Reinforcement learning (RL) is a framework for solving sequential decision problems where an agent interacts with its environment and adapts its policy based on a reward signal. We present an RL algorithm respecting two main requirements:

1. dealing with continuous action and state spaces
2. knowledge added by the designer to the agent should be minimal

**Algorithm**

Randomly initialize critic network \( V \)
Randomly initialize actor network \( \pi \)

for episode \( 0 \) to \( M \) do

Receive initial state \( s_0 \) \( t \leftarrow 0 \)

while \( s_t \notin S^* \) and \( t < T_{\text{max}} \) do

Select action \( a_t \leftarrow \pi(s) \)
Add exploration noise \( a' \leftarrow g(u_t, N) \)
Perform action \( a_t, a' \), observe \( r_{t+1} \) and \( s_{t+1} \)
Store transition \( (s_t, a_t, r_{t+1}, s_{t+1}) \) in \( D_\pi \)
\( t \leftarrow t + 1 \)

for \( (s_t, a_t, r_{t+1}, s_{t+1}) \in D_\pi \) do

\( \delta_t \leftarrow \begin{cases} r_{t+1} - V(s_t), & \text{if } s_{t+1} \in S^* \\ r_{t+1} + \gamma V(s_{t+1}) - V(s_t), & \text{otherwise} \end{cases} \)
\( y_t \leftarrow \begin{cases} a_t, & \text{if } \delta_t > 0 \\ u_t, & \text{otherwise} \end{cases} \)
Store \( (s_t, y_t) \) into \( D_{\text{actor}} \)

Update actor minimizing the MSE with Rprop:

\[
\pi \leftarrow \arg\min_{\pi \in \mathcal{F}_\pi} \sum_{(s_t, y_t) \in D_{\text{actor}}} (\pi(s_t) - y_t)^2
\]

\( V_0 \leftarrow V \)

for \( k \leftarrow 1 \) to \( K \) do

for \( (s_t, r_{t+1}, s_{t+1}) \in D_\pi \) do

\( v_{k,t} \leftarrow \begin{cases} r_{t+1}, & \text{if } s_{t+1} \in S^* \\ r_{t+1} + \gamma V_{k-1}(s_{t+1}), & \text{otherwise} \end{cases} \)
Store \( (s_t, v_{k,t}) \) in \( D_{\text{critic},k} \)

Update critic minimizing MSE with Rprop:

\[
V_k \leftarrow \arg\min_{V \in \mathcal{F}_V} \sum_{(s_t, v_{k,t}) \in D_{\text{critic},k}} (V(s_t) - v_{k,t})^2
\]

\( V \leftarrow V_K \)
Clear \( D_\pi, D_{\text{actor}} \) and \( D_{\text{critic},*} \)

**Method**

Actor-critic is a solution to handle continuous actions spaces in Markov Decision Process \((S, A, R, T, \gamma)\). The critic learns the value-function \( V_\pi \) that describes the expected return following \( \pi \) after the state \( s \). The goal of the actor is to find which state, by finding the best policy according to \( V \):

\[
\begin{align*}
V_\pi(s_t) &= \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right] \\
\pi V(s_t) &= \arg\max_{a \in A} T(s_{t+1} \mid s_t, a) \left[ R(s_t, a, s_{t+1}) + \gamma V_\pi(s_{t+1}) \right]
\end{align*}
\]

**Contributions**

Inspired by Fitted Q Iteration (FQI) \([1]\) and Continuous Actor Critic Learning Automaton (CACLA) \([2]\), we formulated a new on-policy, non-linear, off-line, model-free, actor-critic algorithm. Unlike FQI, it deals with continuous action and state spaces and performs better than CACLA with fewer meta-parameters.

**Results**

Median and quartile of the best registered performance in Acrobot (lower better) and Cartpole (higher better) environment during RL learning.

**Futures Directions**

In order to increase data efficiency, \( D_\pi \) should not be cleared and the sample generated from the previous episodes should be used to improve the actor and the critic updates.

For the critic, it can be done through the importance sampling term \( \frac{\pi(a_t | s_t)}{\pi'(a_t | s_t)} \) making the actor-critic off-policy. The main challenge is how to use the previous samples to update the policy parameters.

**References**


**Source Code**

The source code and data are available at:
drl.gforge.inria.fr