation would help understand better why those algorithms work well, what are their properties and limitations, and how to further improve them.

We first show that CACLA and NFAC can be viewed as policy gradient methods and that they are closely related to a specific form of the stochastic policy gradient (SPG) [Sutton *et al.*, 1999]. Then we discuss some of their properties and limitations. Moreover, we extend them with trust region updates and call the new algorithm Penalized Neural Fitted Actor Critic (PeNFAC). Finally, we experimentally show that PeNFAC performs well on three high-dimensional continuous environments compared to the state-of-the-art methods.

2 Background

A continuous Markov Decision Process (MDP) [Sutton, 1988] is a tuple $(\mathcal{S}, \mathcal{A}, T, R, T_0)$ where $\mathcal{S}$ is a continuous state space, $\mathcal{A}$ is a continuous action space with m dimensions, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is a transition function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, T_0 is a distribution over initial states. In the model-free setting, it is assumed that the transition function T and the reward function R are unknown and can only be sampled at specific states according to the interaction between the agent and the environment.

The following notations are used: μ represents a deterministic policy and π a stochastic one. Thus, for a given state $s \in \mathcal{S}$, $\mu(s)$ is an action, $\pi(a|s)$ is the probability of sampling action a from the policy π, and $\pi(\cdot|s)$ is a distribution over the action space $\mathcal{A}$. For a policy π, we denote the discounted state distribution by:

$$d^\pi_\gamma(s) = \int_{\mathcal{S}} T_0(s_0) \sum_{t=0}^{\infty} \gamma^t p(s|s_0, t, \pi) ds_0$$

where $\gamma \in [0,1)$ is a discount factor and $p(s|s_0,t,\pi)$ is the probability of being in state s after applying policy π t timesteps from state s_0. Its state value function is defined by $V^\pi(s) = \mathbb{E}_\pi\big[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\big|S_0 = s\big]$ where $\mathbb{E}_\pi$ is the expectation induced by T and π, and for all t, S_t and A_t are random variables. Its action value function is given by $Q^\pi(s,a) = \mathbb{E}_\pi\big[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\big|S_0 = s, A_0 = a\big]$, and its advantage function by $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

In reinforcement learning, the goal is to find a policy that optimizes the expectation of the discounted rewards:

$$J(\pi) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\big|S_0 \sim T_0\Big].$$

Due to the continuity of the state/action spaces, this optimization problem is usually restricted to a class of parametrized policies, which we denote π_θ (stochastic case) or μ_θ (deterministic case). To simplify notations, we may write π or μ instead of π_θ or μ_θ. The stochastic policy gradient (SPG) in the continuous case can be written as [Sutton *et al.*, 1999]:

$$\nabla_\theta J(\pi) = \int_{\mathcal{S}} d^\pi_\gamma(s) \int_{\mathcal{A}} A^\pi(s,a) \nabla_\theta \pi_\theta(a|s) da ds. \tag{1}$$

The DPG is defined as [Silver *et al.*, 2014]:

$$\nabla_\theta J(\mu) = \int_{\mathcal{S}} d^\mu_\gamma(s) \Delta_{\text{DPG}}(s, \mu_\theta) ds, \tag{2}$$

where

$$\Delta_{\text{DPG}}(s, \mu_\theta) = \nabla_a A^\mu(s,a)\big|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s). \tag{3}$$

Policy gradient methods usually take a step according to those directions: $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J$. However, it is difficult to select a proper learning rate α to control the step size. If α is too big, the method may diverge. If it is too low, the learning will converge slowly (thus requiring more samples). To overcome this difficulty, a trust region method can be used to control the step size [Schulman *et al.*, 2015]. Indeed, one can guarantee monotonic gradient updates by exploiting an approximation of the policy advantage function [Kakade and Langford, 2002] of $\tilde{\pi}$ with respect to π, which measures the difference of performance between the two policies:

$$\begin{aligned} J(\tilde{\pi}) =& J(\pi) + \int_{\mathcal{S}} d^{\tilde{\pi}}_\gamma(s) \int_{\mathcal{A}} \tilde{\pi}(a|s) A^\pi(s,a) da ds, \\ \approx& J(\pi) + \int_{\mathcal{S}} d^{\pi}_\gamma(s) \int_{\mathcal{A}} \tilde{\pi}(a|s) A^\pi(s,a) da ds. \end{aligned} \tag{4}$$

The latter approximation holds when the two policies are close, which can be enforced by a KL divergence constraint in trust region policy optimization [Schulman *et al.*, 2015].

3 Algorithms

In this section, we recall three related algorithms (CACLA, CAC, NFAC) that we discuss later.

3.1 Continuous Actor Critic Learning Automaton

Continuous Actor Critic Learning Automaton (CACLA) [Van Hasselt and Wiering, 2007] is an actor-critic method that learns a stochastic policy π and its estimated value function $\hat{V}^\pi$. We assume in this paper that CACLA uses isotropic Gaussian exploration, which implies that π can be written as follows:

$$\pi_{\theta,\sigma}(\cdot|s) = \mathcal{N}\big(\mu_\theta(s), \sigma^2 I\big) \tag{5}$$

where I is the identity matrix and $\sigma > 0$ possibly annealed during learning. CACLA alternates between two phases:
1) a hill climbing step in the action space using a random optimization (RO) algorithm [Maatyas, 1965],
2) a gradient-like update in the policy parameter space. RO consists in repeating the following two steps:

i) sample a new action a', which is executed in the environment in current state s, by adding a normally distributed noise to the current action $a = \mu(s)$,

ii) if $R(s,a') + \gamma \hat{V}^\pi(s') > \hat{V}^\pi(s)$ then $a \leftarrow a'$ else a does not change.

Phase 2) is based on following update:

$$\text{If } \delta(s,a) > 0 : \tilde{\theta} \leftarrow \theta - \alpha\big(\mu_\theta(s) - a\big) \nabla_\theta \mu_\theta(s), \tag{6}$$

where $\delta(s,a) = R(s,a) + \gamma \hat{V}^\pi(s') - \hat{V}^\pi(s)$ is the temporal difference (TD) error. As the expectation of the TD error is equal to the advantage function, this update can be interpreted as follows: if an exploratory action a has a positive advantage then policy μ should be updated towards a.

Note that although CACLA executes a stochastic policy π, it can be seen as learning a deterministic policy μ. Van Hasselt and Wiering [2007] state that when learning in continuous action space, moving away from a bad action could be meaningless. Indeed, while for stochastic policies, the probability of a bad action can be decreased, for deterministic policies, moving in the action space in the opposite direction of an action with a negative advantage may not necessarily lead to better actions. Thus, CACLA's update is particularly appropriate for learning continuous deterministic policies.

3.2 Continuous Actor Critic

In our discussion, we also refer to a slightly different version of CACLA, Continuous Actor Critic (CAC) [Van Hasselt and Wiering, 2007]. The only difference between CAC and CACLA is that the update in CAC is scaled by the TD error:

$$\text{If } \delta(s,a) > 0 : \tilde{\theta} \leftarrow \theta - \alpha\delta(s,a)\big(\mu_\theta(s) - a\big)\nabla_\theta\mu_\theta(s), \quad (7)$$

Thus an action with a larger positive advantage (here, estimated by the TD error) will have a bigger impact over the global objective.

3.3 Neural Fitted Actor Critic

The Neural Fitted Actor Critic (NFAC) [Zimmer *et al.*, 2016; Zimmer *et al.*, 2018] algorithm is an efficient instantiation of the CACLA update, which integrates the following techniques: batch normalization, λ-returns for both the critic and the actor, and batch learning with Adam[Kingma and Ba, 2015]. In this algorithm, the update of the parameters is not done anymore at each time step, but at the end of a given number of episodes.

4 Discussions

In this section, we discuss the algorithms to provide some theoretical explanation for their good performance.

4.1 CACLA

We first explain the relationship between an algorithm based on stochastic policy gradient (SPG) and CACLA. For this discussion, we assume that SPG is applied to parametrized policies that are Gaussian policies $\pi_{\theta,\sigma}$ (i.e., Gaussian around μ_θ). Then the first common feature between the two algorithms is that the distributions over states they induce during learning are the same (i.e., $d_\gamma^\pi(s)$) because they both use the same exploratory policy to interact with the environment. Moreover, SPG can be written as follows:

$$\begin{aligned}&\nabla_\theta J(\pi_{\theta,\sigma})\\ &= \int_{\mathcal{S}} d_\gamma^\pi(s) \int_{\mathcal{A}} \pi_{\theta,\sigma}(a|s) A^\pi(s,a) \nabla_\theta \log \pi_\theta(a|s) da ds,\\ &= \frac{1}{\sigma^2}\int_{\mathcal{S}} d_\gamma^\pi(s) \int_{\mathcal{A}} \pi_{\theta,\sigma}(a|s) A^\pi(s,a)\big(a - \mu_\theta(s)\big)\cdot\\ &\qquad\qquad \nabla_\theta\mu_\theta(s) da ds.\end{aligned}$$

For CACLA, we interpret update (6) as a stochastic update in the following direction:

$$\int_{\mathcal{S}} d_\gamma^\pi(s)\Delta_{\text{CACLA}}(s,\mu_\theta)ds, \quad (8)$$

$$\text{with } \Delta_{\text{CACLA}}(s,\mu_\theta) = \int_{\mathcal{A}} \pi_{\theta,\sigma}(a|s) H\big(A^\pi(s,a)\big)\times \big(\mu_\theta(s) - a\big)\nabla_\theta\mu_\theta(s) da,$$

where H is the Heaviside function. Indeed, the inner integral is estimated using a single Monte Carlo sample during the run of CACLA.

Under this form, it is easy to see the similarity between SPG and CACLA. The constant factor $\frac{1}{\sigma^2}$ can be neglected because it may be integrated into the learning rate. The sign difference of the term $(a - \mu_\theta(s))$ is because SPG performs gradient ascent and CACLA gradient descent. So the main difference between SPG and CACLA is the replacement of $A^\pi(s,a)$ by $H(A^\pi(s,a))$. Therefore CACLA optimizes its exploratory stochastic policy through an approximation of SPG hoping to improve the underlying deterministic policy (for a fixed state, the direction of CACLA and SPG are the same up to a scalar).

Moreover, relating CACLA's update with (8) also brings to light two main limitations. The first one concerns the inner integral over the action space which has a high variance. Therefore, we expect CACLA to be less and less data efficient in high-dimension action space (which is the main theoretical justification of DPG over SPG - see Appendix B.1). The second limitation that appears is that over one update, CACLA does not share the same exact optimal solutions as DPG or SPG. Indeed, if we define θ^* such as $\nabla_\theta J(\mu_\theta)\big|_{\theta=\theta^*} = 0$ it is not possible to prove that (8) will also be 0 (because of the integral over the state space). It means that CACLA could decrease the performance of this local optimal solution.

4.2 CAC

Similarly, the update in CAC can be seen as a stochastic update in the following direction:

$$\int_{\mathcal{S}} d_\gamma^\pi(s)\Delta_{\text{CAC}}(s,\mu_\theta)ds,$$

$$\text{with } \Delta_{\text{CAC}}(s,\mu_\theta) = \int_{\mathcal{A}} \pi_{\theta,\sigma}(a|s) A^\pi(s,a) H\big(A^\pi(s,a)\big)\times \big(\mu_\theta(s) - a\big)\nabla_\theta\mu_\theta(s) da.$$

This shows that CAC is even closer to SPG than CACLA and provides a good theoretical justification of this update at a local level (not moving in potentially worst action). However, there is also a justification at a more global level.

Lemma 4.1. *For a fixed state, when the exploration tends to zero, CAC maintains the sign of the DPG update with a scaled magnitude:*

$$\lim_{\sigma\to 0} \Delta_{CAC}(s,\mu_\theta) \leftarrow g^+(s,\pi) \circ \Delta_{DPG}(s,\mu_\theta), \quad (9)$$

where $g^+(s,\pi)$ is a positive function between $[0;1]^n$ with n as the number of parameters of the deterministic policy and $\circ$ is the Hadamard product (element-wise product).

The proof is provided in Appendix A.1. The consequence of this lemma is that, for a given state and low exploration, a local optimal solution for DPG will also be one for CAC. However it is still not the case for the overall update because of the integral over the different states. The weights given to each direction over different states are not the same in CAC and DPG. One might think that in such a case, it would be better to use DPG. However, in practice, the CAC update may in fact be more accurate when using an approximate advantage function. Indeed, there exist cases where DPG with an approximate critic might update towards a direction which could decrease the performance. For instance, when the estimated advantage $\hat{A}\big(s, \mu(s)\big)$ is negative, the advantage around $\mu(s)$ is therefore known to be poorly estimated. In such a case, thanks to the Heaviside function, CAC will not perform any update for actions a in the neighborhood of $\mu(s)$ such that $\hat{A}(s,a) \leq 0$. However, in such a case, DPG will still perform an update according to this poorly estimated gradient.

5 Extension to Trust Region

In this section, we extend the approach to use a trust region method.

5.1 Trust Region for Deterministic Policies

We now introduce a trust region method dedicated to continuous deterministic policies. Given current deterministic policy μ, and an exploratory policy π defined from μ, the question is to find a new deterministic policy $\tilde{\mu}$ that improves upon μ. Because a deterministic policy is usually never played in the environment outside of testing phases, a direct measure between two deterministic policies (i.e., a deterministic equivalent of Equation 4) is not directly exploitable. Instead we introduce the following measure:

Lemma 5.1. *The performance $J(\tilde{\mu})$ of a deterministic policy $\tilde{\mu}$ can be expressed by the advantage function of another stochastic policy π built upon a deterministic policy μ as:*

$$J(\tilde{\mu}) = J(\mu) + \int_{\mathcal{S}} d^{\pi}_{\gamma}(s) \int_{\mathcal{A}} \pi(a|s) A^{\mu}(s,a) da ds + \int_{\mathcal{S}} d^{\tilde{\mu}}_{\gamma}(s) A^{\pi}\big(s, \tilde{\mu}(s)\big) ds. \quad (10)$$

See Appendix A.2 for the proof. The first two quantities in the RHS of (10) are independent of $\tilde{\mu}$. The second one represents the difference of performance from moving from the deterministic policy μ to its stochastic version π. Because $d^{\tilde{\mu}}_{\gamma}$ would be too costly to estimate, we approximate it with the simpler quantity d^{π}_{γ}, as done by Schulman *et al.* [2015] for TRPO, a predecessor to PPO.

Theorem 5.2. *Given two deterministic policies μ and $\tilde{\mu}$, a stochastic Gaussian policy π with mean $\mu(s)$ in state s and independent variance σ, if the transition function T is L-Lipschitz continuous with respect to the action from any state then:*

$$\Big| \int_{\mathcal{S}} d^{\tilde{\mu}}(s) A^{\pi}\big(s, \tilde{\mu}(s)\big) - \int_{\mathcal{S}} d^{\pi}(s) A^{\pi}\big(s, \tilde{\mu}(s)\big) \Big| \leq \frac{\epsilon L}{1-\gamma} \max_{t>0} \Big(\big|\big|\tilde{\mu}(s) - \mu(s)\big|\big|_{2,\infty} + \frac{2m\sigma}{\sqrt{2\pi}} \Big)^t,$$

where $\epsilon = max_{s,a}|A^{\pi}(s,a)|$.

The proof is available in Appendix A.3. Thus, to ensure a stable improvement at each update, we need to keep both $||\mu - \tilde{\mu}||_{2,\infty}$ and σ small. Note that the Lipschitz continuity condition is natural in continuous action spaces. It simply states that for a given state, actions that are close will produce similar transitions.

5.2 Practical Algorithm

To obtain a concrete and efficient algorithm, the trust region method can be combined with the previous algorithms. Its integration to NFAC with a CAC update for the actor is called Penalized Neural Fitted Actor Critic (PeNFAC).

Van Hasselt and Wiering [2007] observed that the CAC update performs worse that the CACLA update in their algorithms. In their setting where the policy and the critic are updated at each timestep, we believe this observation is explained by the use of the TD error (computed from a single sample) to estimate the advantage function. However, when using variance reduction techniques such as λ-returns and learning from a batch of interactions, or when mitigating the update with a trust region constraint, we observe that this estimation becomes better (see Figure 4). This explains why we choose a CAC update in PeNFAC.

In order to ensure that $||\mu - \tilde{\mu}||_{2,\infty}$ stays small over the whole state space, we approximate it with a Euclidean norm over the state visited by π. To implement this constraint, we add a regularization term to the update and automatically adapts its coefficient, for a trajectory $(s_0, s_1, \dots, s_h)$:

$$\sum_{t=0}^{h-1} \Delta_{\text{CAC}}(s_t, \mu_\theta) + \beta \nabla_\theta \big|\big|\mu_{\text{old}}(s_t) - \mu_\theta(s_t)\big|\big|_2^2,$$

where β is a regularization coefficient. Similarly to the adaptive version of Proximal Policy Optimization (PPO) [Schulman *et al.*, 2017], β is updated in the following way (starting from $\beta \leftarrow 1$):

- if $\hat{d}(\mu, \mu_{\text{old}}) < d_{\text{target}}/1.5$: $\beta \leftarrow \beta/2$,
- if $\hat{d}(\mu, \mu_{\text{old}}) > d_{\text{target}} \times 1.5$: $\beta \leftarrow \beta \times 2$,

where $\hat{d}(\mu, \mu_{\text{old}}) = \frac{1}{\sqrt{mL}} \sum_{s \sim \pi} ||\mu_{\text{old}}(s) - \mu_\theta(s)||_2$ with L being the number of gathered states. Those hyper-parameters are usually not optimized because the learning is not too sensitive to them. The essential value to adapt for the designer is d_{target}. Note that the introduction of this hyperparameter mitigates the need to optimize the learning rate for the update of the policy, which is generally a much harder task.

6 Experiments

We performed two sets of experiments to answer the following questions:

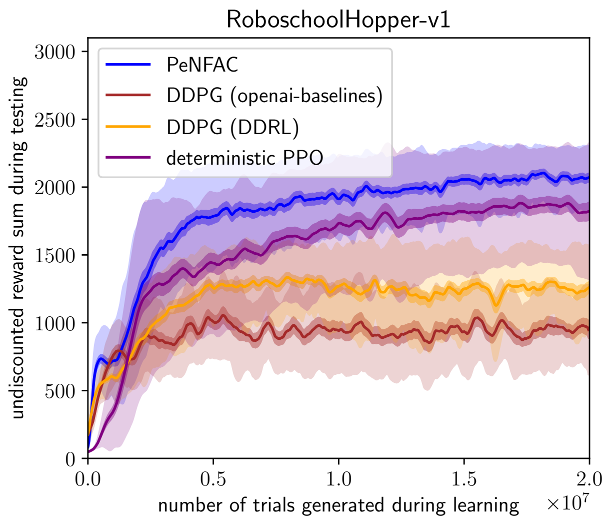

Figure 1: Comparison of PeNFAC, DDPG and deterministic PPO over 60 different seeds for each algorithm in Hopper.

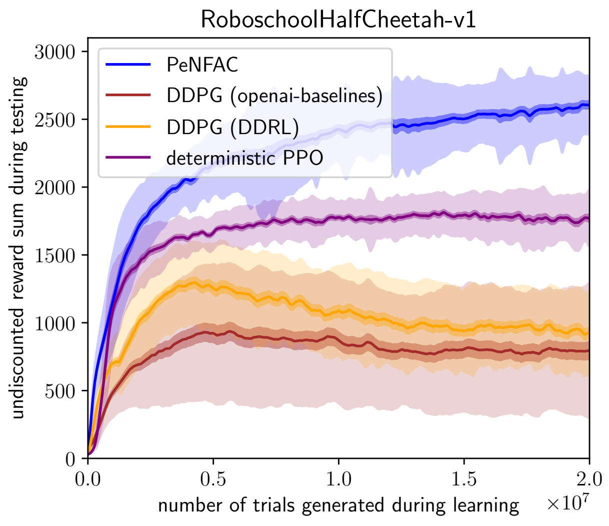

Figure 2: Comparison of PeNFAC, DDPG and deterministic PPO over 60 different seeds for each algorithm in HalfCheetah.

1) How does PeNFAC compare with state-of-the-art algorithms for learning deterministic policies?

2) Which components of PeNFAC contribute the most to its performance?

The experiments were performed on environments with continuous state and action spaces in a time-discretized simulation. We chose to perform the experiments on OpenAI Roboschool [Schulman *et al.*, 2017], a free open-source software, which allows anyone to easily reproduce our experiments. In order to evaluate the performance of an algorithm, deterministic policies μ obtained during learning are evaluated at a constant interval during testing phases: policy μ is played in the environment without exploration. The interactions gathered during this evaluation are not available to any algorithms. The source code of the PeNFAC algorithm is available at github.com/matthieu637/ddrl. The hyperparameters used are reported in Appendix E as well as the considered range during the grid search.

6.1 Performance of PeNFAC

We compared the performance of PeNFAC to learn continuous deterministic policies with two state-of-the-art algorithms: PPO and DDPG. A comparison with NFAC is available in the ablation study (Section 6.2) and in Appendix C. Because PPO learns a stochastic policy, for the testing phases, we built a deterministic policy as follows $\mu(s) = \mathbb{E}[a|a \sim \pi_\theta(\cdot, s)]$. We denote this algorithm as "deterministic PPO". In Appendix D, we experimentally show that this does not penalize the comparison with PPO, as deterministic PPO provides better results than standard PPO. For PPO, we used the OpenAI Baseline implementation. To implement PeNFAC and compare it with NFAC, we use the DDRL library [Zimmer *et al.*, 2018]. Given that DDPG is present in those two libraries, we provided the two performances for it. The OpenAI Baseline version uses an exploration in the parameter space and the DDRL version uses n-step returns.

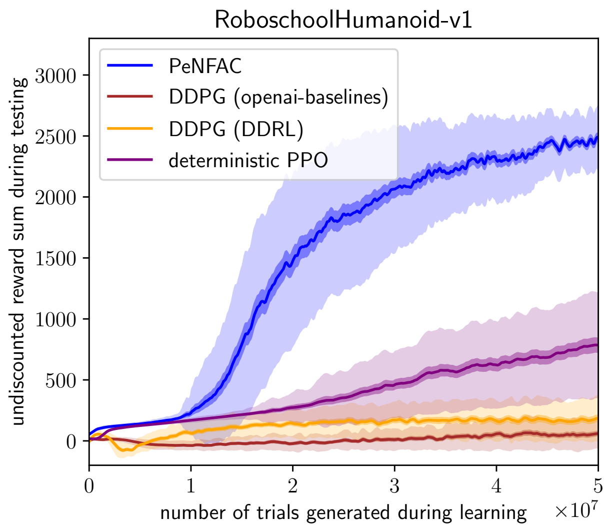

Figure 3: Comparison of PeNFAC, DDPG and deterministic PPO over 60 seeds for each algorithm in Humanoid.

We performed learning experiments over three high-dimensional domains: Hopper, HalfCheetah and Humanoid. Dimensions of $\mathcal{S} \times \mathcal{A}$ are 15×3 (Hopper), 26×6 (HalfCheetah) and 44×17 (Humanoid).

The neural network architecture is composed of two hidden layers of 64 units for either the policy or the value function. The choice of the activation function in the hidden units was optimized for each algorithm: we found that ReLU was better for all of them except for PPO (where tanh was better). The output activation of the critic is linear and the output activation of the actor is tanh.

In Figures 1-4, the lighter shade depicts one standard deviation around the average, while the darker shade is the standard deviation divided by the square root of the number of seeds.

In Figures 1-3, PeNFAC outperforms DDPG and deterministic PPO during the testing phase. On Humanoid, even after optimizing the hyperparameters, we could not obtain the same results as those of Schulman *et al.* [2017]. We conjecture that this may be explained as follows: 1) the Roboschool-Humanoid moved from version 0 to 1, 2) deterministic PPO might be less efficient than PPO, 3) neither LinearAnneal for the exploration, nor adaptive Adam step size is present in the OpenAI Baseline implementation. However, we argue that the comparison should still be fair since PeNFAC also does not use those two components. On Humanoid, we did not find a set of hyperparameters where DDPG could work correctly with both implementations.

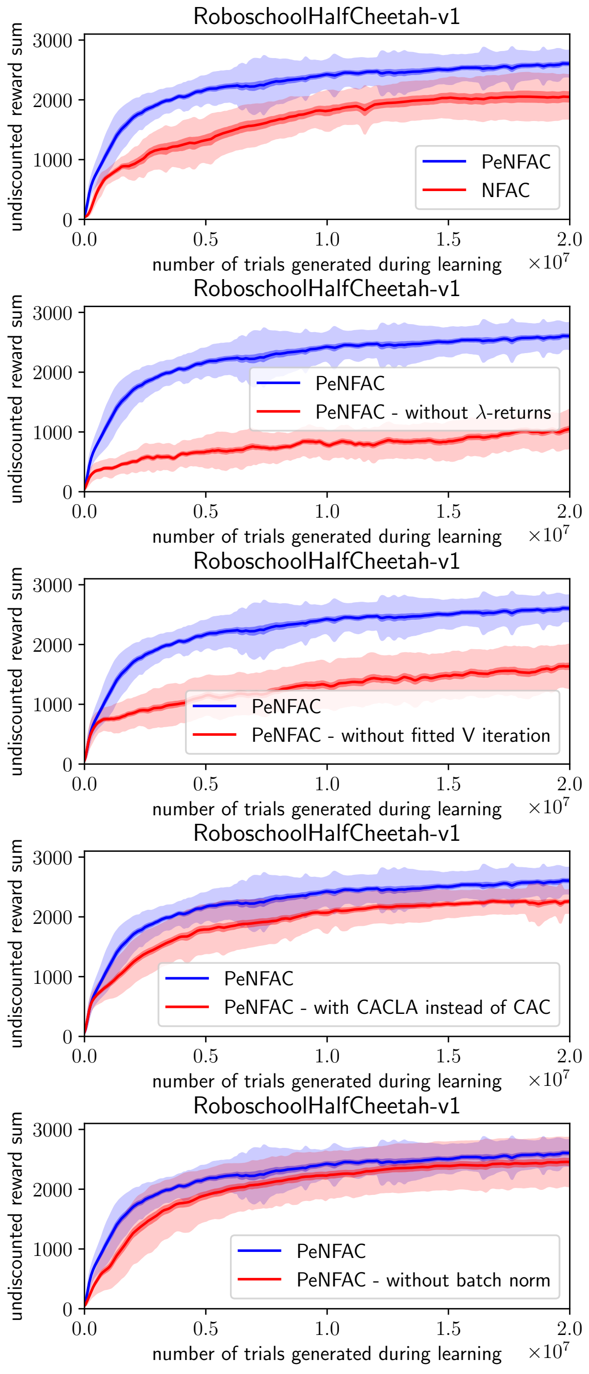

Figure 4: Comparison of the different components (λ-returns, fitted value-iteration, CAC vs CACLA update, batch normalization) of the PeNFAC algorithm during the testing phase over the HalfCheetah environment and 60 seeds for each version.

6.2 Components of PeNFAC

In Figure 4, we present an ablation analysis in the HalfCheetah domain to understand which components of the PenFAC algorithm are the most essential to its good performance. From top to bottom plots of Figure 4, we ran PenFAC with or without trust region, with or without λ-returns, with or without fitted value iteration, with CACLA update or CAC update, and finally with or without batch normalization.

It appears that λ-returns and fitted value iteration are the most needed, while the effect of batch normalization is small and mostly helps in the beginning of the learning.

We also tried updating the actor every timestep without taking into account the sign of the advantage function (i.e., using SPG instead of CAC), but the algorithm was not able to learn at all. This also demonstrates that the CAC update is an essential component of PenFAC.

7 Conclusion

In the context of learning deterministic policies, we studied the properties of two not very well-known but efficient updates, Continuous Actor Critic Learning Automaton (CACLA) and Continuous Actor Critic (CAC). We first showed how closely they both are related to the stochastic policy gradient (SPG). We explained why they are well designed to learn continuous deterministic policies when the value function is only approximated. We also highlighted the limitations of those methods: a potential poor sample efficiency when the dimension of the action space increases and no guarantee that the underlying deterministic policy will converge toward a local optimum of $J(\mu_\theta)$ even with a linear approximation.

In the second part, we extended Neural Fitted Actor Critic (NFAC), itself an extension of CACLA, with a trust region constraint designed for deterministic policies and proposed a new algorithm, Penalized NFAC (PeNFAC). Finally, we tried our implementation on various high-dimensional continuous environments and showed that PeNFAC performs better than DDPG and PPO to learn continuous deterministic policies.

As future work, we plan to consider off-policy learning and the combination of the updates of CAC and DPG together to ensure the convergence toward a local optimum while benefiting from the good updates of CAC.

Acknowledgments

This work has been supported in part by the program of National Natural Science Foundation of China (No. 61872238). Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[Colas *et al.*, 2018] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms. In *International Conference on Machine Learning*, Stockholm, Sweden, July 2018.

[Henderson *et al.*, 2018] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015.

[Kakade and Langford, 2002] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, volume 2, pages 267–274, 2002.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy L. Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations*, pages 1–13, 2015.

[Konda and Tsitsiklis, 1999] Vijay R. Konda and John N. Tsitsiklis. Actor-Critic Algorithms. *Neural Information Processing Systems*, 13:1008–1014, 1999.

[Lillicrap *et al.*, 2016] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

[Maatyas, 1965] Josef Maatyas. Random optimization. *Automation and Remote control*, 26(2):246–253, 1965.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Prokhorov and Wunsch, 1997] Danil V. Prokhorov and Donald C. Wunsch. Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8(5):997–1007, 1997.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Michael Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *International Conference on Machine Learning*, page 16, 2015.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395, 2014.

[Smith, 2005] Russell Smith. Open dynamics engine. 2005.

[Sutton *et al.*, 1999] Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In Advances in Neural Information Processing Systems 12*, pages 1057–1063, 1999.

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

[Van Hasselt and Wiering, 2007] Hado Van Hasselt and Marco A. Wiering. Reinforcement learning in continuous action spaces. In *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, 2007.

[Zimmer *et al.*, 2016] Matthieu Zimmer, Yann Boniface, and Alain Dutech. Neural Fitted Actor-Critic. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2016.

[Zimmer *et al.*, 2018] Matthieu Zimmer, Yann Boniface, and Alain Dutech. Developmental reinforcement learning through sensorimotor space enlargement. In *The 8th Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics*, September 2018.