**Problem**

Reinforcement learning is a framework for solving sequential problems where an agent interacts with its environment and adapts its policy based on a reward signal. We tackle the problem of learning a deterministic policy $\mu$ with continuous state and action spaces in the model-free setting where the exploration strategy is Gaussian: $\pi_{\theta,\sigma}(s) = N(\mu(s), \sigma^2I)$.

**Contributions**

We provide theoretical explanations and properties of an unorthodox policy update: Continuous Actor-Critic (CAC) [Van Hasselt and Wiering, 2007]. We propose a new trust region actor-critic algorithm based on this CAC update, Penalized Neural Fitted Actor-Critic (PeNFAC), which surpasses the state-of-the-art algorithms to learn deterministic policies in several classic control problems.

**Policy Gradient**

Objective: $J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$  
Iterative policy update: $\theta_{t+1} \leftarrow \theta_t + \alpha \Delta_\theta$  
Policy gradient: $\Delta_\theta = \nabla_\theta J(\theta)$

**Stochastic Policy Gradient (SPG)** [Sutton et al., 1999]

$$\Delta_{\text{SPG}}(s, \mu_o) = A^*(s, a) \nabla_\theta \log \pi_o(a|s) |_{\mu_o(s)}$$

**Deterministic Policy Gradient (DPG)** [Prokhorov and Wunsch, 1997; Silver et al., 2014]

$$\Delta_{\text{DPG}}(s, \mu_o) = \nabla_a Q^*(s, a) \nabla_\theta \mu_o(a) |_{\mu_o(s)}$$

**Trust Region**

Stochastic: [Kakade and Langford, 2002]

$$J(\tilde{\pi}) = J(\pi) + \int_S d\pi^*(s) \int_A \tilde{\pi}(a|s) A^*(s, a) \nabla_\theta \log \pi_o(a|s) |_{\mu_o(s)} ds$$

Deterministic with $\pi$ as behavior policy:

$$J(\tilde{\mu}) = J(\mu) + \int_S d\mu^*(s) \int_A \tilde{\pi}(a|s) A^*(s, a) \nabla_\theta \mu_o(a) |_{\mu_o(s)} ds$$

Approximation error:

$$\left| \int_S d\tilde{\pi}(s) A^*(s, \tilde{\mu}(s)) - \int_S d\pi(s) A^*(s, \mu(s)) \right| \leq \epsilon L \sigma \sqrt{N}$$

where $\epsilon = \max_{s,a} |A^*(s,a)|$ and $L$ is a Lipschitz constant.

$\rightarrow$ keeps $\sigma$ small, and $\tilde{\mu}$ and $\mu$ close

**Continuous Actor Critic Policy Update**

$$\Delta_{\text{CAC}}(s, \mu_o) = \frac{1}{L} \sum_{s, a, \mu(s)} \left( \frac{1}{L} \int_S d\pi_o(a|s) A^*(s, a) \nabla_\theta \log \pi_o(a|s) |_{\mu_o(s)} \right)$$

**Relation with Stochastic Policy Gradient**

Rewriting of SPG: $\nabla_\theta J(\pi_{o,\theta}) = \int_S d\pi_o(s) \int_A \pi_o(a|s) A^*(s, a) \nabla_\theta \log \pi_o(a|s) |_{\mu_o(s)} ds,$

Rewriting of CAC: $\nabla_\theta J(\pi_{o,\theta}) = \int_S d\pi_o(s) \int_A \pi_o(a|s) A^*(s, a) \nabla_\theta \mu_o(a) |_{\mu_o(s)} ds,$

- CAC is only attracted by good actions: moving away from the current deterministic position in continuous spaces will not necessarily lead to an improvement.

**Relation with Deterministic Policy Gradient**

For a fixed state, when the exploration tends to zero, CAC maintains the sign of the DPG update with scaled magnitudes:

$$\lim_{\sigma \to 0} \Delta_{\text{CAC}}(s, \mu_o) = g^+(s, \pi) \circ \Delta_{\text{DPG}}(s, \mu_o),$$

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.

- DPG needs to learn the action-value function $Q$ whereas the state-value function $V$ is enough for CAC
- DPG makes the assumption that it can access estimate of $\nabla_a Q$
- TD error is a good estimation of the advantage function in nearly deterministic MDPs

**Practical Algorithm and Experimental Results**

PeNFAC is a batch version of CAC with trust region. The $\lambda$-returns are used to estimate the advantage function $A$ and the critic learns the state-value function $V$.

Actor update:

$$\sum_{t=0}^{\infty} \Delta_{\text{CAC}}(s, \mu_o) + \beta \nabla_\theta \|\theta_{\text{MAC}}(s_t) - \mu_o(s_t)\|^2_2$$

**References**


**Futures Directions**

In order to increase data efficiency, we want to take into account off-policy updates [Imani et al., 2018] and move to more informed exploration strategy. To improve computation efficiency, we want to design distributed implementations [Espeholt et al., 2018].

**Source Code**

Free open source algorithms and benchmarking problems