

# 08MIAR-Aprendizaje por refuerzo

**Sesión 7 – Espacios de acciones continuos:  
DDPG & PPO**



**Universidad  
Internacional  
de Valencia**

De:



Planeta Formación y Universidades

# Índice

Introducción

*Deep Deterministic Policy Gradient (DDPG)*

Algoritmo: DDPG

*Proximal Policy Optimization (PPO)*

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

# Índice

## Introducción

*Deep Deterministic Policy Gradient (DDPG)*

Algoritmo: DDPG

*Proximal Policy Optimization (PPO)*

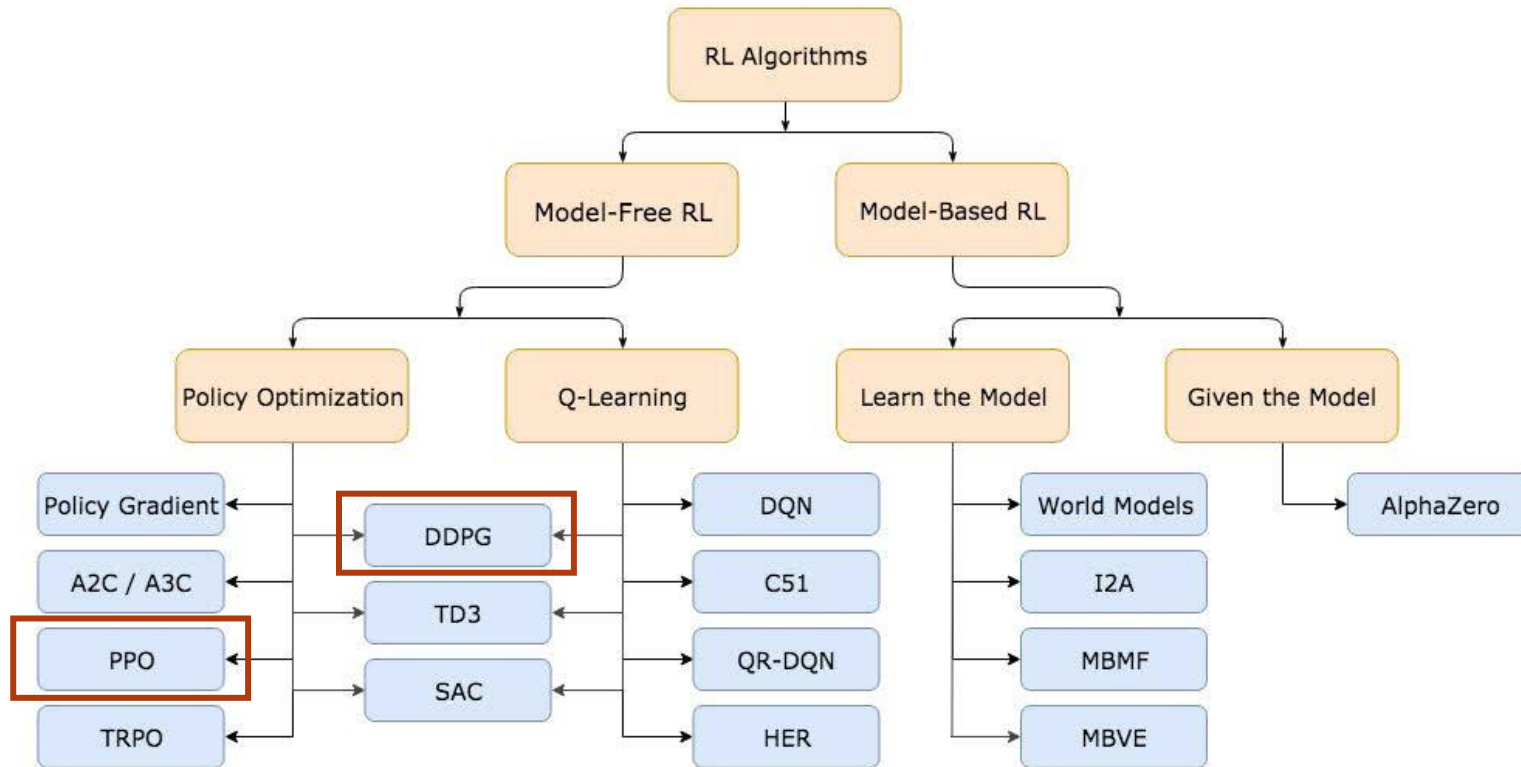
Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

# Introducción



# Introducción

Una vez hemos estudiado las diferentes versiones de algoritmos de la familia de Policy gradients vamos a entrar en detalle en dos soluciones muy extendidas dentro del mundo del aprendizaje por refuerzo, **DDPG y PPO**.

De hecho, PPO se puede interpretar como un siguiente nivel dentro de la evolución natural de la familia de Policy Gradient.

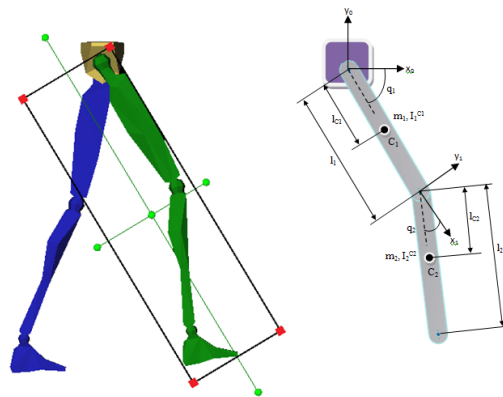
Por su parte, DDPG es un algoritmo que combina características de DQN y de Policy Gradient, sacando partido de ambos enfoques.

Es importante resaltar cómo estos dos algoritmos (al igual que la familia de Policy Gradients) **pueden trabajar en espacios de acciones continuos**. Sin embargo, **DDPG está especialmente diseñado para ello, y PPO es una solución más general**.

## Espacio de acciones continuo

Además de desarrollarse en **espacios de acciones continuos**, los algoritmos de DDPG y PPO incluyen **novedades** también desde el punto de vista **del aprendizaje**, del uso que pueden hacer de una ejecución multiproceso o de la combinación de diferentes arquitecturas de modelos que pertenecen a diferentes definiciones.

Estas características hacen que estos dos algoritmos sean muy utilizados en entornos que, además de la necesidad de trabajar en espacios continuos y complejos, necesiten de una **alta carga computacional** en la simulación del entorno, como pueden ser entornos de **robótica o de navegación autónoma**.



Movimientos: izq-derecha-arriba-abajo

**Discreto** –  $a = [0 \ 0 \ 1 \ 0]$

**Continuo** –  $a = [-1 \ -0.45 \ 0.8 \ 0.3]$

↑ ↑ ↑ ↑  
Grados, normalizados  
entre  $[-1, 1]$

# Índice

Introducción

***Deep Deterministic Policy Gradient (DDPG)***

Algoritmo: DDPG

***Proximal Policy Optimization (PPO)***

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

# Introducción

---

## Deterministic Policy Gradient Algorithms

---

**David Silver**  
DeepMind Technologies, London, UK

DAVID@DEEPMIND.COM

**Guy Lever**  
University College London, UK

GUY.LEVER@UCL.AC.UK

**Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller**  
DeepMind Technologies, London, UK

\*@DEEPMIND.COM

### Abstract

In this paper we consider *deterministic* policy gradient algorithms for reinforcement learning with continuous actions. The deterministic policy gradient has a particularly appealing form: it is the expected gradient of the action-value function. This simple form means that the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient. To ensure adequate exploration, we introduce an off-policy actor-critic algorithm that learns a deterministic target policy from an exploratory behaviour policy. We demonstrate that deterministic policy gradient algorithms can significantly outperform their stochastic counterparts in high-dimensional action spaces.

case, as policy variance tends to zero, of the stochastic policy gradient.

From a practical viewpoint, there is a crucial difference between the stochastic and deterministic policy gradients. In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions.

In order to explore the full state and action space, a stochastic policy is often necessary. To ensure that our deterministic policy gradient algorithms continue to explore satisfactorily, we introduce an off-policy learning algorithm. The basic idea is to choose actions according to a stochastic behaviour policy (to ensure adequate exploration), but to learn about a deterministic target policy (exploiting the ef-

**Deep Deterministic Policy Gradient (DDPG)** es un algoritmo que **aprende** de manera simultánea **una función Q y una policy (Actor-Critic por tanto)**.

Usa una estrategia **off-policy** y la ecuación de Bellman para modelar la función Q. Para entrenar la **policy**, aprovecha la **función Q modelada como factor de relevancia en la misma**.

- DDPG es un algoritmo off-policy (Por lo tanto, no tendremos los discounted rewards calculados)
- Algoritmo híbrido.
- DDPG sólo se puede utilizar en entornos con espacios de acciones continuos
- DDPG se puede ver como la “**forma de aplicar Q-learning en espacios de acciones continuos**”.

<https://spinningup.openai.com/en/latest/algorithms/ddpg.html>



# Introducción

## Deterministic Policy Gradient Algorithms

**David Silver**  
 DeepMind Technologies, London, UK

DAVID@DEEPMIND.COM

**Guy Lever**  
 University College London, UK

GUY.LEVER@UCL.AC.UK

**Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller**  
 DeepMind Technologies, London, UK

\*@DEEPMIND.COM

### Abstract

In this paper we consider *deterministic* policy gradient algorithms for reinforcement learning with continuous actions. The deterministic policy gradient has a particularly appealing form: it is the expected gradient of the action-value function. This simple form means that the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient. To ensure adequate exploration, we introduce an off-policy actor-critic algorithm that learns a deterministic target policy from an exploratory behaviour policy. We demonstrate that deterministic policy gradient algorithms can significantly outperform their stochastic counterparts in high-dimensional action spaces.

case, as policy variance tends to zero, of the stochastic policy gradient.

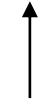
From a practical viewpoint, there is a crucial difference between the stochastic and deterministic policy gradients. In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions.

In order to explore the full state and action space, a stochastic policy is often necessary. To ensure that our deterministic policy gradient algorithms continue to explore satisfactorily, we introduce an off-policy learning algorithm. The basic idea is to choose actions according to a stochastic behaviour policy (to ensure adequate exploration), but to learn about a deterministic target policy (exploiting the ef-


En espacio de acciones continuas, **utilizar una política voraz para la exploración y maximización de la función Q resulta problemático.**

Por el contrario, una alternativa computacionalmente abarcable consiste en **actualizar la policy en dirección al gradiente de la función Q estimada.**

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s \sim \rho^{\mu^k}} \left[ \nabla_{\theta} Q^{\mu^k}(s, \mu_{\theta}(s)) \right]$$



Función  
Q (critic)



Policy  
(actor)

# Introducción

**Error** mediante Bellman para entrenar el **critic**.

**Recompensa esperada a futuro.** Utilizamos Q en el estado siguiente. Realmente para esto utilizaremos una target network.

(1) Actualizamos  
Función Q (**critic**)

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t)$$

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}$$

(2) Actualizamos la Policy  
(**actor**)

**Gradientes de la policy**

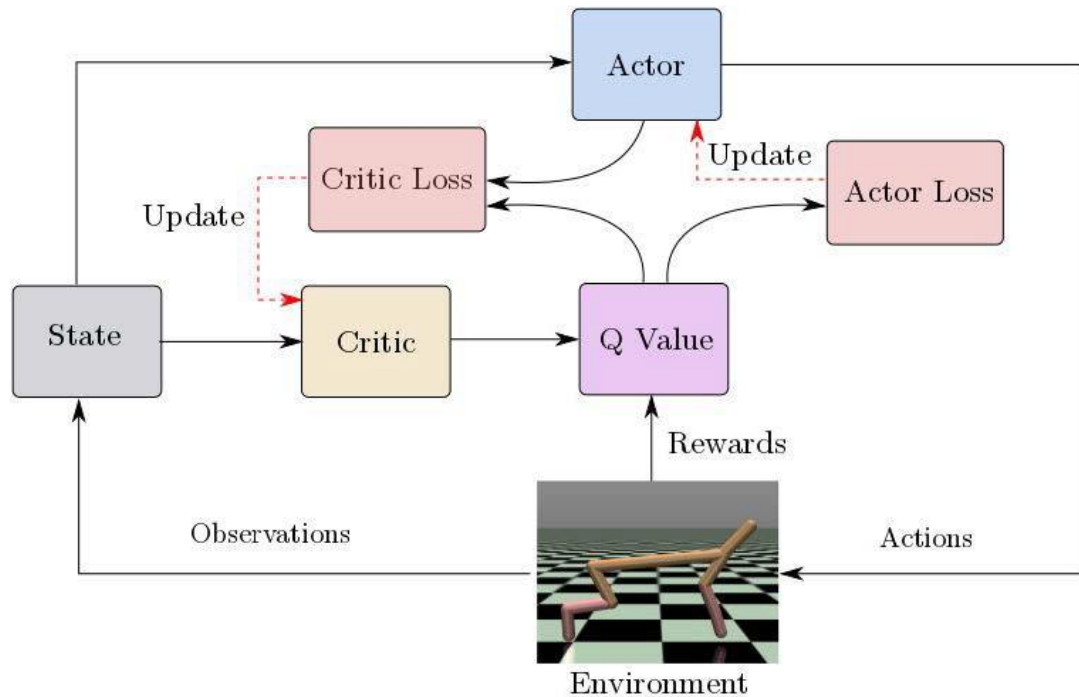
Fijaros! Obtenemos la Q para las acciones dadas por el actor. **Queremos maximizar la Q, por ello ponderamos con sus gradientes, que nos dirigen a maximizarla.**

## Características principales

Algunas de las características (y decisiones) que se nos presentan con DDPG son:

- El **espacio de acciones es continuo**. Cómo podemos aplicar una estrategia seleccionando la acción que maximice la recompensa esperada en un espacio continuo?
- Al ser **off-policy**, se utilizará un **replay buffer** para almacenar la experiencia que se va acumulando.
- Se definirán **cuatro redes neuronales diferentes**: Dos redes para predicción (**actor y critic networks**) y dos redes target (**actor y critic target networks**).
- El proceso de **exploración es diferente** a los estudiados hasta ahora. Se llevará a cabo incluyendo una **función de ruido en el momento de la selección de la acción**.

# Características principales



```
class Actor(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, learning_rate = 3e-4):
        super(Actor, self).__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.linear3 = nn.Linear(hidden_size, output_size)

    def forward(self, state):
        """
        Param state is a torch tensor
        """
        x = F.relu(self.linear1(state))
        x = F.relu(self.linear2(x))
        x = torch.tanh(self.linear3(x))

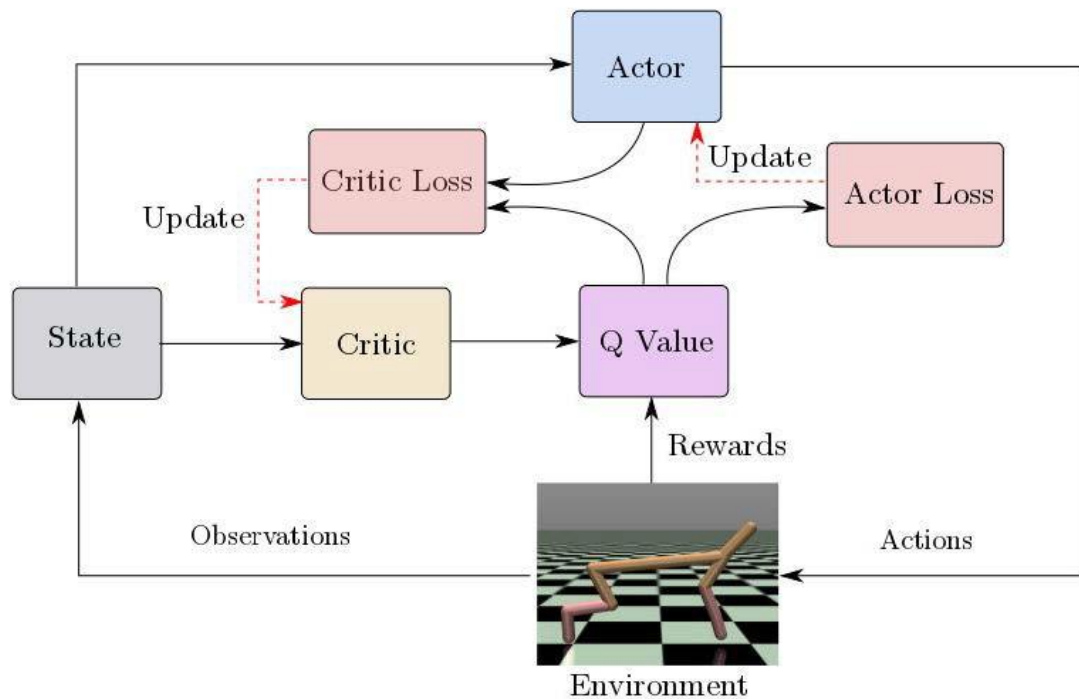
        return x
```

```
class Critic(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(Critic, self).__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.linear3 = nn.Linear(hidden_size, output_size)

    def forward(self, state, action):
        """
        Params state and actions are torch tensors
        """
        x = torch.cat([state, action], 1)
        x = F.relu(self.linear1(x))
        x = F.relu(self.linear2(x))
        x = self.linear3(x)

        return x
```

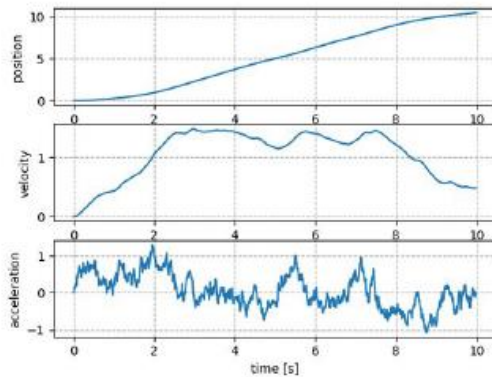
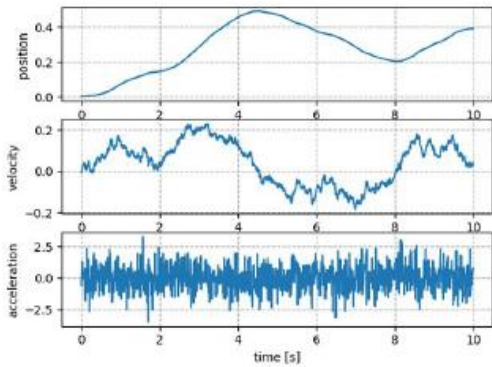
## Características principales



```
# Critic loss
Qvals = self.critic.forward(states, actions)
next_actions = self.actor_target.forward(next_states)
next_Q = self.critic_target.forward(next_states, next_actions.detach())
Qprime = rewards + self.gamma * next_Q
critic_loss = self.critic_criterion(Qvals, Qprime)

# Actor loss
policy_loss = -self.critic.forward(states, self.actor.forward(states)).mean()
```

## Características principales



El proceso de exploración tipo que encontraremos en el algoritmo será llevado a cabo utilizando el **proceso de Ornstein-Uhlenbeck**.

La idea de este proceso es utilizar un proceso que mantenga una **correlación temporal** para que la exploración sea más suave que con otras posibles distribuciones.

$$\mu'(s_t) = \mu(s_t | \theta_t^\mu) + \mathcal{N}$$

[https://en.wikipedia.org/wiki/Ornstein%E2%80%93Uhlenbeck\\_process](https://en.wikipedia.org/wiki/Ornstein%E2%80%93Uhlenbeck_process)  
<https://www.quora.com/Why-do-we-use-the-Ornstein-Uhlenbeck-Process-in-the-exploration-of-DDPG>

# Índice

Introducción

*Deep Deterministic Policy Gradient (DDPG)*

**Algoritmo: DDPG**

*Proximal Policy Optimization (PPO)*

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

# Algoritmo: DDPG

---

**Algorithm 1** DDPG algorithm
 

---

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .

Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer  $R$

**for** episode = 1, M **do**

  Initialize a random process  $\mathcal{N}$  for action exploration

  Receive initial observation state  $s_1$

**for** t = 1, T **do**

    Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise

    Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$

    Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$

    Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$

    Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

    Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

  Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

**end for**

**end for**

---



# Índice

Introducción

*Deep Deterministic Policy Gradient (DDPG)*

Algoritmo: DDPG

***Proximal Policy Optimization (PPO)***

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

# Introducción

## Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov  
OpenAI  
{joschu, filip, prafulla, alec, oleg}@openai.com

### Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a “surrogate” objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

La idea clave en la que se centra el algoritmo de PPO es:

**Cómo podemos mejorar lo máximo posible la policy actual, usando la trayectoria recolectada, controlando que el aprendizaje no colapse?**

- Algoritmo basado en otro conocido como TRPO.
- PPO es un **algoritmo on-policy**. Pertenece a la familia de Policy Gradient.
- PPO puede ser utilizado para entornos con **espacios de acciones tanto discretos como continuos**.
- Normalmente, las implementaciones que encontraremos de PPO utilizan una **ejecución multiproceso para tareas de recolección de datos**.

<https://spinningup.openai.com/en/latest/algorithms/ppo.html>

## Características principales

Algunas de las características (y decisiones) que se nos presentan con PPO son:

- El espacio de **acciones** puede ser **discreto o continuo**.
- Al igual que los otros algoritmos de su familia, sigue una filosofía **on-policy**. Nuevamente, tenemos que decidir el tamaño de la **trayectoria** para almacenar los datos que se utilicen para el entrenamiento. En este caso, tenemos que tener en cuenta que la ejecución **multiproceso** aumenta el volumen recolectado.
- La arquitectura de modelo utilizada es similar a **Actor-Critic**.
- La **exploración** se realiza de la misma manera que los algoritmos Actor-Critic, utilizando una **distribución de probabilidad aleatoria** ponderada.

## Objetivo de PPO

Algunas de las **limitaciones que presenta policy gradients** (de algunas ya hemos hablado):

- Ineficacia de las muestras: Las muestras sólo se utilizan una vez. Después, la política se actualiza y la nueva política se utiliza para muestrear otra trayectoria. Como el muestreo suele ser caro, esto puede resultar prohibitivo. → A2C/A3C multiproceso
- Alta varianza de la recompensa: El gradiente de la política es un enfoque de aprendizaje Monte Carlo, que tiene en cuenta la trayectoria completa de la recompensa (es decir, un episodio completo). Estas trayectorias suelen tener una gran varianza, lo que dificulta la convergencia. → Actor-Critic
- **Actualizaciones de políticas incoherentes**: las actualizaciones de políticas tienden a sobrepasar y no alcanzar el pico de recompensa, o a detenerse prematuramente. Especialmente en las arquitecturas de redes neuronales, los gradientes desvanecientes y explosivos constituyen un grave problema. **El algoritmo puede no recuperarse de una mala actualización. --> PPO**

## Características principales

La mayor diferencia aparece en la función de coste, PPO ofrece dos versiones diferentes:

- Una función de coste basada en **Kullback-Leibler divergence**
- Una función de coste basada en **Clipped surrogate objective**

Será la **segunda opción** la que trataremos en la asignatura.

## Características principales

La primera diferencia que introduce esta función es la **sustitución del logaritmo de la probabilidad de la acción por el ratio de la probabilidad de la acción con la policy actual en comparación con la versión de la policy anterior.**

Este ratio será mayor que 1 cuando la acción es más probable con la nueva policy y estará entre 0 y 1 cuando la acción sea menos probable.

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[ \log \pi_{\theta}(a_t | s_t) \hat{A}_t \right]$$

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[ r_t(\theta) \hat{A}_t \right]$$

## Características principales

Para **estabilizar** el proceso de aprendizaje (con probabilidades que sean muy grandes), PPO introduce **dos límites** para los casos en los que este ratio se dispare y, por tanto, los cambios en la policy sean muy drásticos.

Básicamente, junto al ratio calculado se define un rango (a partir de un nuevo **hiperparámetro**, con valor por defecto 0.1/0.2) para que la actualización de los pesos del modelo se lleve a cabo con el valor mínimo de este rango. **Así aseguramos que la actualización se va haciendo de forma “suave” y en una buena dirección.**

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[ \min \left( \overbrace{r_t(\theta) \hat{A}_t}^{\text{same objective from before}}, \overbrace{\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t}^{\text{same, but } r(\theta) \text{ is clipped between } (1 - \epsilon, 1 + \epsilon)} \right) \right]$$

min of the same objective from before and the clipped one

# Características principales

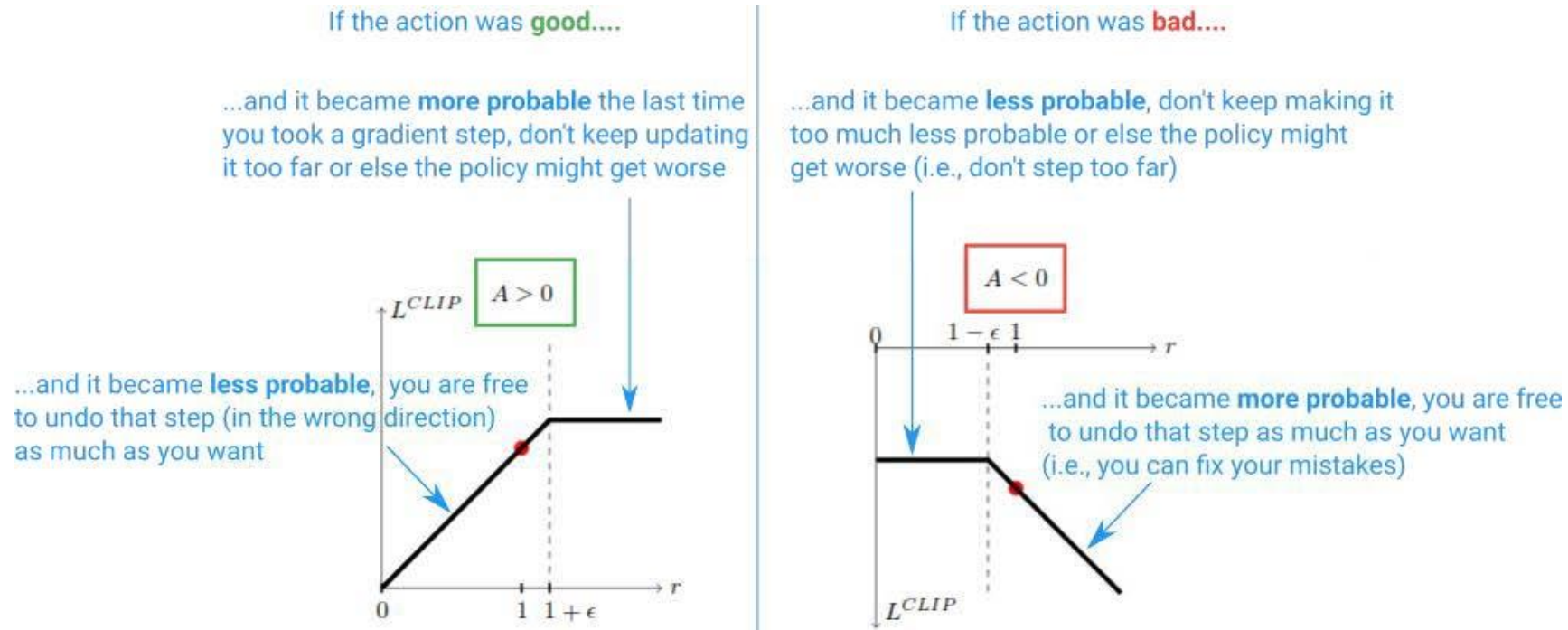


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function  $L^{CLIP}$  as a function of the probability ratio  $r$ , for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e.,  $r = 1$ . Note that  $L^{CLIP}$  sums many of these terms.

<https://stackoverflow.com/questions/46422845/what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl>



# Índice

Introducción

*Deep Deterministic Policy Gradient (DDPG)*

Algoritmo: DDPG

*Proximal Policy Optimization (PPO)*

**Algoritmo: PPO**

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

# Algoritmo: PPO

---

## Algorithm 1 PPO-Clip

---

- 1: Input: initial policy parameters  $\theta_0$ , initial value function parameters  $\phi_0$
- 2: **for**  $k = 0, 1, 2, \dots$  **do**
- 3:   Collect set of trajectories  $\mathcal{D}_k = \{\tau_i\}$  by running policy  $\pi_k = \pi(\theta_k)$  in the environment.
- 4:   Compute rewards-to-go  $\hat{R}_t$ .
- 5:   Compute advantage estimates,  $\hat{A}_t$  (using any method of advantage estimation) based on the current value function  $V_{\phi_k}$ .
- 6:   Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left( \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7:   Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left( V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

# Índice

Introducción

*Deep Deterministic Policy Gradient (DDPG)*

Algoritmo: DDPG

*Proximal Policy Optimization (PPO)*

Algoritmo: PPO

**Alternativas a DDPG y PPO**

Conclusiones

Bibliografía recomendada

# Alternativas a DDPG y PPO

<https://arxiv.org/abs/1801.01290>

## Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja<sup>1</sup> Aurick Zhou<sup>1</sup> Pieter Abbeel<sup>1</sup> Sergey Levine<sup>1</sup>

### Abstract

Model-free deep reinforcement learning (RL) algorithms have been demonstrated on a range of challenging decision making and control tasks. However, these methods typically suffer from two major challenges: very high sample complexity and brittle convergence properties, which necessitate meticulous hyperparameter tuning. Both of these challenges severely limit the applicability of such methods to complex, real-world domains. In this paper, we propose soft actor-critic, an off-policy actor-critic deep RL algorithm based on the

of these methods in real-world domains has been hampered by two major challenges. First, model-free deep RL methods are notoriously expensive in terms of their sample complexity. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more. Second, these methods are often brittle with respect to their hyperparameters: learning rates, exploration constants, and other settings must be set carefully for different problem settings to achieve good results. Both of these challenges severely limit the applicability of model-free deep RL to real-world tasks.

$$J(\pi) = \sum_{t=0}^T \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_{\pi}} [r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot | \mathbf{s}_t))]$$

<https://arxiv.org/abs/1707.01495>

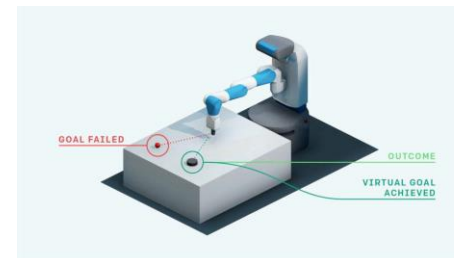
## Hindsight Experience Replay

Marcin Andrychowicz\*, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel<sup>†</sup>, Wojciech Zaremba<sup>†</sup>  
OpenAI

### Abstract

Dealing with sparse rewards is one of the biggest challenges in Reinforcement Learning (RL). We present a novel technique called *Hindsight Experience Replay* which allows sample-efficient learning from rewards which are sparse and binary and therefore avoid the need for complicated reward engineering. It can be combined with an arbitrary off-policy RL algorithm and may be seen as a form of implicit curriculum.

We demonstrate our approach on the task of manipulating objects with a robotic arm. In particular, we run experiments on three different tasks: pushing, sliding, and pick-and-place, in each case using only binary rewards indicating whether or not the task is completed. Our ablation studies show that Hindsight Experience Replay is a crucial ingredient which makes training possible in these challenging environments. We show that our policies trained on a physics simulation can be deployed on a physical robot and successfully complete the task. The video presenting our experiments is available at <https://goo.gl/5MrQnI>.



Sample a set of additional goals for replay  $G := \mathbb{S}(\text{current episode})$   
for  $g' \in G$  do  
     $r' := r(s_t, a_t, g')$   
    Store the transition  $(s_t || g', a_t, r', s_{t+1} || g')$  in  $R$   
end for

# Índice

Introducción

*Deep Deterministic Policy Gradient (DDPG)*

Algoritmo: DDPG

*Proximal Policy Optimization (PPO)*

Algoritmo: PPO

Alternativas a DDPG y PPO

**Conclusiones**

Bibliografía recomendada

## Conclusiones

- Siempre que trabajemos en espacios de acciones continuos tendremos que utilizar algoritmos de la **familia de Policy Gradients** (como PPO) o utilizar **enfoques híbridos** que nos permitan utilizar funciones Q (como DDPG).
- Los **espacios de acciones continuos** son muy comunes en entornos complejos y que necesitan de mucha capacidad computacional, como por ejemplo robótica o conducción autónoma.
- Además de funcionar muy bien en espacios de acciones continuos, **PPO es uno de los algoritmos que forman parte del estado del arte actual en multitud de retos**. Además, principalmente por su implementación intuitiva, hace que sea uno de los algoritmos por defecto cuando implementamos una solución de aprendizaje por refuerzo.

# Índice

Introducción

*Deep Deterministic Policy Gradient (DDPG)*

Algoritmo: DDPG

*Proximal Policy Optimization (PPO)*

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

**Bibliografía recomendada**

## Bibliografía recomendada

- *Learning Dexterity*, OpenAI

<https://openai.com/blog/learning-dexterity/>

- *What is the way to understand Proximal Policy Optimization Algorithm in RL?*, Cyberman, A.  
Stackoverflow

<https://stackoverflow.com/questions/46422845/what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl>





viu

**Universidad**  
Internacional  
de Valencia