Off-Policy Neural Fitted Actor-Critic
Matthieu Zimmer, Yann Boniface, Alain Dutech

To cite this version:

HAL Id: hal-01413886
https://hal.archives-ouvertes.fr/hal-01413886
Submitted on 11 Dec 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A new off-policy, offline, model-free, actor-critic reinforcement learning algorithm dealing with continuous environments in both states and actions is presented. It addresses discrete time problems where the goal is to maximize the discounted sum of rewards using stationary policies. Our algorithm allows to trade-off between data-efficiency and scalability. The amount of a priori knowledge is kept low by: (1) using neural networks to learn both the critic and the actor, (2) not relying on initial trajectories provided by an expert, and (3) not depending on known goal states. Experimental results compare data-efficiency to 4 state-of-the-art algorithms on three benchmark environments.

This article largely reproduces a previous work [34] by adding a higher dimensional environment, improving control architectures and provides batch normalization for others state-of-the-art algorithms.
In this work, we present an offline, model-free, off-policy, actor-critic RL algorithm that allows a trade-off between scalability and data-efficiency. It is based on the fitted actor-critic family and benefits from the improvements proposed by Deep Deterministic Policy Gradient (DDPG).

## 2 Background

We are interested in RL problems, modeled as Markov Decision Processes (MDP) \( \langle S, A, T, R \rangle \), where the state space \( S \) and the action space \( A \) are continuous. The goal is to seek for an optimal policy \( \pi^* \) maximizing the expected discounted reward:

\[
\pi^* = \arg \max_\pi J(\pi) = \arg \max_\pi \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t \times R(s_t, \pi_t(s_t)) \right],
\]

where \( t \) denotes a time step and \( 0 < \gamma < 1 \) is the discount factor.

When the state space \( S \) is continuous, classical value-function methods like Least-Squares Temporal Difference (LSTD) rely on an estimation of \( Q : S \times A \rightarrow \mathbb{R} \), the sequential values of actions in each state:

\[
Q^\pi(s, a) = \mathbb{E}_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right],
\]

where \( r_t \) is the reward obtained at time \( t \) from \( R \) following \( \pi \). Being data-efficient means to search for the best policy given the collected samples. An example of a neural data-efficient, critic-only algorithm is Fitted Q Iteration (FQI), which updates the Q function several times using the Bellman operator as an approximated version of Value Iteration. Instead of iterating over all states and actions, it relies only on the collected samples \((s_t, a_t, r_{t+1}, s_{t+1})\):

\[
Q_{k+1} = \arg \min_{Q \in \mathcal{F}} \sum_{i=1}^{N} \left[ Q(s_t, a_t) - \left( r_{t+1} + \gamma \max_{a' \in A} Q_k(s_{t+1}, a') \right) \right]^2.
\]

When the action space \( A \) is continuous, the use of an actor (i.e., a parametric policy) becomes crucial to overcome the complexity of the argmax search. This often leads to actor-only methods like Policy Gradient, Covariance Matrix Adaptation Evolution Strategy (CMA-ES) or Trust Region Policy Optimization (TRPO). The major drawbacks of actor-only methods are the high variability of the cost \( J \) (because there is no critic) and, for gradient-based methods, the plateau effect and local minima that can lead to poor policies. On the other hand, actor-critic algorithms try to combine both the advantages of previous methods. The critic learns a value function thus reducing the variability of the approximation of \( J \) and the actor learns the parametric policy, allowing the use of continuous actions.

We now present three state-of-the-art actor-critic algorithms that we will use for comparison in our experiments (from least to most data-efficient):

- Continuous Actor Critic Learning Automaton (CACLA) is a successful actor-critic algorithm that uses neural networks for both the critic and the actor. Due to its online nature and its on-policy updates, it cannot achieve good data efficiency (the collected data is used then forgotten). In some environments, CACLA performs better than CMA-ES.
- Neural Fitted Actor Critic (NFAC) may achieve a better data efficiency than CACLA since it uses FQI updates. However, the data is forgotten after each end of episode because the actor features on-policy update.
- Deep Deterministic Policy Gradient (DDPG) is also an actor-critic algorithm. It accomplishes online updates of the policy and Q function, and it reuse previous samples through its off-policy update. Based on Neural Fitted Q with Continuous Actions, DDPG is more scalable due to online updates, targets networks and batch normalization. The target networks serve to slow down the weights updates to increase the stability of learning, by soft updating a copy of the policy and the value function.

Recently, two new methods have been proposed to increase the efficiency of some RL algorithms.
• When the dimensions of action space $A$ are bounded, instead of limiting the output of the neural policy with a last layer (for instance with a hyperbolic tangent) that squashes the gradient obtained from the critic, it is preferable to have an unbounded last layer with an adapted gradient strategy [11].

• Retrace($\lambda$) is a new strategy to weight a sample for off-policy learning [20], it provides low-variance, safe and efficient updates.

3 Algorithm

Our algorithm, that we name Data Efficient Neural Fitted Actor Critic (DENFAC), can be seen as a neural version of a fitted actor-critic (FAC) algorithm [1]. It contains an approximated version of both Value and Policy Iteration (for the critic and the actor respectively).

The critic is updated with a FQI update where the argmax operator is replaced by the policy choice.

$$Q_{k+1} = \arg\min_{Q \in F} \sum_{(s_t, a_t, r_{t+1}, s_{t+1}) \in D} c(s_t, a_t) \left[ Q(s_t, a_t) - (r_{t+1} + \gamma Q_k(s_{t+1}, \pi_k(s_{t+1})) \right]^2,$$

(4)

$$\pi_{k+1} = \arg\max_{\pi \in F_a} \sum_{s_t \in D} Q_{k+1}(s_t, \pi_k(s_t)),$$

(5)

where $c(s_t, a_t) = \min\left(1, \frac{\pi_{k-1}(a_t|s_t)}{\pi_k(a_t|s_t)}\right)$ is the weight associated to a sample [20], and $\pi_0$ is the policy that gathered the sample. This coupled optimization can be applied multiple times without acquiring new samples.

DENFAC is an off-line algorithm, therefore the execution part of one episode consists only of performing the policy choices and collecting the samples $(s_t, a_t, r_{t+1}, s_{t+1})$ that are added to $D$ (the replay buffer). The off-line part is depicted in Algorithm 1. The algorithm is data-efficient because it performs a type of FQI. Furthermore, unlike DDPG, it performs updates over the largest set of data given a computational constraint. This might requires too much computational time so the data-efficiency vs scalability dilemma can be adjusted through the length of $D$. Another meta-parameter of Algorithm 1, reset_critic that reset the weight of the critic to another initial solution, can lead to an even better data-efficiency by allowing the critic to get out of local minima.

A challenging problem in this algorithm is how to handle the growth of $D$. In this work, we consider $D$ as a First-In First-Out (FIFO) queue. So the agent accesses to its memory of the latest episodes. It can be enough in some simple environments but it is clearly sub-optimal in a data-efficiency point of view. Defining a weight associated to each data of $D$ from the $\delta$-error might be a solution [23]. However, in contrast to [23] the $\delta$-error depends not only on the critic but also of the actor. It is not clear yet if the sampling from $D$ should be the same for the actor and the critic.

DENFAC learns a deterministic policy, thus during the execution part an exploration strategy must be used. It can greatly influence the data-efficiency. We do not address this issue in this work. In the experimental setup, each algorithm uses the same exploration strategy: a Gaussian noise is added to the actor choice.

<table>
<thead>
<tr>
<th>FAC</th>
<th>DDPG</th>
<th>NFAC</th>
<th>DENFAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline &amp; Batch</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Off-policy</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Fitted Critic</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Actor updated through $\nabla Q$</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Learn Q</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Reset Networks</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Retrace</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Target Networks</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Batch Normalization</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Figure 1: Properties of the nearest actor-critic algorithms: FAC [1], DDPG [18] and NFAC [33].
Data: $D$ replay buffer of $N$ samples, $Q_0$ value-function, $\pi_b$ previous policies, $K$ number of fitted iteration, $G$ number of gradient descent for actor updates, inverting gradient strategy, reset critic strategy

Result: $\pi_K$ the next policy to play, $Q_K$ the next value function

for $k \leftarrow 1$ to $K$ do
  for $(s_t, a_t, u_t, r_{t+1}, s_{t+1}) \in D$ do
    $q_{k,t} \leftarrow \begin{cases} r_{t+1}, & \text{if } s_{t+1} \in S^* \\ r_{t+1} + \gamma Q_{k-1}(s_{t+1}, \pi_{k-1}(s_{t+1})), & \text{otherwise} \end{cases}$
  end
  $Q_k \leftarrow$ randomly initialize critic network if reset critic else $Q_{k-1}$

Update critic by minimizing the loss:

$$L = \frac{1}{N} \sum_{t=1}^{N} \min \left( 1, \frac{\pi_{k-1}(a_t|s_t)}{\pi_b(a_t|s_t)} \right) \left( q_{k,t} - Q_k(s_t, a_t) \right)^2$$

Randomly initialize actor network $\pi_k$

for $g \leftarrow 1$ to $G$ do
  Update the actor policy using the batch gradient:
  if inverting gradient then
    $$\nabla_a = \nabla_a \begin{cases} (a_{\text{max}} - a)/(a_{\text{max}} - a_{\text{min}}), & \text{if } \nabla_a < 0 \\ (a - a_{\text{min}})/(a_{\text{max}} - a_{\text{min}}), & \text{otherwise} \end{cases}$$
  end
  $$\nabla_\theta \pi_k = \frac{1}{N} \sum_{t=1}^{N} \nabla_a Q(s_t, a)|_{a=\pi_k(s_t)} \nabla_\theta \pi_k(s_t)$$
end

Algorithm 1: Data Efficient Neural Fitted Actor Critic (DENFAC)

4 Experimental Setup

An experimental comparison of DENFAC, DDPG, CMA-ES, NFAC and CACLA is done into three environments: Acrobat [26], Cartpole [22] and Half-Cheetah [32].

In Acrobat (double swing-up), the reward function is defined as (1) +1 if the goal is reached (arm straight up), (2) the normalized max height of end effector if 500 steps are reached, and (3) 0 otherwise.

In Cartpole (inverted pendulum), the reward function is defined as (1) 0 when the cart position is between $[-0.05; 0.05]$ and the pole angle between $[-\frac{\pi}{6}; \frac{\pi}{6}]$, (2) $-2 \times (500 - \text{last_step})$ if it exits at last_step (pole angle $\not\in [-\frac{\pi}{2}, \frac{\pi}{2}]$) or cart position $\not\in [-2.4; 2.4]$), and (3) -1 otherwise.

In Half-Cheetah, the reward function is $R(s, a) = v_x(s) - 0.05 \cdot ||a||^2_2 - 1 \cdot g(s)$ where $v_x(s)$ is the speed of the cheetah on x axis and $g(s)$ is 1 if the heel, the or 0 otherwise.

The discount factor is fixed to $\gamma = 0.9$ (Acrobat) and $\gamma = 0.99$ (Cartpole and Half-Cheetah). States are composed of the joint positions/angles and joint position/angle velocities. Dimensions of $S \times A$ are $4 \times 1$ (Acrobat), $4 \times 1$ (Cartpole) and $20 \times 6$ (Half-Cheetah).

The neural networks use (1) Adam learning algorithm [16], (2) the leaky rectified linearity (ReLU) [6], and (3) batch normalization [13]. Critic networks contain 2 hidden layers of 50 and 7 neurons. The structure of the actor networks is fixed to: $1 \times 5 \times 1$ units (Acrobat), $1 \times 20 \times 1$ units (Cartpole), and $20 \times 20 \times 10 \times 6$ units (Half-Cheetah). The last layer of the critic and actor networks is linear. The actor policy is a truncated Gaussian policy between $[a_{\text{min}} = -1, a_{\text{max}} = 1]$ with $\sigma = 0.05$ where the mean is determined by the output of the last linear layer. Each weight is initialized from a normal distribution $N(0, 0.01)$. Batch normalization is applied on each layer for both the actor and the critic, except on the last two layers.
For each experimental setup, we first optimize all the meta-parameters of DDPG and then apply them to DENFAC. To obtain a fair comparison, we also optimized the number of updates performed by DDPG, and we applied the inverting gradient strategy (when it was better) to make it more data-efficient. NFAC and CACLA algorithms are improved with batch normalization, denoted as NFAC+ and CACLA+ in Figure 3. Only CMA-ES do not use batch normalization for its policy as it does not rely on the gradient.

Figure 3 shows that DENFAC quickly develops good policies on each task compared to DDPG (even if DDPG is online). However, unexpectedly DENFAC, which is off-policy, has the same order of performance that NFAC+, which is on-policy. It lets us think that either $D$ should not be a simple FIFO queue, or that the policy update of NFAC+ allows a better exploration.

Surprisingly, the actor-only method CMA-ES achieves very good data-efficiency on those environments. In higher dimensional environments, it can lack scalability [4] but here it outperforms online algorithms like CACLA or DDPG even if they have access to the reward information faster (online algorithms).
Figure 4: Median and quartiles of the best registered performance Half-Cheetah (the higher, the better) environments during RL learning with each algorithm.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Algorithm</th>
<th>$\alpha_a$</th>
<th>$\alpha_c$</th>
<th>inverting gradient</th>
<th>mini batch size</th>
<th>$\tau$</th>
<th>additional updates</th>
<th>$K$</th>
<th>$G$</th>
<th>batch size $D$</th>
<th>reset critic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobot</td>
<td>DDPG</td>
<td>0.1</td>
<td>No</td>
<td>64</td>
<td>0.001</td>
<td>8</td>
<td></td>
<td></td>
<td>25</td>
<td>6000</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>DENFAC</td>
<td>0.1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25</td>
<td>3000</td>
<td>Yes</td>
</tr>
<tr>
<td>Cartpole</td>
<td>DDPG</td>
<td>0.1</td>
<td>Yes</td>
<td>64</td>
<td>0.1</td>
<td>8</td>
<td></td>
<td></td>
<td>10</td>
<td>3000</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>DENFAC</td>
<td>0.1</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>3000</td>
<td>No</td>
</tr>
<tr>
<td>Half-Cheetah</td>
<td>DDPG</td>
<td>0.1</td>
<td>Yes</td>
<td>64</td>
<td>0.001</td>
<td>8</td>
<td></td>
<td></td>
<td>10</td>
<td>3000</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>DENFAC</td>
<td>0.1</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>3000</td>
<td>No</td>
</tr>
</tbody>
</table>

Figure 5: Best meta-parameters found for DDPG and DENFAC.

CACLA or CACLA+ cannot reach the goal in only 750 episodes on Acrobot. On those environments, it’s the less data-efficient algorithm but also the faster in computational time. It proves that having a replay buffer like NFAC, DDPG and DENFAC helps to improve data-efficient.

We did not notice that adding a L2 regularization term in the critic, as done in the original version of DDPG, improves it in those environments. In some experiments, we also run our algorithm in an online setting or with target networks, but this did not improve the data-efficiency, while requiring more computations (results not shown here).

5 Conclusions and further work

We investigated the data-efficiency vs scalability dilemma in three fully continuous environments. Data-efficiency often implies more computational time spent on each data impeding the scalability. In some cases, resetting the weights of the neural networks shows even more data-efficiency. All those additional costs must be negligible compared to the cost of producing data in the environment otherwise such methods are not appropriate. DENFAC is more data-efficient than 4 state-of-the-art actor-critic algorithms in some environments but comes at a higher computational cost. To further improve DENFAC, it should be analyzed how to replace the FIFO queue for $D$ and if uniform sampling could be improved [23]. Moreover, DENFAC lacks stability in learning, target networks did not help, slowing down the change in the policy might increase his stability [24].
Acknowledgments

The data has been numerically analyzed with the free software package GNU Octave [15]. We used Caffe as neural network library [14] and Open Dynamic Engine (ODE) as physic engine [25]. 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


