

3

DDPG

INTRODUCCIÓN

- ▶ Supongamos que se busca controlar un brazo robótico.
- ▶ Acciones discretas: subir, bajar, agarrar, soltar.



BCN3D MOVEO (brazo robótico de código abierto impreso en 3D)



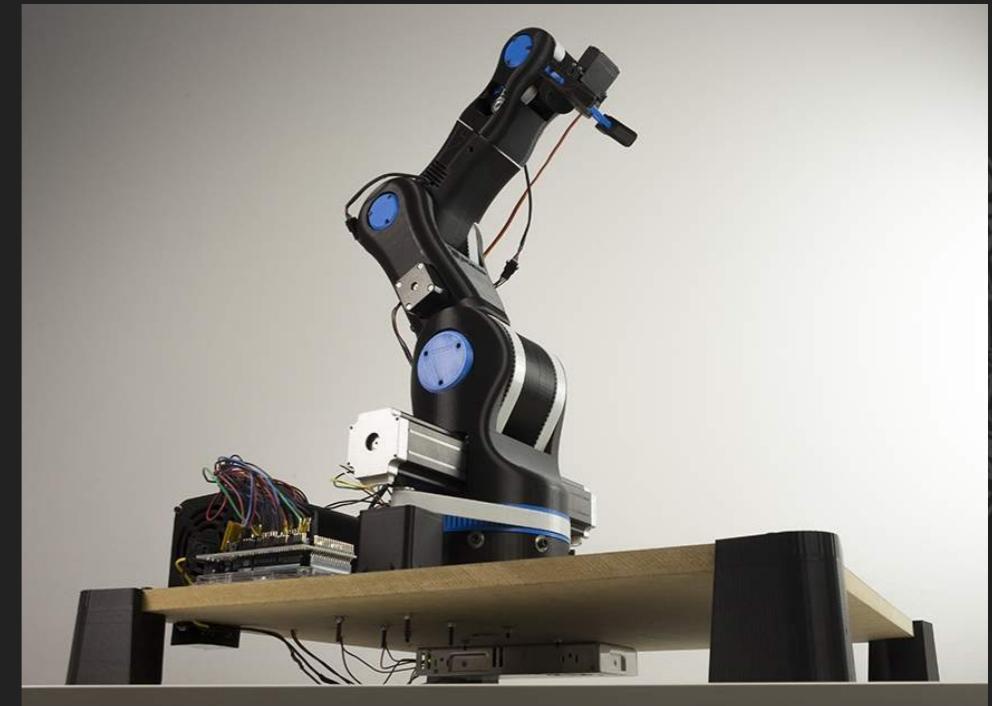
INTRODUCCIÓN

- ▶ Pero, si los movimientos son delicados:

- ✓ Girar pinza 10°
 - ✓ Mover brazo 0.7 mm.
 - ✓ Aplicar fuerza de 2N con la pinza.

- ▶ En estas situaciones, se necesita más que simplemente elegir una acción: hay que decidir **cuánta acción se requiere**.

- ▶ Este es el ámbito de los **espacios de acción continua**, y es aquí donde el **Gradiente de Política Determinista Profundo (DDPG)** se destaca.



BCN3D MOVEO (brazo robótico de código abierto impreso en 3D)

INTRODUCCIÓN

► DQN (y sus variantes):

✓ ↑ acciones discretas

✓ ↓ acciones continuas

► Gradiente de Política Determinista (DPG) (Silver et al., 2014):

✓ ↓ exploración

✓ ↓ estabilidad

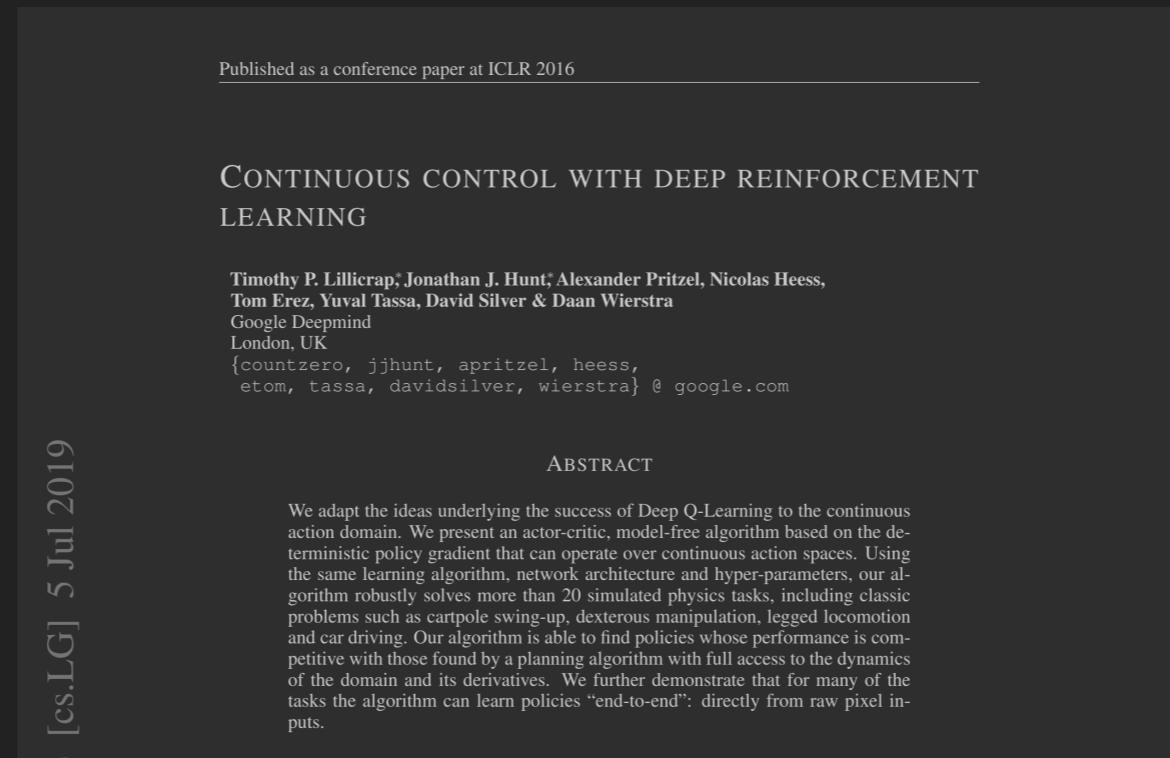
The image shows a dark thumbnail of a research paper. At the top, it reads 'Deterministic Policy Gradient Algorithms'. Below that is the author list: 'David Silver DeepMind Technologies, London, UK', 'Guy Lever University College London, UK', 'Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller DeepMind Technologies, London, UK'. To the right of the authors are their email addresses: 'DAVID@DEEPMIND.COM', 'GUY.LEVER@UCL.AC.UK', and '*@DEEPMIND.COM'. Underneath the author section is a short abstract in white text.

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.

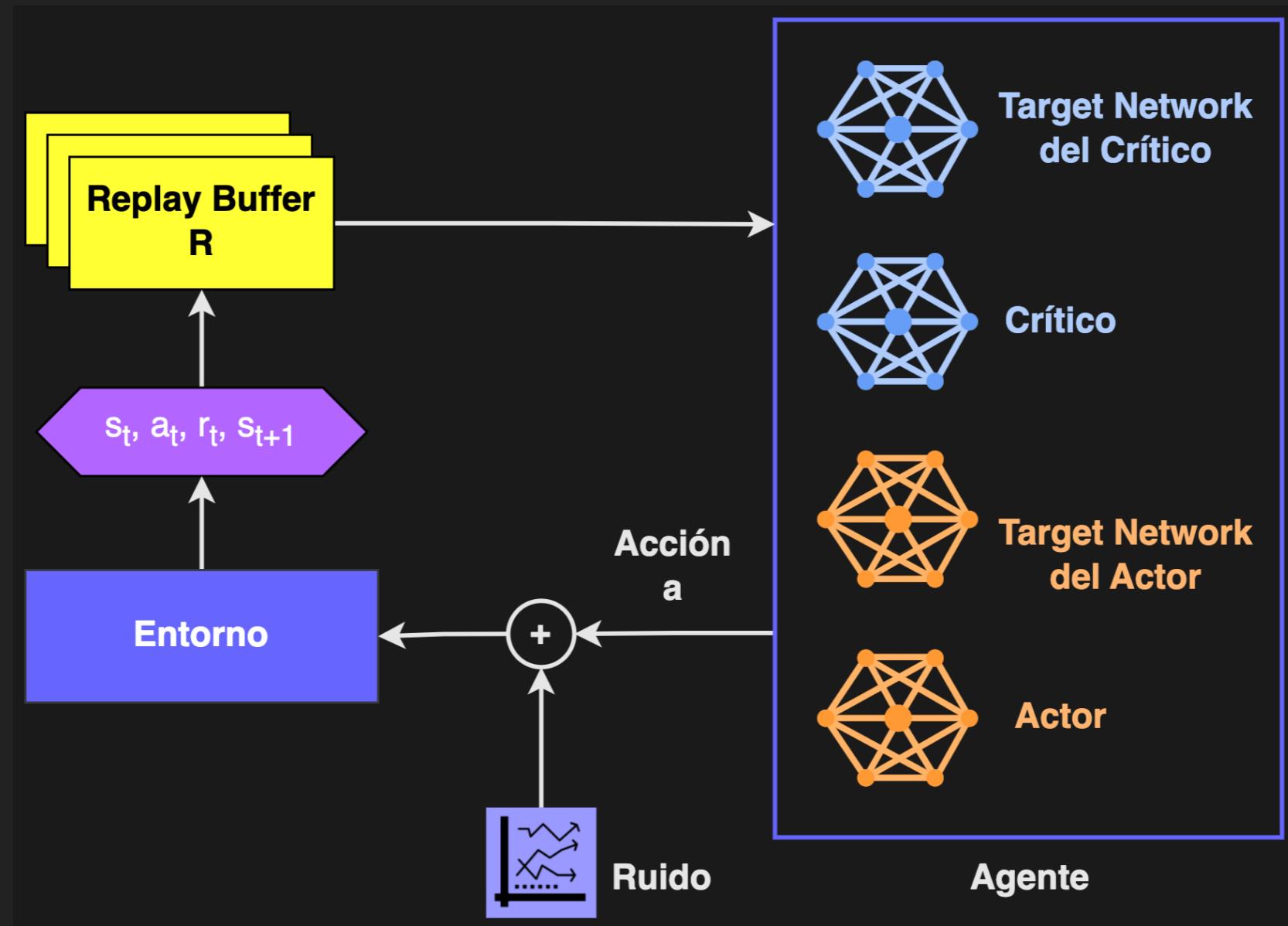


DDPG

- ▶ DDPG fue presentado en 2016 con el título de “Control continuo con aprendizaje por refuerzo profundo” (Lillicrap et al., 2016).
- ▶ DDPG combina las ventajas de DPG y DQN para mejorar la estabilidad y el rendimiento en entornos con espacios de acción continua.



ARQUITECTURA BÁSICA DE DDPG



COMPONENTES DE DDPG

- ▶ Veamos cuales son sus componentes...

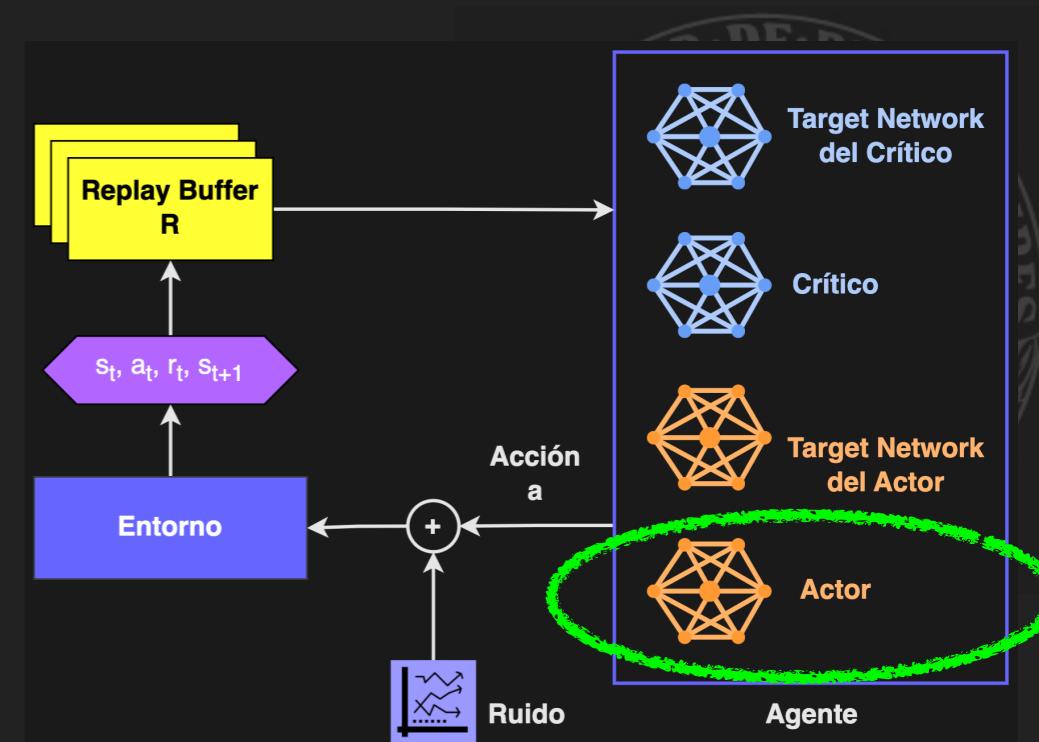


ACTOR (POLICY NETWORK)

► Actor:

- ✓ Aproxima la política óptima determinística (sin exploración).
- ✓ Entrada: Observación o Estado (s).
- ✓ Salida: Acción continua óptima (a).
- ✓ Decide qué acción tomar dado el Estado s en el que se encuentra.
- ✓ Los parámetros de la red (es decir, los pesos) están representados por θ^μ .
- ✓ Asigna el estado actual s_t a una única acción.

$$a_t = \mu(s_t | \theta^\mu)$$

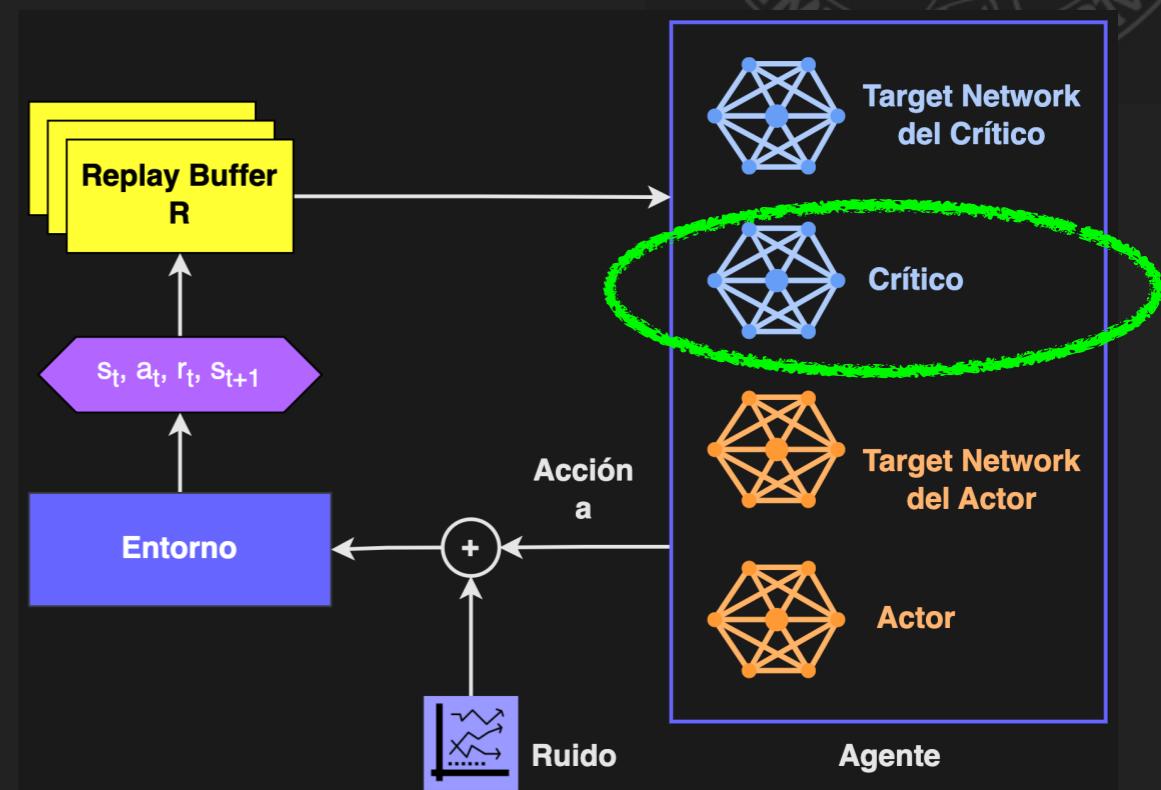


CRÍTICO (Q-NETWORK)

► Crítico (Q-Network)

- ✓ Aproxima la función de valor Q (evalúa o mide qué tan buena es la Acción a_t del actor en un Estado s).
- ✓ Entrada: Estado s , Acción a .
- ✓ Salida: Valor Q esperado -> $Q=R+\text{Gamma}^*Q'$

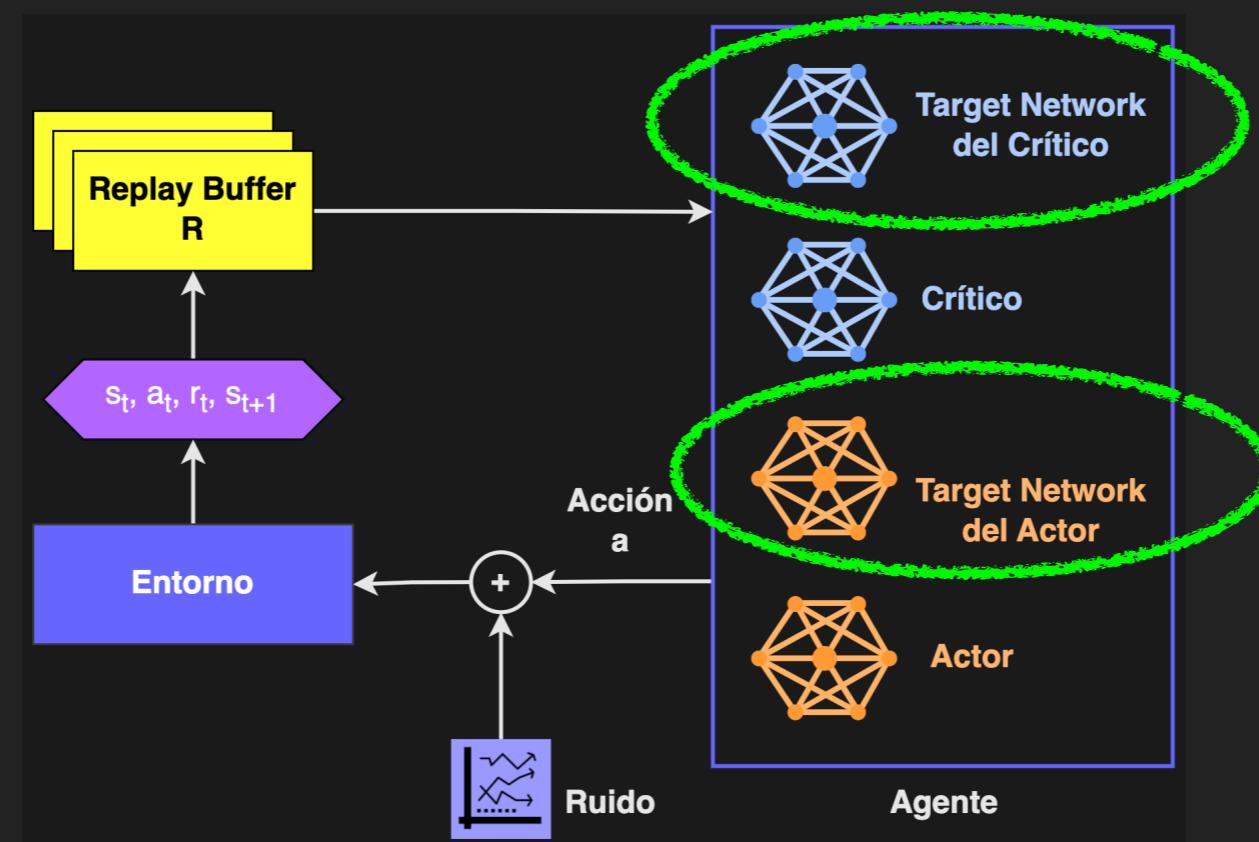
$$Q(s, a | \theta^Q)$$



REDES OBJETIVO (TARGET NETWORKS)

► Redes objetivo (Target Networks)

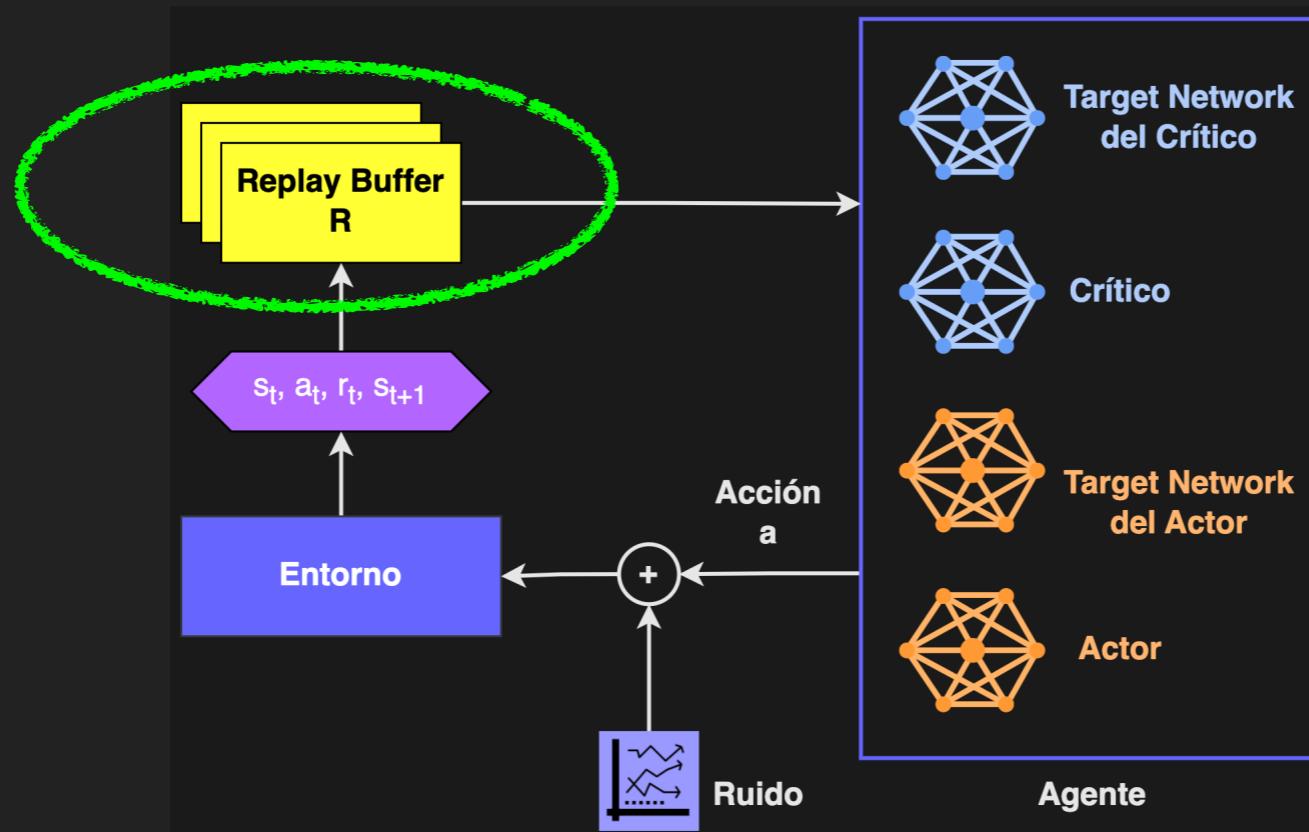
✓ Copias estables del actor y crítico para mejorar la convergencia (similar a DQN).



REPLAY BUFFER

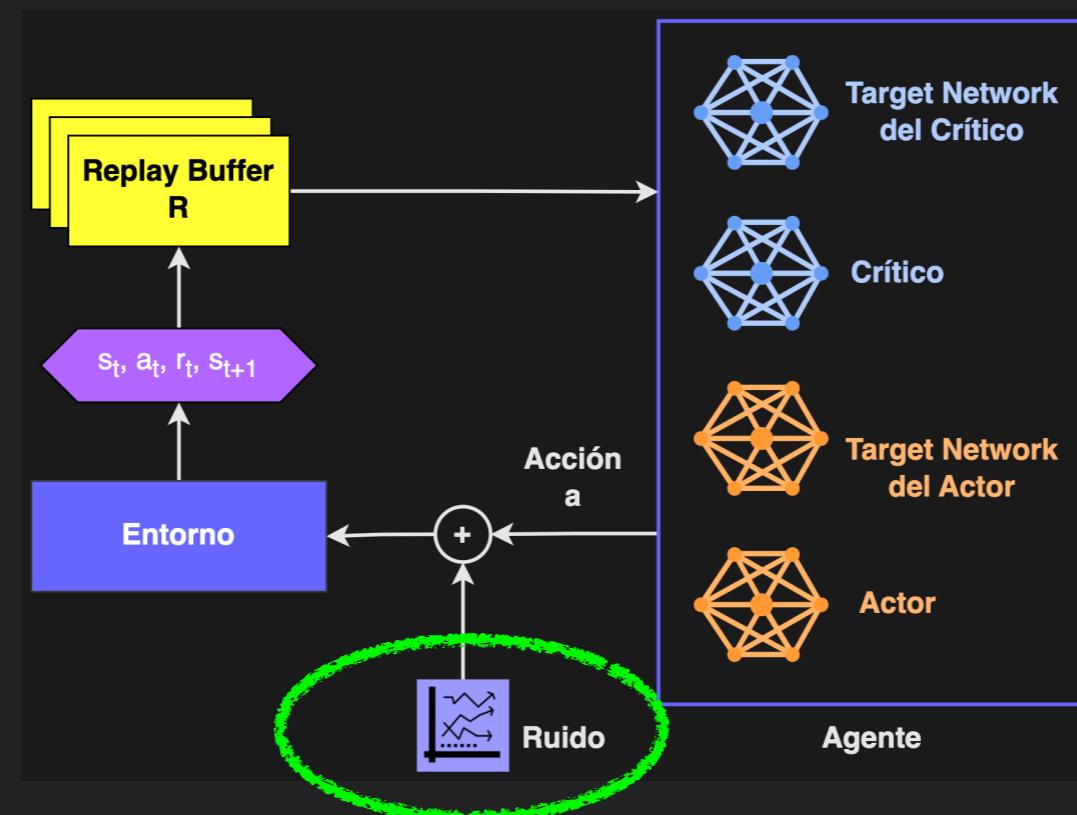
► Replay Buffer:

✓ Almacena experiencias (s, a, r, s') para entrenamiento fuera de política (off-policy).



RUIDO PARA EXPLORACIÓN

- ▶ Como la política es determinística, y por sí sola no explora el entorno, se añade ruido a la acción (ej: OU Noise, ruido gaussiano) para explorar (Uhlenbeck et al., 1930).
- ▶ Originalmente, el paper usa ruido de Ornstein-Uhlenbeck (OU) y sus características son:
 - ✓ Correlacionado en el tiempo → produce trayectorias más suaves (útil para entornos físicos).
 - ✓ Modela procesos estocásticos de forma más realista que un ruido blanco.



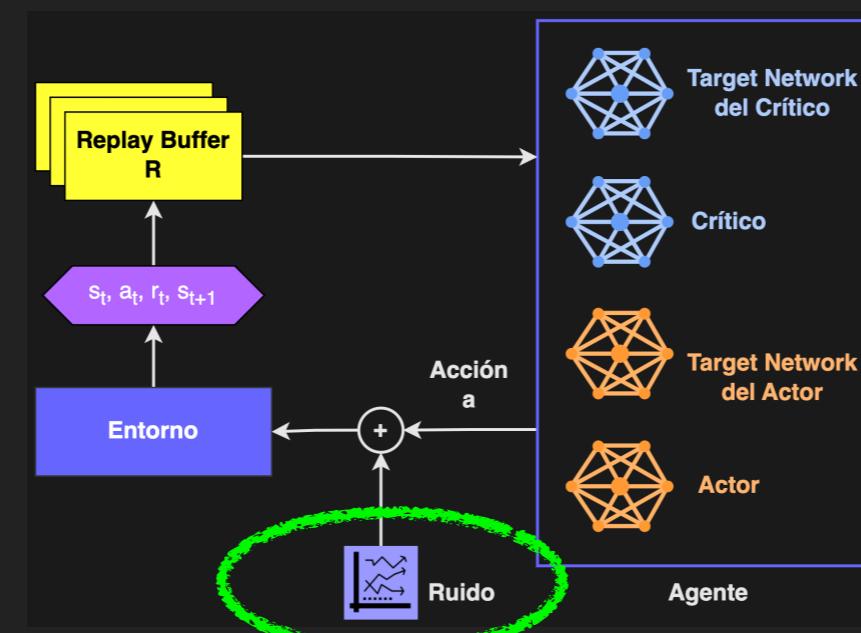
RUIDO PARA EXPLORACIÓN

- El proceso OU se define como:

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t$$

- donde:

- ✓ θ : velocidad de regreso al promedio. A mayor theta regresa mas rápidamente a la media.
- ✓ μ : media (objetivo de retorno o valor al que tiende el ruido).
- ✓ dt : Paso de tiempo (discretización del proceso). Valores pequeños hacen que el ruido cambie más lentamente.
- ✓ σ : Varianza (amplitud del ruido). Es la desviación estándar del ruido.
- ✓ dW_t : ruido gaussiano.



COMO FUNCIONA DDPG?

- ✓ Generación de trayectorias
- ✓ Entrenamiento del Crítico
- ✓ Entrenamiento del Actor
- ✓ Actualización suave de redes objetivo



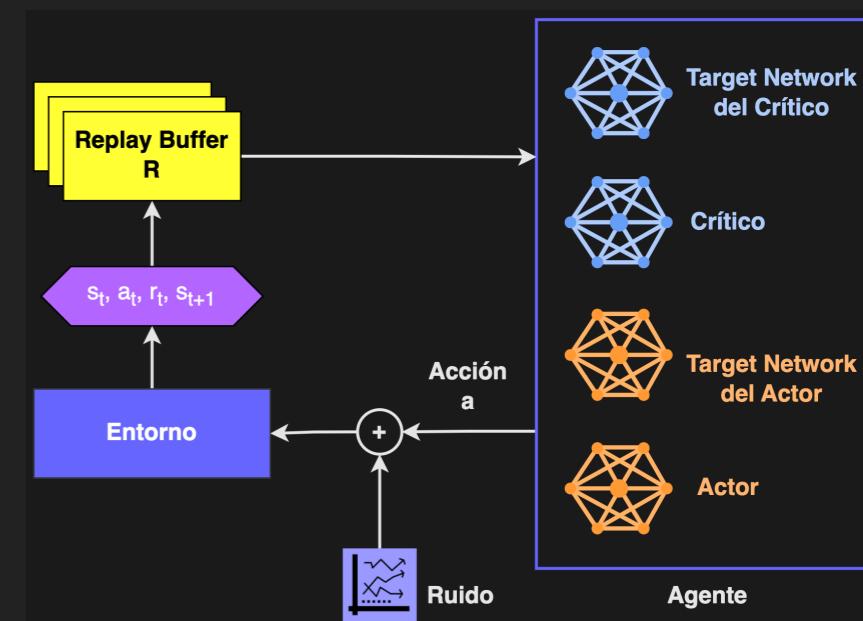
COMO FUNCIONA DDPG?

- ✓ Generación de trayectorias
- ✓ Entrenamiento del Crítico
- ✓ Entrenamiento del Actor
- ✓ Actualización suave de redes objetivo



GENERACIÓN DE TRAYECTORIAS

- ▶ El actor elige una acción $a = \mu(s) + \text{ruido}$.
- ▶ Donde:
 - ✓ μ es la función del actor (una red neuronal).
 - ✓ Dado un **estado s** , devuelve una acción óptima $a = \mu(s)$.
- ▶ A diferencia de los casos de políticas estocásticas (que devuelven una distribución π), $\mu(s)$ siempre devuelve la misma acción para el mismo s . Se usa μ para denotar que es una política determinística.
- ▶ Se ejecuta a , se observa (s, a, r, s') y se guarda en el replay buffer.



COMO FUNCIONA DDPG?

- ✓ Generación de trayectorias
- ✓ Entrenamiento del Crítico
- ✓ Entrenamiento del Actor
- ✓ Actualización suave de redes objetivo



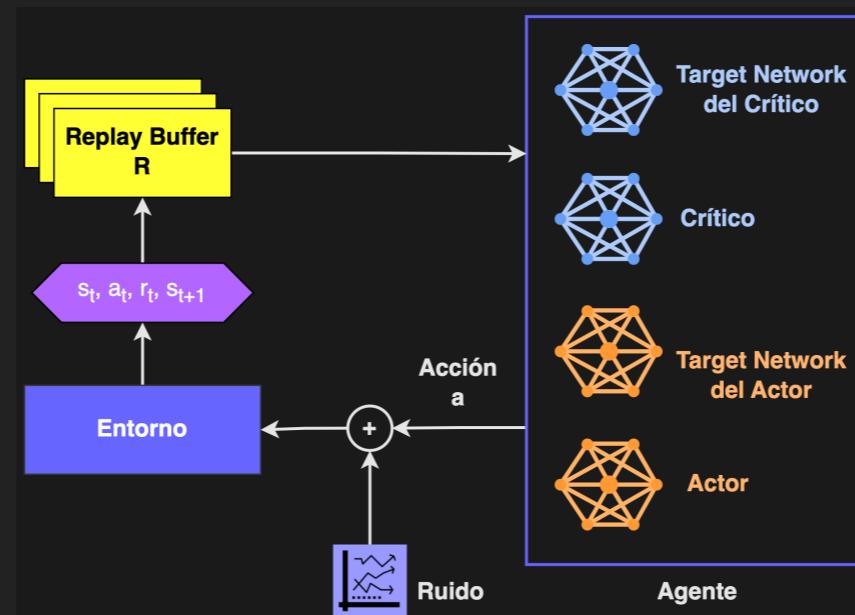
ENTRENAMIENTO DEL CRÍTICO

- ▶ Se muestrea un batch del buffer.
- ▶ Se calcula el target Q usando la red objetivo:

$$y = r + \gamma \cdot Q_{\text{target}}(s', \mu_{\text{target}}(s'))$$

- ▶ Se minimiza el error MSE:

$$L = \frac{1}{N} \sum (y - Q(s, a))^2$$



COMO FUNCIONA DDPG?

- ✓ Generación de trayectorias
- ✓ Entrenamiento del Crítico
- ✓ Entrenamiento del Actor
- ✓ Actualización suave de redes objetivo

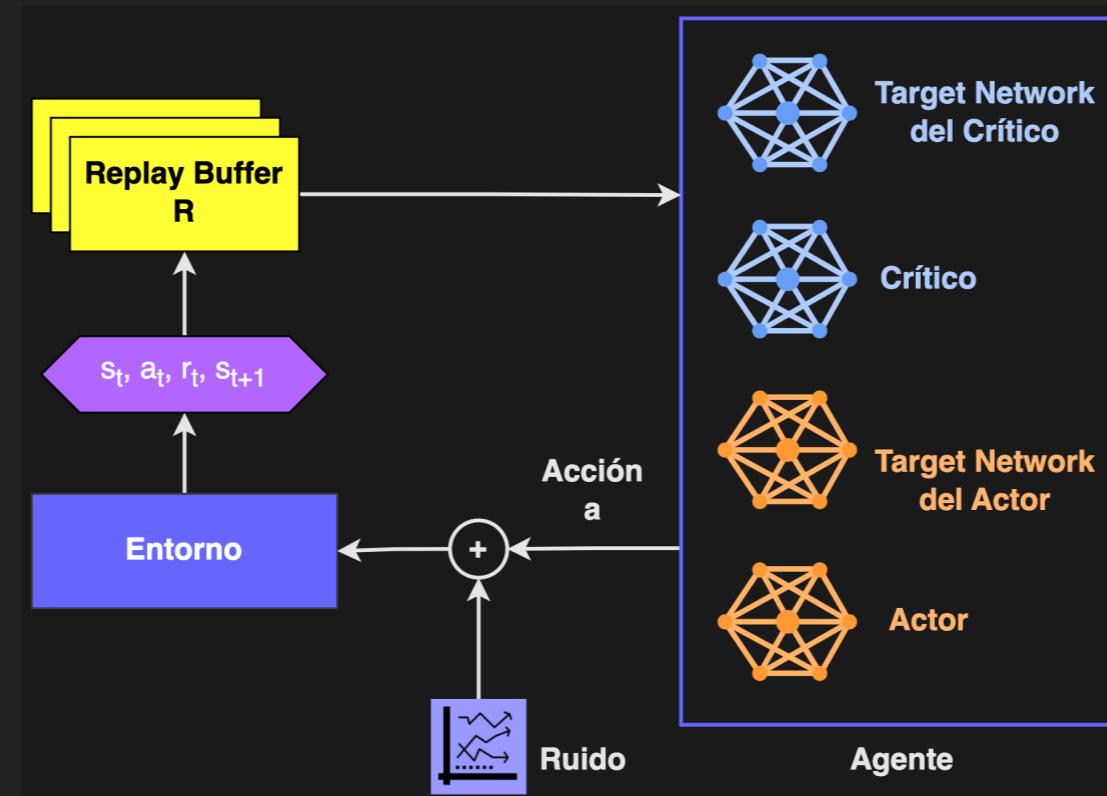


ENTRENAMIENTO DEL ACTOR

- ▶ Se actualiza para maximizar el valor Q:

$$\nabla_{\theta_\mu} J \approx \frac{1}{N} \sum \nabla_a Q(s, a) \nabla_{\theta_\mu} \mu(s)$$

- ▶ (El actor aprende en la dirección que mejora Q).



COMO FUNCIONA DDPG?

- ✓ Generación de trayectorias
- ✓ Entrenamiento del Crítico
- ✓ Entrenamiento del Actor
- ✓ Actualización suave de redes objetivo

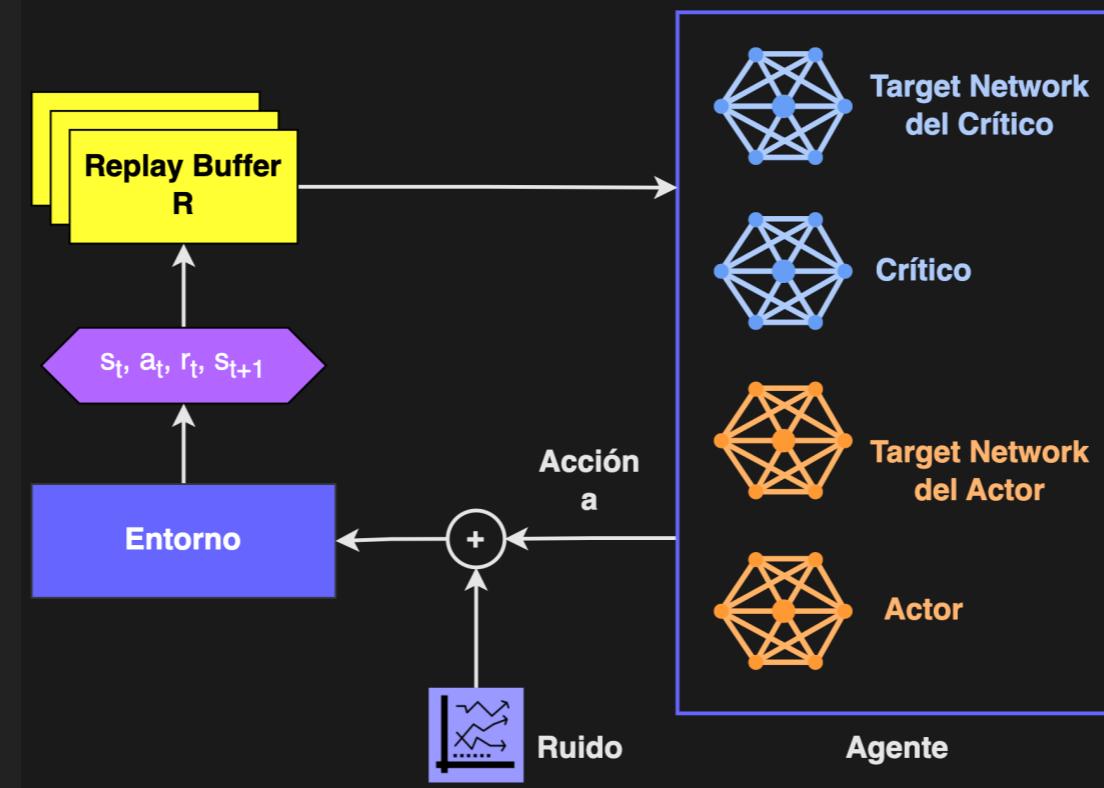


ACTUALIZACIÓN SUAVE DE REDES OBJETIVO

- ▶ Se usa Polyak Averaging (interpolación lenta o soft update):

$$\theta_{\text{target}} = \tau\theta + (1 - \tau)\theta_{\text{target}}$$

- ▶ (típicamente $\tau \approx 0.001$).



VENTAJAS DE DDPG

- ✓ Funciona bien en **espacios de acción continuos**.
- ✓ **Off-policy**, aprovecha datos viejos (replay buffer).
- ✓ Más estable que Policy Gradients puro (gracias al crítico).

Pero...

- ✓ Sensible a hiperparámetros (learning rate, ruido, τ).
- ✓ Puede ser **inestable** (requiere ajuste fino).



PSEUDOCÓDIGO

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
Initialize target network Q' and μ' 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:

$$\begin{aligned}\theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}\end{aligned}$$

end for
end for



PSEUDOCÓDIGO Y ARQUITECTURA DETALLADA

Algorithm 1 DDPG algorithm

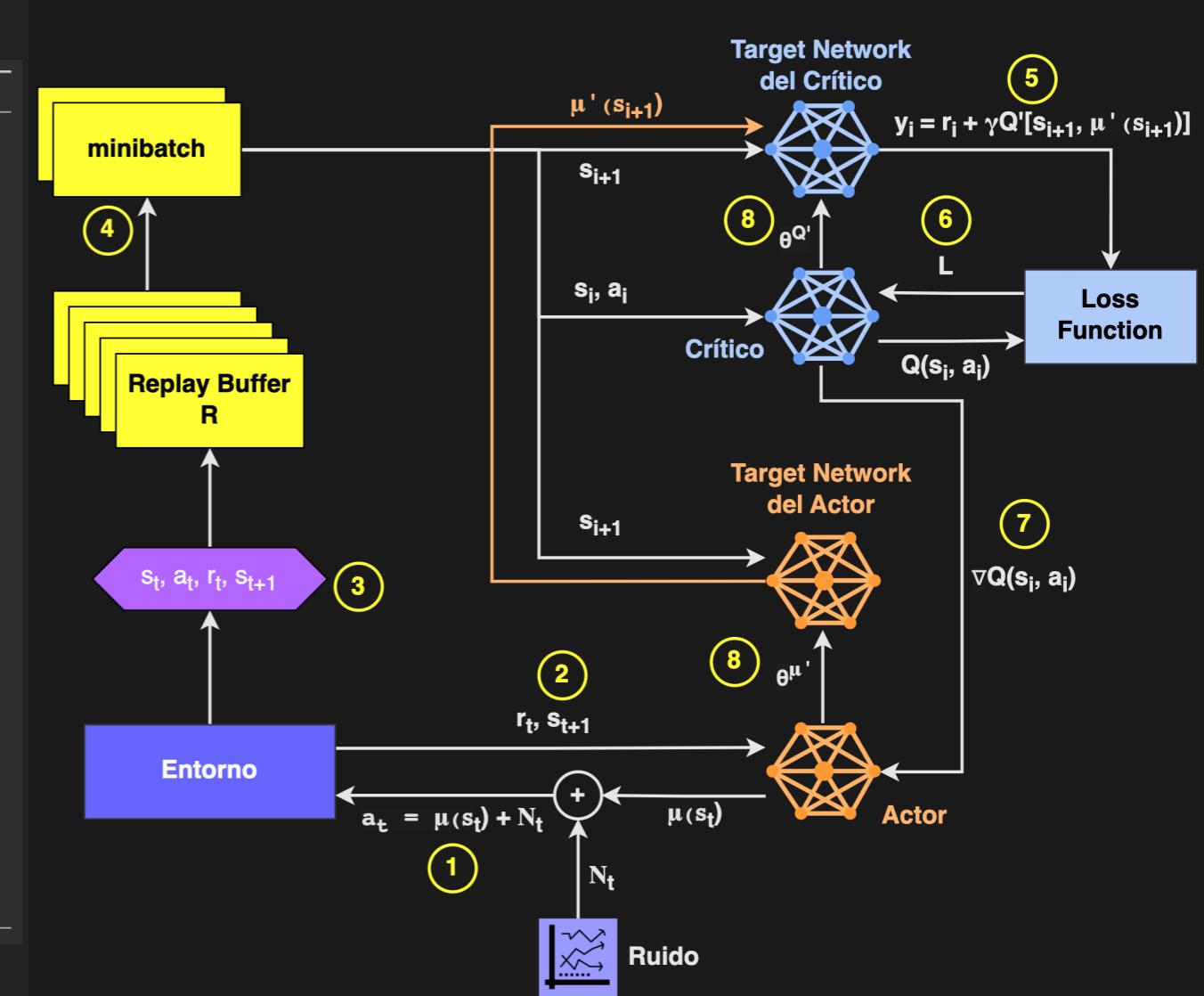
Randomly initialize critic network $Q(s, a | \theta^Q)$ and actor $\mu(s | \theta^\mu)$ with weights θ^Q and θ^μ .
 Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
 Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process \mathcal{N} for action exploration
 Receive initial observation state s_1
for t = 1, T **do**
 1 Select action $a_t = \mu(s_t | \theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
 2 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 3 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 4 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 5 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'})$
 6 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2$
 7 Update the actor policy using the sampled policy gradient:

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

8 Update the target networks:

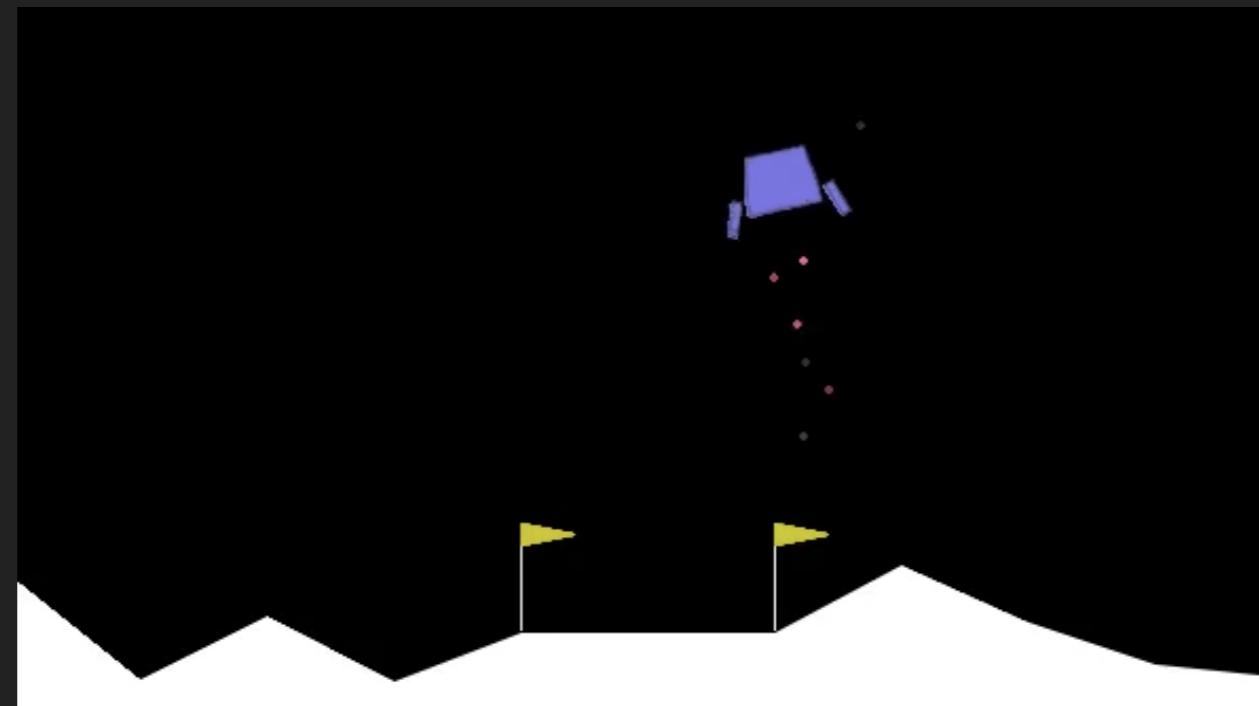
$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \end{aligned}$$

end for
end for



EJEMPLO

- ▶ Aprendizaje del alunizaje con LunarLanderContinuous-v3 de Gymnasium y con la biblioteca de agentes Stable Baseline 3.
- ▶ <https://github.com/aear-uba/ar2/tree/main/src/ddpg>



PARÁMETROS EN SB3

```
from stable_baselines3.common.noise import OrnsteinUhlenbeckActionNoise

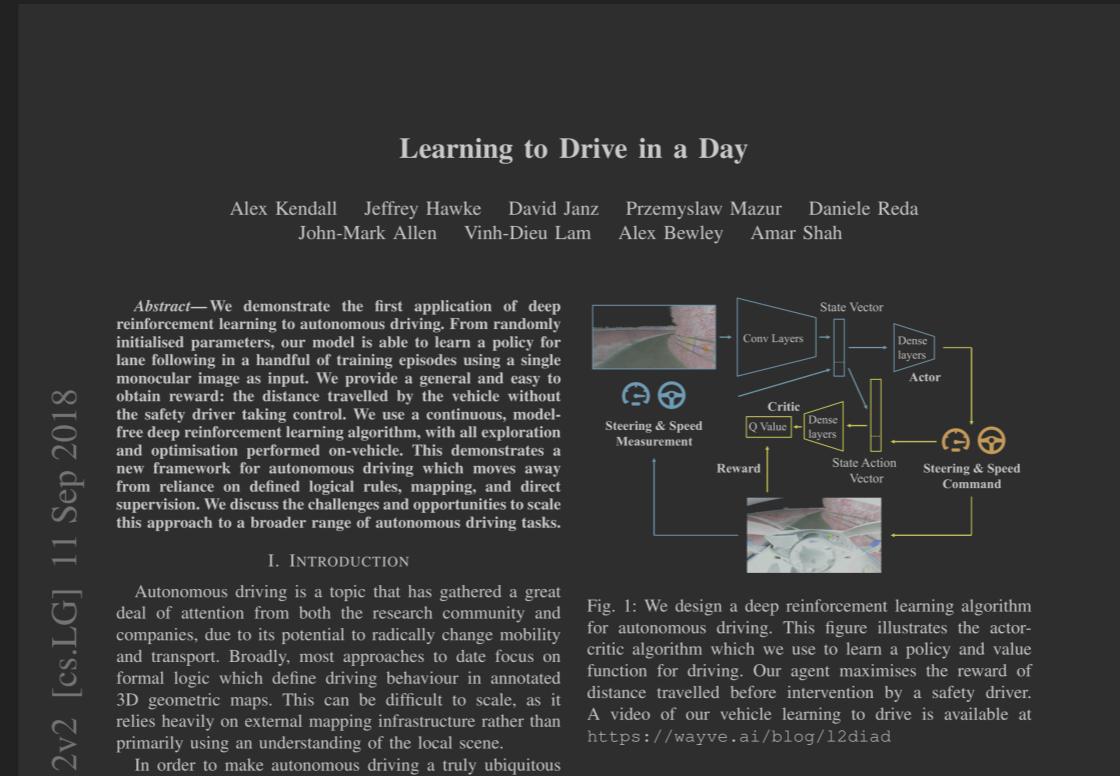
ou_noise = OrnsteinUhlenbeckActionNoise(
    mean=np.zeros(n_actions),
    sigma=0.1 * np.ones(n_actions),
    theta=0.15,
    dt=1e-2
)

model = DDPG(
    policy='MlpPolicy', # Tipo de política ('MultiInputPolicy' o 'CnnPolicy')
    env=train_env,
    action_noise=action_noise, # Ruido de acción para la exploración
    learning_rate=1e-3,
    buffer_size=200_000, # Tamaño máximo del replay buffer
    learning_starts=10_000, # Número de pasos a tomar antes de empezar a entrenar
    batch_size=128, # Tamaño del minibatch muestreado del buffer
    tau=0.005, # Coef. de actualización suave para las target networks
    gamma=0.99,
    gradient_steps=-1, # Núm. de pasos de gradiente a realizar por paso 1 0 -1
    train_freq=1, # Frecuencia de entrenamiento (en pasos de entorno)
    verbose=1,
    seed=42
)
```



EJEMPLO REAL (A NIVEL PROTOTIPO)

- ▶ Entrenamiento de un **coche autónomo** con DDPG.
- ▶ El trabajo se titula “Learning to drive in a day” (Kendall et al., 2018).
- ▶ <https://www.youtube.com/watch?v=eRwTbRtnT1I>
- ▶ <https://wayve.ai/thinking/learning-to-drive-in-a-day/>



ASPECTOS GENERALES

- ▶ Los parámetros son inicializados aleatoriamente.
- ▶ El modelo es capaz de aprender una política para el seguimiento de un carril en pocos episodios de entrenamiento.
- ▶ Utiliza una única imagen monocular como entrada.
- ▶ Recompensa: es proporcional a la distancia recorrida por el vehículo sin que el conductor de seguridad tome el control.



ASPECTOS GENERALES

- ▶ La mayoría de los enfoques se **centran en la lógica formal** que define el comportamiento de conducción en mapas geométricos 3D anotados.
- ▶ Difícil de escalar (dependencia de la infraestructura de mapeo externa en lugar de utilizar una **comprensión de la escena local**).



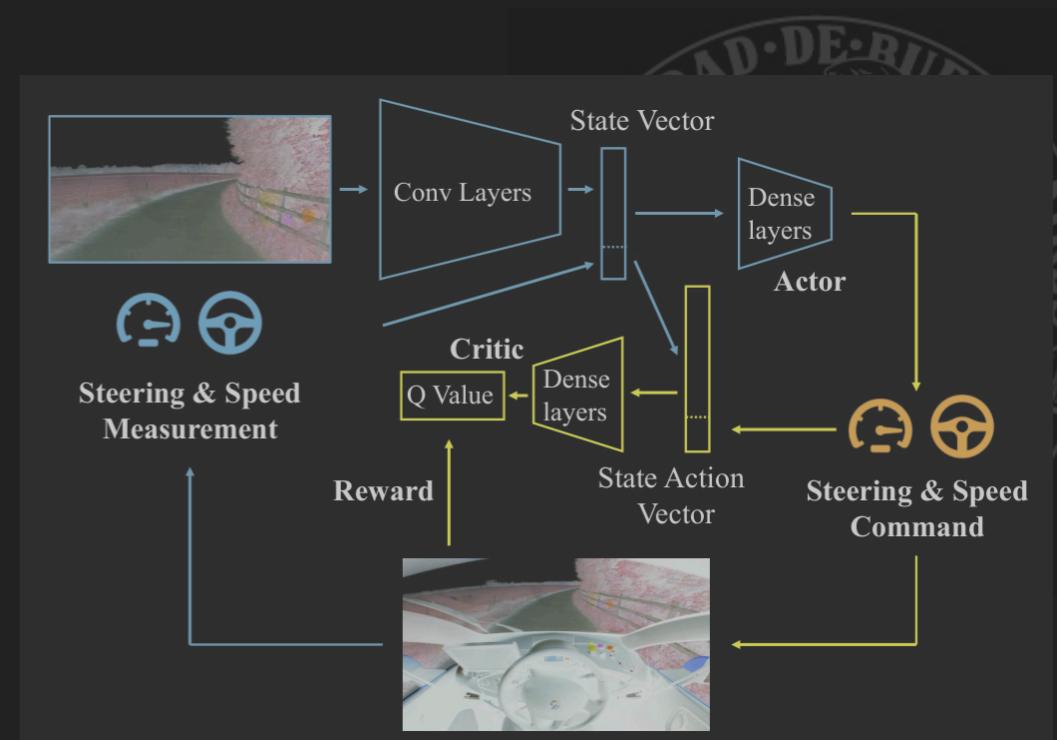
ASPECTOS GENERALES

- ▶ Algunos trabajos en esta área han demostrado que esto es posible en carreteras rurales utilizando:
 - ✓ GPS para localización aproximada
 - ✓ LIDAR para comprender la escena local



ARQUITECTURA

- ▶ Entradas al sistema: Imagen de la cámara, Medición de Dirección y Velocidad.
- ▶ Procesamiento:
 - ✓ La imagen pasa por Capas Convolucionales.
 - ✓ La salida de las capas convolucionales y la Medición de Dirección y Velocidad forman el "Vector de Estado".
- ▶ El Vector de Estado alimenta a:
 - ✓ Actor: tiene "Capas Densas" y produce un "Comando de Dirección y Velocidad".
 - ✓ Crítico: lee la "Acción" (del Actor), tiene "Capas Densas" y produce un "Valor Q". El Valor Q y el estado actual ayudan a formar un "Vector de Estado-Acción".
- ▶ La Recompensa se genera basada en el rendimiento.



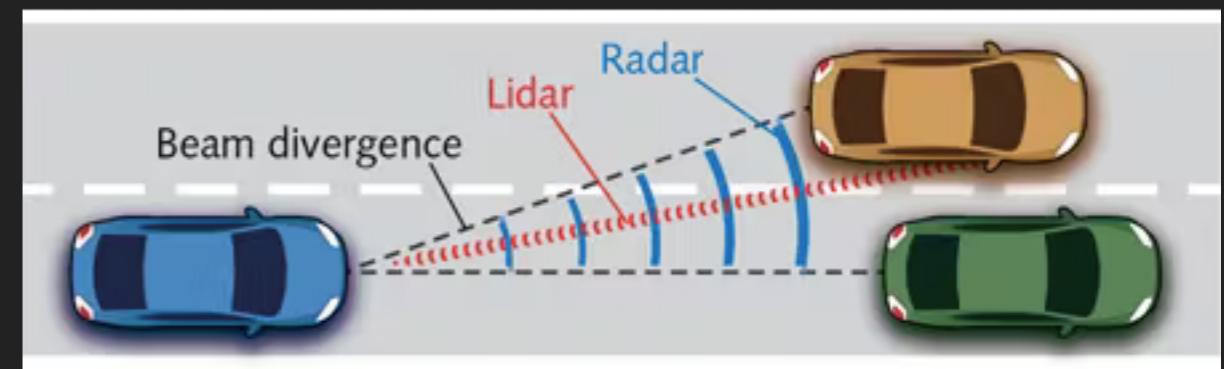
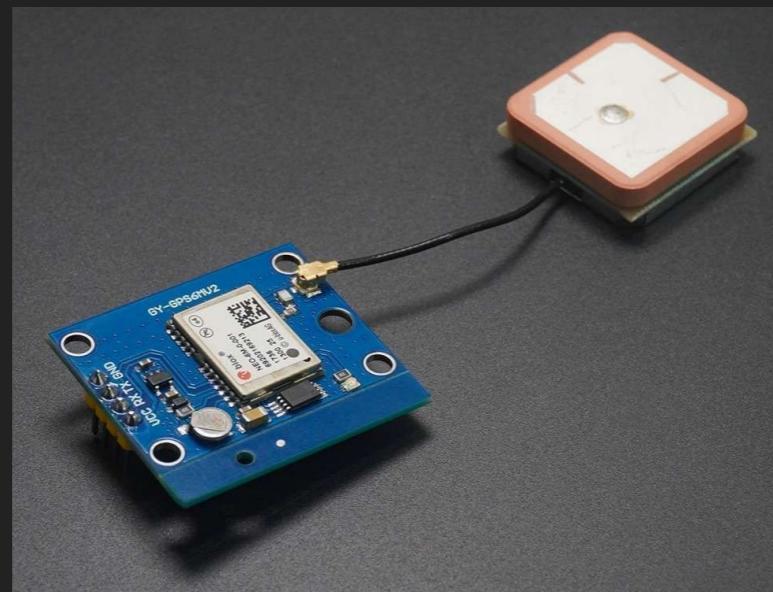
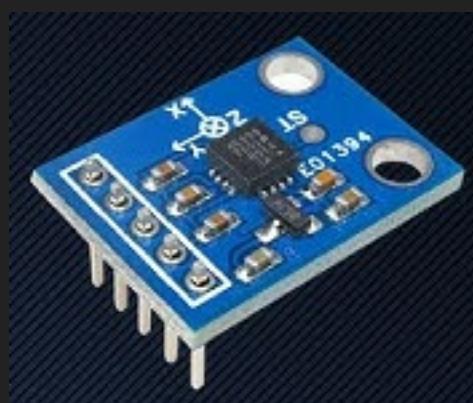
VARIABLES SIMULADAS VS. VARIABLES REALES

- ▶ RL se ha utilizado para entrenar agentes de conducción autónoma en videojuegos.
- ▶ En tales casos se simplifica el problema dado que carece de acceso a señales de recompensa de verdad fundamental (**ground truth**).
- ▶ Algunas variables no están disponibles en el mundo real, como el **ángulo del coche con respecto al carril**.



ESPACIO DE ESTADOS

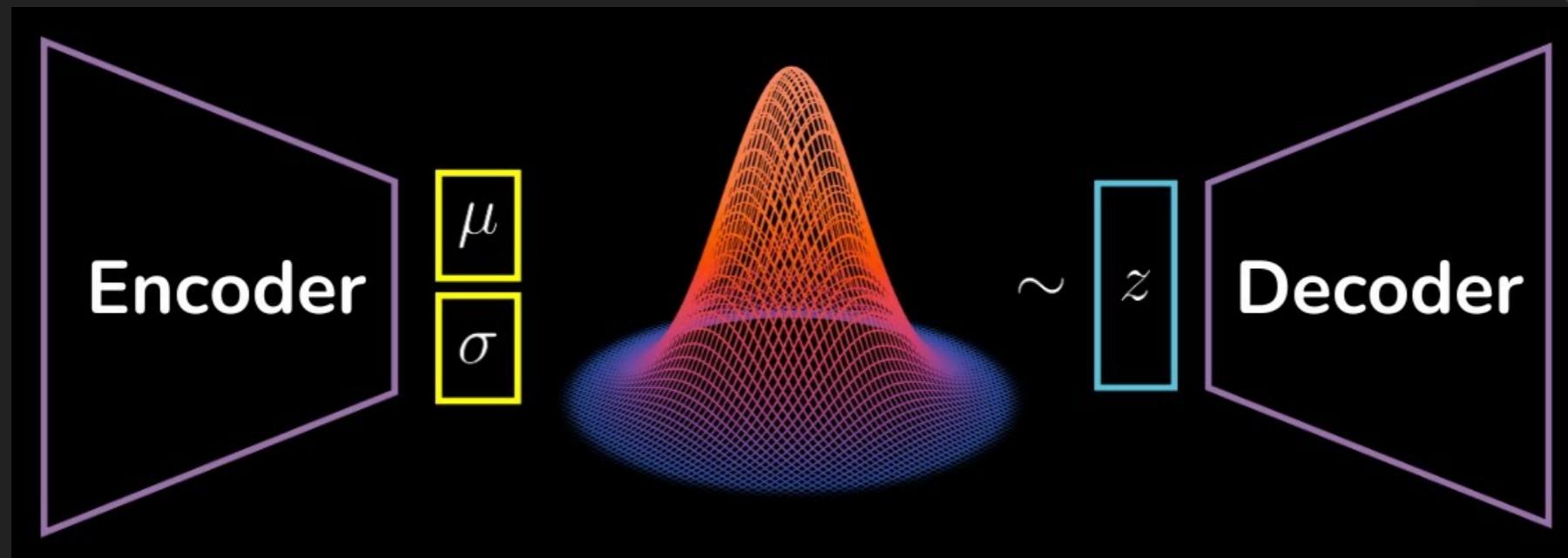
- ▶ El espacio de estados esta dado por las **observaciones** O_t que el algoritmo recibe en cada paso de tiempo.
- ▶ La mayoría de los sistemas de conducción autónoma cuentan con **sensores** para proporcionar **observaciones** sofisticadas (**LIDAR***, **IMUs**, unidades **GPS**, sensores de profundidad **IR**)
- ▶ Para tareas de conducción simples es suficiente usar una imagen de **cámara monocular**, junto con la **velocidad** y el **ángulo de dirección** observados del vehículo.



*Light detecting and ranging
Radio detecting and ranging

TRATAMIENTO DE LA IMAGEN

- ▶ Alternativa 1: La imagen en bruto podría alimentarse directamente al algoritmo RL a través de una serie de convoluciones.
- ▶ Alternativa 2: Representación pequeña y comprimida de la imagen, usando, por ejemplo, un **Autoencoder Variacional (VAE)**
- ▶ En este caso se entrenó el VAE en línea a partir de cinco episodios de exploración aleatorios, utilizando una pérdida KL y una pérdida de reconstrucción L2.



ESPACIO DE ACCIONES

- ▶ La opciones de control del acelerador son:
 - ✓ **Discreto**, encendido o apagado.
 - ✓ **Continuo**, en un rango isométrico a $[0, 1]$
- ▶ Una alternativa es reparametrizar el acelerador en términos de un **punto de ajuste de velocidad**, En lugar de que el algoritmo controle directamente cuánto pisa el acelerador, solo ordena: "Ir a 30 km/h".
- ▶ Luego, un "controlador clásico" (un sistema de control más simple y tradicional, no de aprendizaje profundo) se encargar de accionar el acelerador (y el freno si es necesario) para **alcanzar y mantener esa velocidad de 30 km/h**.
- ▶ Esto simplifica la tarea del algoritmo de RL, ya que solo tiene que pensar en la velocidad deseada, no en los detalles de cómo lograrla con el pedal.
- ▶ Se usó un **espacio de acciones bidimensional**:
 - ✓ ángulo de dirección en el rango $[-1, 1]$.
 - ✓ punto de ajuste de velocidad en km/h.



ESPACIO DE ACCIONES BIDIMENSIONAL

► El algoritmo toma dos decisiones simultáneamente en cada paso:

► Ángulo de dirección (un valor continuo en el rango $[-1, 1]$).

✓ -1 significa girar el volante completamente a la izquierda.

✓ +1 significa girar el volante completamente a la derecha.

✓ 0 significa mantener el volante recto.

✓ Valores intermedios (-0.5, 0.2, etc.) significan giros parciales.

► Punto de ajuste de velocidad en km/h: El algoritmo decide a qué velocidad quiere que vaya el coche (por ejemplo, 15 km/h, 22 km/h). Luego, un sistema subyacente (un controlador clásico) se encarga de que el coche alcance esa velocidad.

► El algoritmo DDPG controla dos cosas de forma continua:

✓ Cuánto y hacia dónde girar el volante.

✓ A qué velocidad quiere que el coche se mueva.



ARQUITECTURA DE ENTRENAMIENTO BASADA EN TAREAS

- ▶ **Tareas del conductor:** entrenar, probar, deshacer y terminar.
- ▶ **Entrenar:** se agrega ruido a la salida del modelo y el cual se optimiza.
- ▶ **Probar:** ejecuta directamente las acciones de salida del modelo.
- ▶ Durante los primeros episodios se omitió la optimización para favorecer la exploración del espacio de estados.
- ▶ Luego, se continuó el experimento hasta que la recompensa de prueba dejó de aumentar.
- ▶ Cada episodio se ejecuta hasta que el sistema detecta que ha perdido la automatización (es decir, el conductor intervino).

```
1: while True do
2:   Request task
3:   Waiting for environment reset
4:   if task is train then
5:     Run episode with noisy policy
6:     if exploration time is over then
7:       Optimise model
8:     end if
9:   else if task is test then
10:    Run episode with optimal policy
11:   else if task is undo then
12:     Revert previous train/test task
13:   else if task is done then
14:     Exit experiment
15:   end if
16: end while
```



ARQUITECTURA DE ENTRENAMIENTO BASADA EN TAREAS

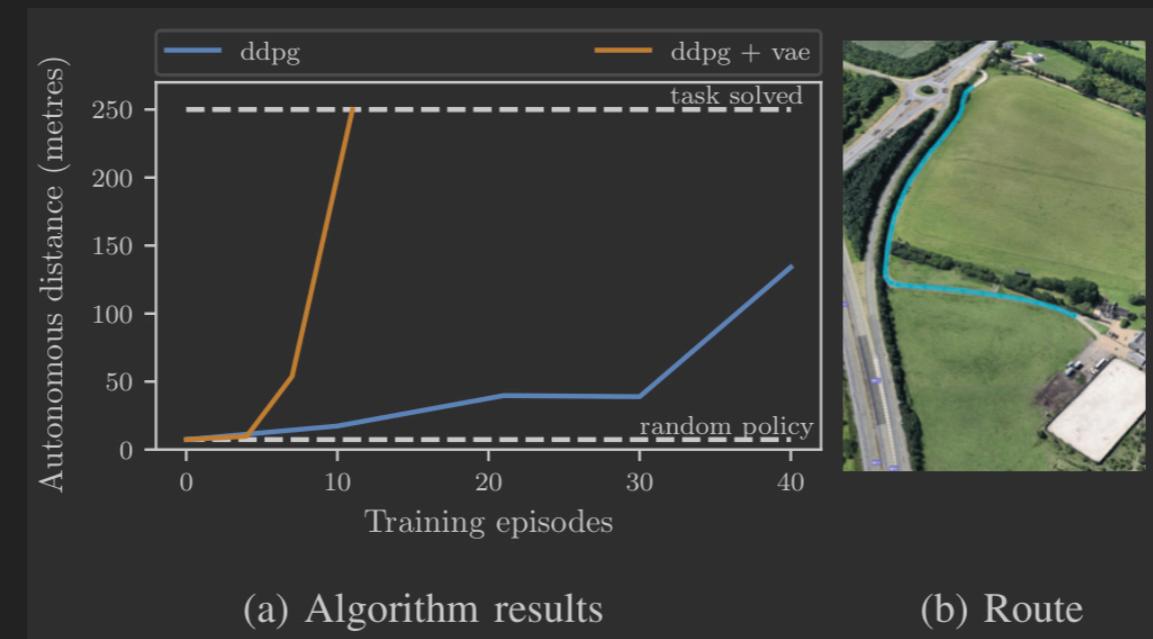
- ▶ **Terminar:** El sistema puede **terminar** un episodio por una variedad de razones válidas en la conducción normal.
- ✓ Estos episodios no pueden considerarse para fines de entrenamiento.
- ▶ **Deshacer:** permite **deshacer** el episodio y restaurar el modelo como estaba antes de ejecutar ese episodio.
- ✓ Ejemplo: encontrar a otros conductores que buscan usar la carretera que se utiliza como entorno.

```
1: while True do
2:     Request task
3:     Waiting for environment reset
4:     if task is train then
5:         Run episode with noisy policy
6:         if exploration time is over then
7:             Optimise model
8:         end if
9:     else if task is test then
10:        Run episode with optimal policy
11:    else if task is undo then
12:        Revert previous train/test task
13:    else if task is done then
14:        Exit experiment
15:    end if
16: end while
```



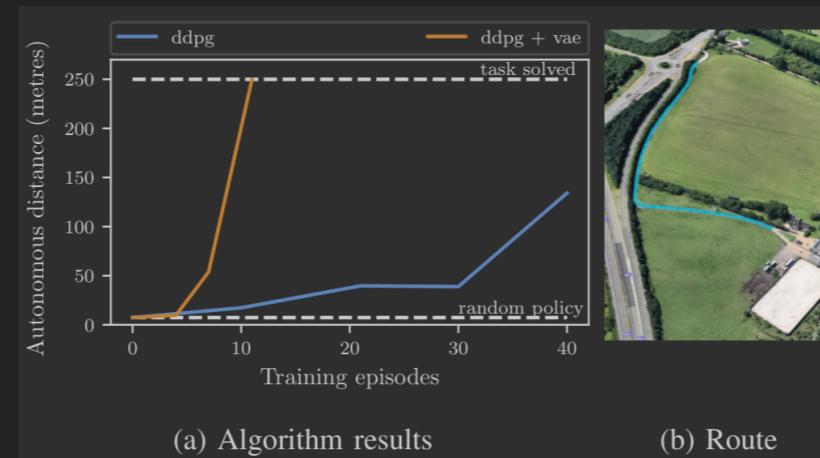
RESULTADOS

- ▶ Gráfico (a) Resultados del algoritmo: Muestra "Distancia autónoma (metros)" en el eje Y y "Episodios de entrenamiento" en el eje X.
- ▶ Hay dos líneas: "ddpg" y "ddpg + vae".
- ▶ La línea "ddpg + vae" alcanza la línea "tarea resuelta" (task solved) (250m) mucho más rápido (alrededor de 11 episodios) que la línea "ddpg" (alrededor de 35 episodios).
- ▶ Una línea de "política aleatoria" (random policy) muestra un rendimiento muy bajo.
- ▶ Gráfico (b) Ruta: Muestra una vista aérea de una carretera corta y recta.



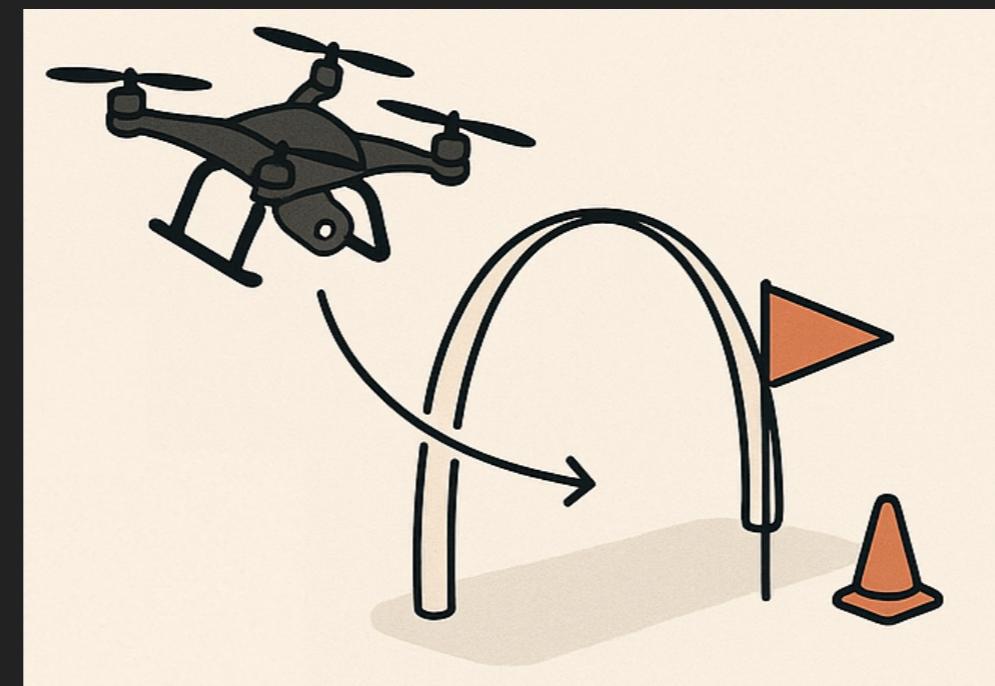
TRABAJO FUTURO Y CONCLUSIONES

- ▶ Existen dos os áreas que podrían mejorar RL en la conducción autónoma real:
 - ✓ aprendizaje semi-supervisado
 - ✓ transferencia de dominio
- ▶ Los modelos RL podrían beneficiarse de los datos de conducción capturados desde cámaras de tablero en **vehículos cotidianos**. Estos podrían usarse para pre-entrenar el autoencoder de imágenes.
- ▶ La **transferencia de dominio**, podría permitir crear simulaciones lo suficientemente convincentes como para que los datos de estas puedan usarse para entrenar una política que pueda transferirse directamente a un coche real.
- ▶ Se demostró la facilidad con la que el RL puede aplicarse a la conducción autónoma.



OTRAS APLICACIONES DE DDPG

- ▶ drones y vehículos autónomos
- ▶ robótica
- ▶ optimización energética (edificios, datacenters, producción de energía, eólica, solar,...)



REFERENCIAS BIBLIOGRÁFICAS Y WEB (I)

- ▶ Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014, January). Deterministic policy gradient algorithms. In International conference on machine learning (pp. 387-395). Pmlr.
- ▶ Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y. & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
- ▶ G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," Phys. Rev., vol. 36, pp. 823-841, Sep 1930.
- ▶ Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J. M., ... & Shah, A. (2019, May). Learning to drive in a day. In 2019 international conference on robotics and automation (ICRA) (pp. 8248-8254). IEEE.
- ▶ <https://wayve.ai/thinking/learning-to-drive-in-a-day/>
- ▶ <https://www.washingtonpost.com/technology/2019/04/05/watch-self-driving-car-learn-navigate-narrow-european-streets-like-human-driver/>

