1

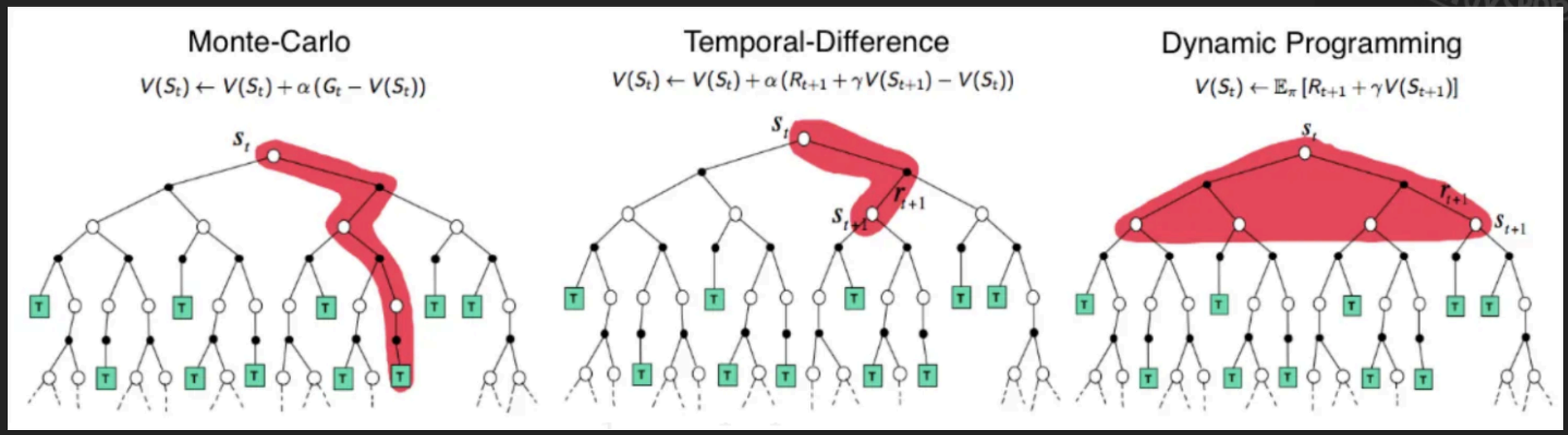
VARIANTES DE ACTOR-CRITIC

VARIANTES

- ✓ A2C (2016)
- ✓ A3C (2016)
- ✓ SAC (2018)
- ✓ DDPG (2016)
- ✓ TRPO (2015)
- ✓ PPO (2017)
- ✓ TD3 (2018)
- ✓ IMPALA (2018)

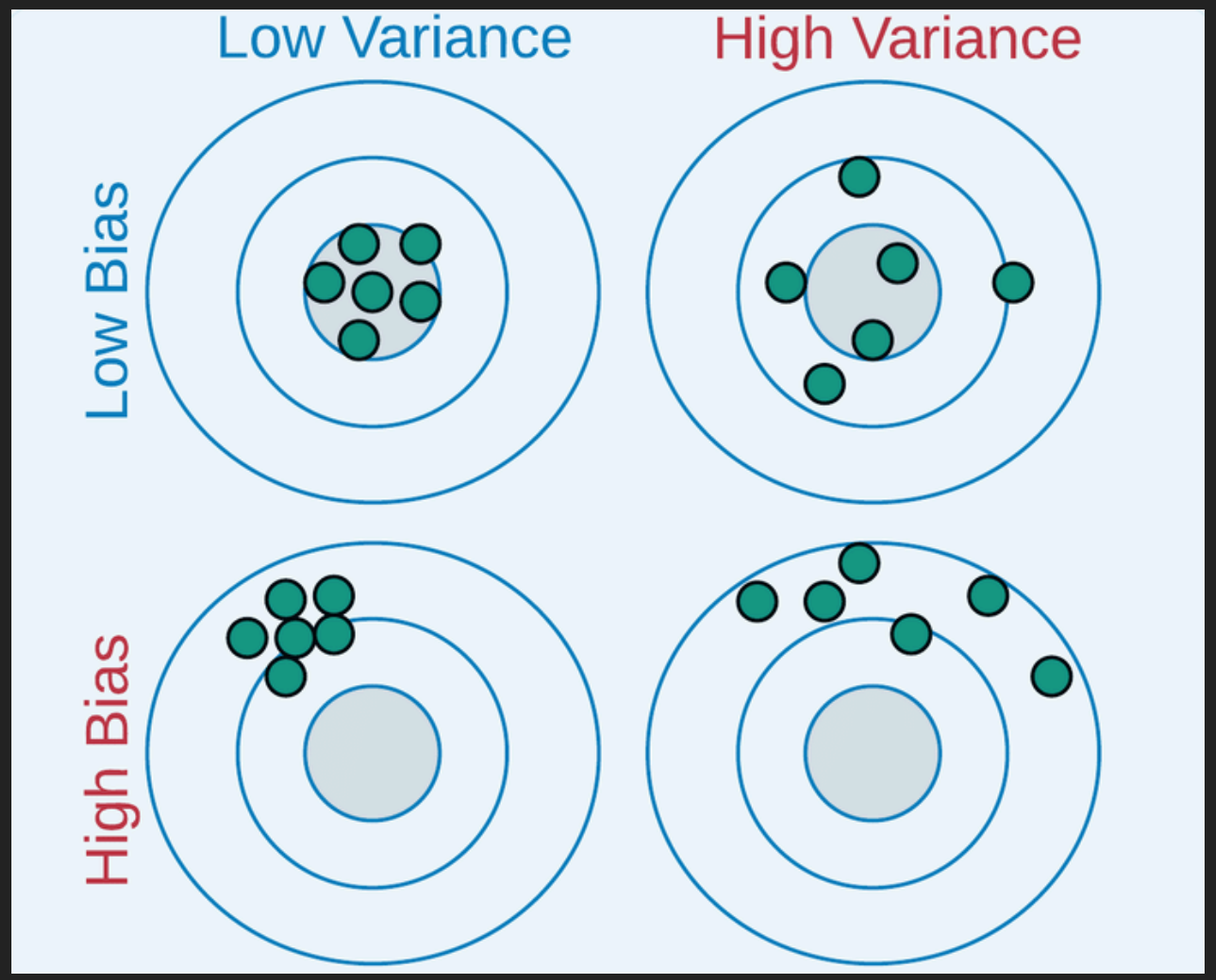
ESTIMACIÓN DE V(S)

- Monte Carlo: espera hasta el final de un episodio para obtener el retorno real G_t. Luego, usa este retorno completo y real para actualizar el valor de todos los estados visitados en ese episodio (S_t incluido).
- TD: actualiza la estimación del valor de St después de cada paso, sin esperar al final del episodio. Lo hace usando la recompensa real inmediata R_{t+1} y la propia estimación actual del valor del siguiente estado $V(S_{t+1})$ (bootstrapping).
- Programación Dinámica: requiere un modelo completo del entorno (es decir, conocer las probabilidades de transición P(s'|s, a) y las recompensas esperadas R(s, a, s')). Actualiza el valor de un estado basándose en los valores calculados (usando el modelo) de todos los posibles estados sucesores.



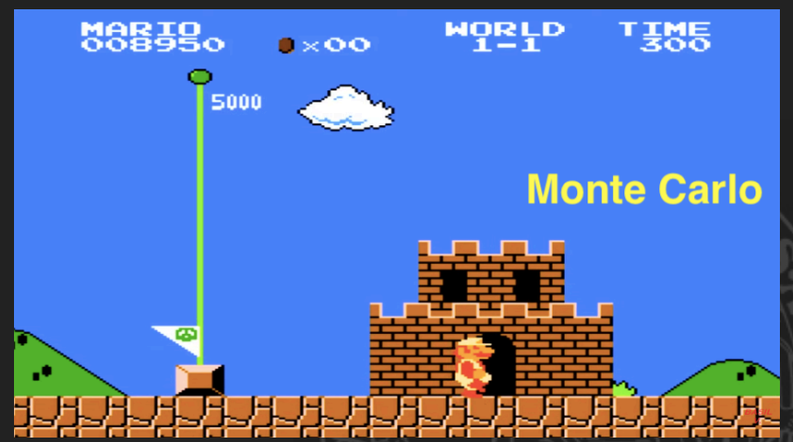
VARIANZA Y BIAS

- La varianza es medir cuánto las estimaciones se parecen entre sí.
- El bias es medir cuan alejada esta la estimación del valor real.



RETORNOS

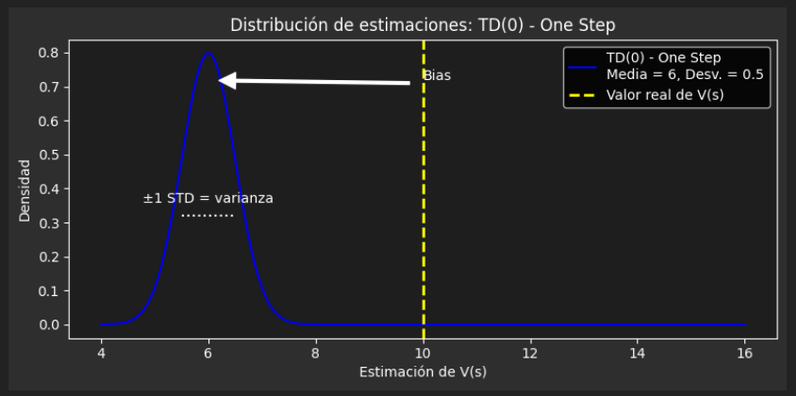
- Los retornos para estimar V(s) son: one-step, n-step, Monte Carlo (“T-step”).

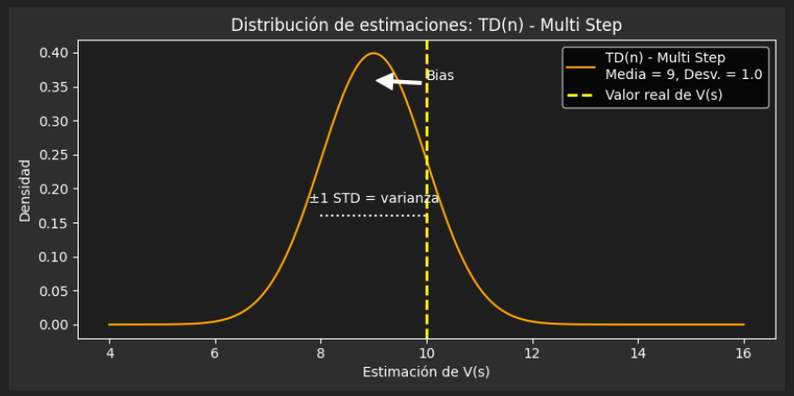


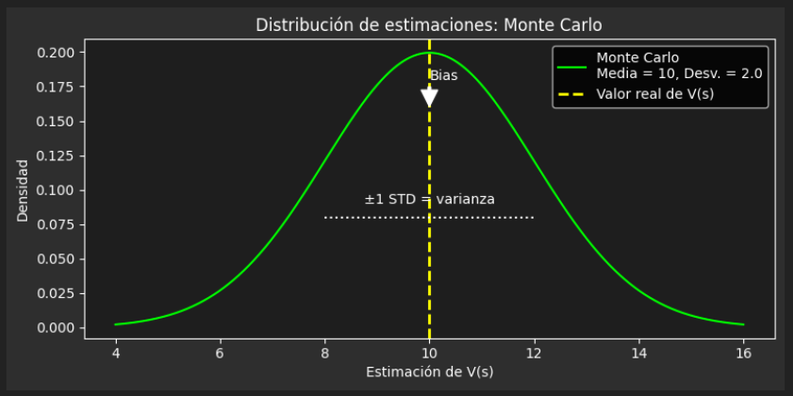
- Diferencia Temporal
- Actor-Critic

- Actor-Critic
- Advantage Actor-Critic (A2C)

- Monte Carlo







$$V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$$

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} +$$

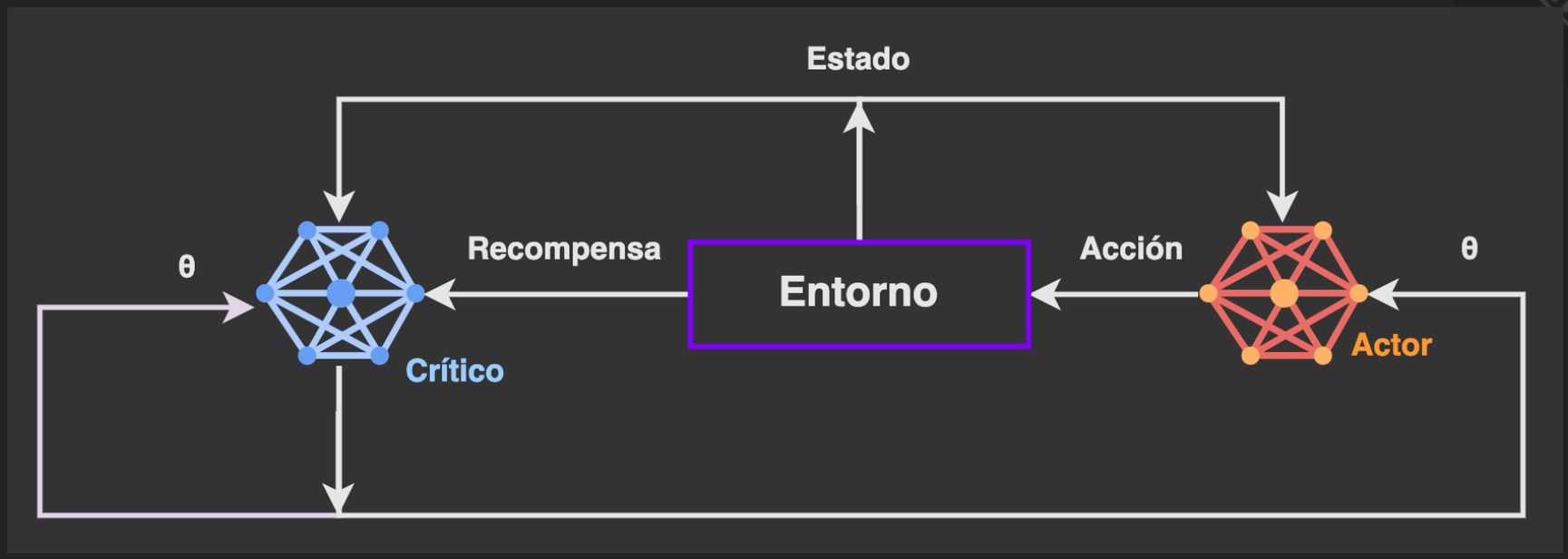
$$V(s) \leftarrow V(s) + \alpha\left[G_t - V(s)\right]$$

ACTOR-CRITIC

- Objetivo: encontrar una política (una forma de actuar) óptima (π*) para el agente.
- Componentes:

✓ Actor

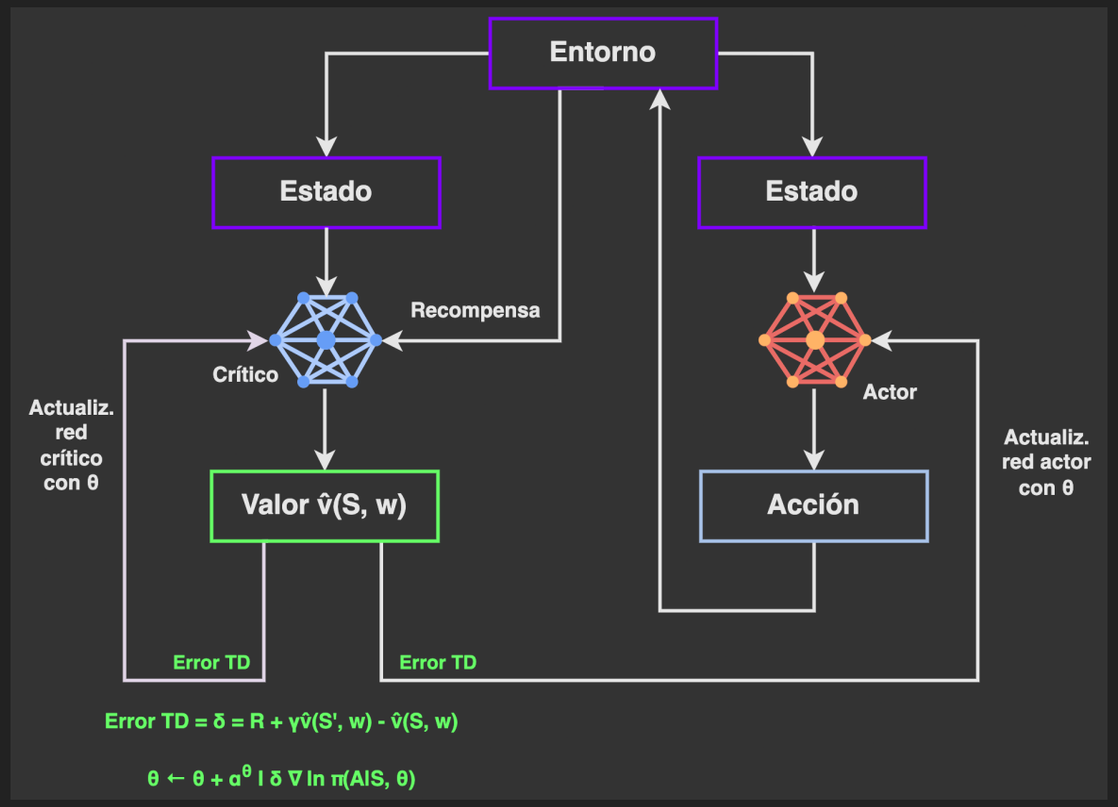
✓ Crítico



ACTOR-CRITIC

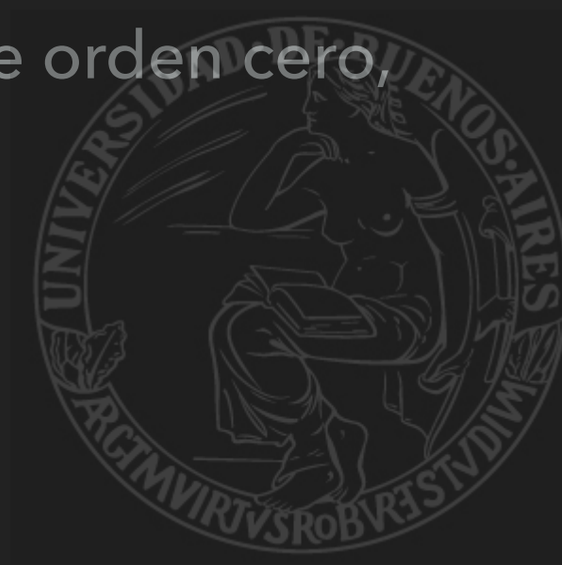
- ✓ Actor: Red neuronal que decide qué acción tomar en un estado dado. Está parametrizada por su pesos θ. Su objetivo es mejorar la política π(a|s, θ) para que elija acciones que lleven a mayores recompensas acumuladas.

- ✓ Crítico: Red neuronal que evalúa qué tan buena es una acción o un estado. Estima la función de valor de estado v̂(s, w), que predice la recompensa acumulada esperada a partir de un estado s. Está parametrizado por w. Ayuda al Actor diciéndole si las acciones tomadas fueron mejores o peores de lo esperado.



ACTOR-CRITIC

- One-step: Las actualizaciones se basan en la información del siguiente paso inmediato (la recompensa R y el valor estimado del siguiente estado S').
- Esto se conoce como aprendizaje por Diferencia Temporal de orden cero, TD(0). Si fuese n-step se lo suele denominar TD(λ).

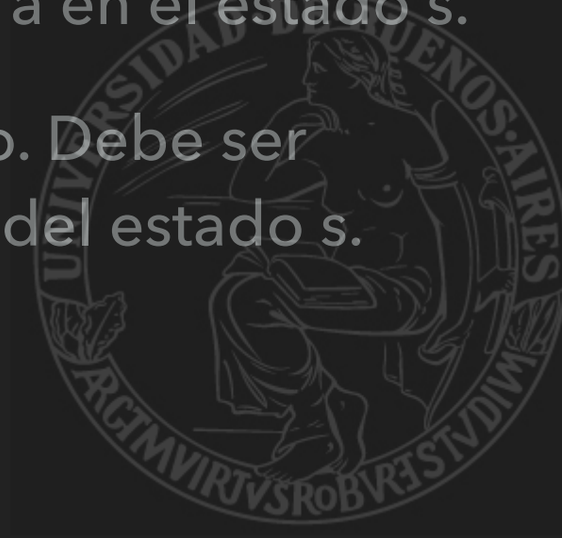


One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- Inputs:
  - ✓ π(a|s, θ), política del Actor. Debe ser diferenciable respecto a θ para poder calcular gradientes. Define la probabilidad de tomar la acción a en el estado s.
  - ✓ v̂(s, w): es la función de valor de estado estimada por el Crítico. Debe ser diferenciable respecto a w. Estima el valor (retorno esperado) del estado s.



One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

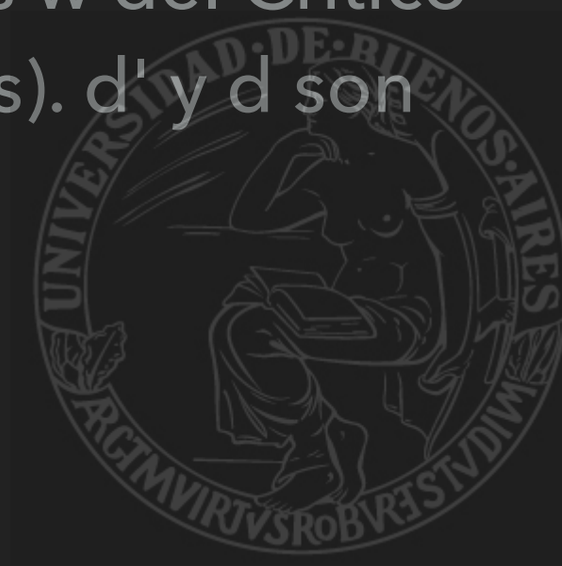
- Parameters:
  - ✓ $\alpha_\theta > 0$, $\alpha_w > 0$: Tasas de aprendizaje (o tamaños de paso) para el Actor (θ) y el Crítico (w), respectivamente. Controlan cuanto se actualizan los parámetros en cada paso.

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- Initialize (Inicialización):
  - ✓ Se inicializan los parámetros θ del Actor y los pesos w del Crítico (a menudo con ceros o valores aleatorios pequeños). d' y d son las dimensiones de estos vectores de parámetros.

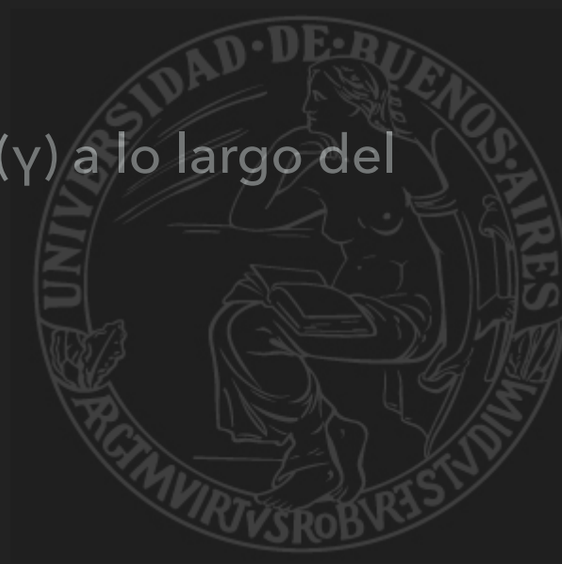


One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- Initialize S (first state of episode):
  - ✓ Se obtiene el estado inicial S para el nuevo episodio.
- I ← 1:
  - ✓ Se inicializa una variable I. Esta variable acumula el factor de descuento (γ) a lo largo del episodio (I representará γ^t en el paso t). Empieza en 1 porque γ^0 = 1.



One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- Loop while S is not terminal (for each time step) = Bucle interno (por cada paso de tiempo dentro del episodio):
  - ✓ Este bucle se ejecuta mientras el agente no haya llegado a un estado final.

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- $A \sim \pi(\cdot|S, \theta)$:
  - ✓ El Actor elige una acción A basándose en la política actual π para el estado actual S. La acción se muestra (samplea) de la distribución de probabilidad definida por la política.

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- Take action A, observe S', R:

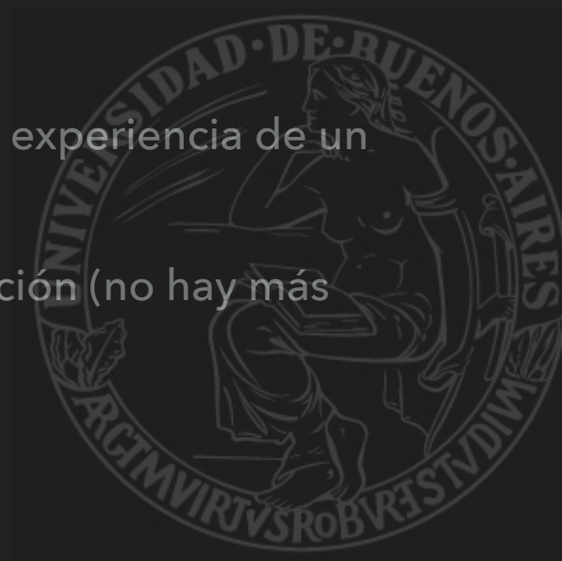
  ✓ El agente ejecuta la acción A en el entorno. Como resultado, observa el siguiente estado S' y recibe una recompensa inmediata R.

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- $\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$: Cálculo del Error TD (Temporal Difference) (sirve para calcular los pesos w del Crítico).
- $R + \gamma \hat{v}(S', w)$: Es el objetivo TD (TD target). Es una estimación mejorada del valor del estado S, basada en la recompensa real R obtenida y el valor estimado del siguiente estado S' (descontado por γ).
- $\hat{v}(S, w)$: Es la estimación actual del valor del estado S según el Crítico.
- δ: El Error TD. Mide la diferencia entre el objetivo TD (lo que debería ser el valor de S según la experiencia de un paso) y la estimación actual del Crítico.
- (if S' is terminal, then $\hat{v}(S', w) \doteq 0$): Si el siguiente estado es terminal, su valor es cero por definición (no hay más recompensas futuras).

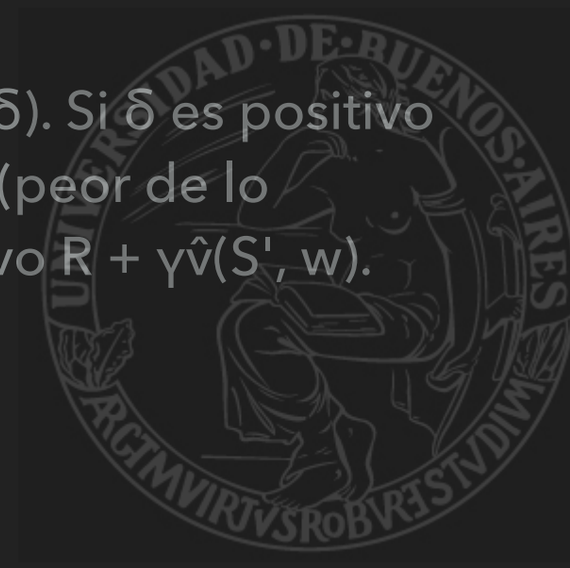


One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- $w \leftarrow w + \alpha w\, \delta\, \nabla \hat{v}(S, w)$: Actualización de los pesos w del Crítico
- $\nabla \hat{v}(S, w)$: Es el gradiente de la función de valor estimada con respecto a sus pesos w, evaluado en el estado actual S.
- La actualización mueve los pesos w en la dirección que reduce el error TD (δ). Si δ es positivo (el resultado fue mejor de lo esperado), $\hat{v}(S, w)$ aumentará. Si δ es negativo (peor de lo esperado), $\hat{v}(S, w)$ disminuirá. El objetivo es que $\hat{v}(S, w)$ se acerque al objetivo $R + \gamma \hat{v}(S', w)$.

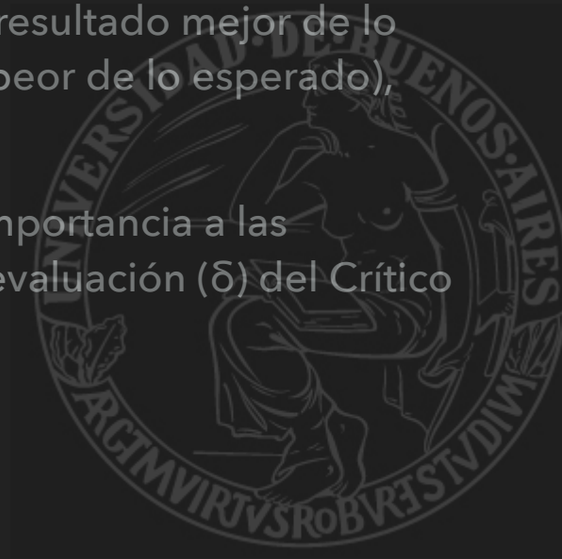


One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- θ ← θ + αθ I δ ∇lnπ(A|S, θ): Actualización del Actor
- ∇lnπ(A|S, θ): Es el gradiente del logaritmo de la probabilidad de la política para la acción específica A que se tomó en el estado S. Este gradiente apunta en la dirección que aumenta la probabilidad de tomar esa acción A en el estado S.
- δ: El error TD calculado por el Crítico actúa como una señal de ventaja. Si δ > 0 (la acción llevó a un resultado mejor de lo esperado), la actualización aumentará la probabilidad de tomar la acción A en el estado S. Si δ < 0 (peor de lo esperado), disminuirá esa probabilidad.
- I: El factor de descuento acumulado (γ^t). Pondera la actualización, dando potencialmente menos importancia a las decisiones tomadas más tarde en el episodio (aunque esto depende del valor de γ). El Actor usa la evaluación (δ) del Crítico para saber en qué dirección ajustar su política.



One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- $I \leftarrow \gamma^I$:
  - ✓ Se actualiza el factor de descuento acumulado para el siguiente paso de tiempo ($\gamma^{t+1} = \gamma * \gamma^t$).

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

ACTOR-CRITIC

- $S \leftarrow S'$:
  - ✓ El estado actual se actualiza al siguiente estado observado.

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

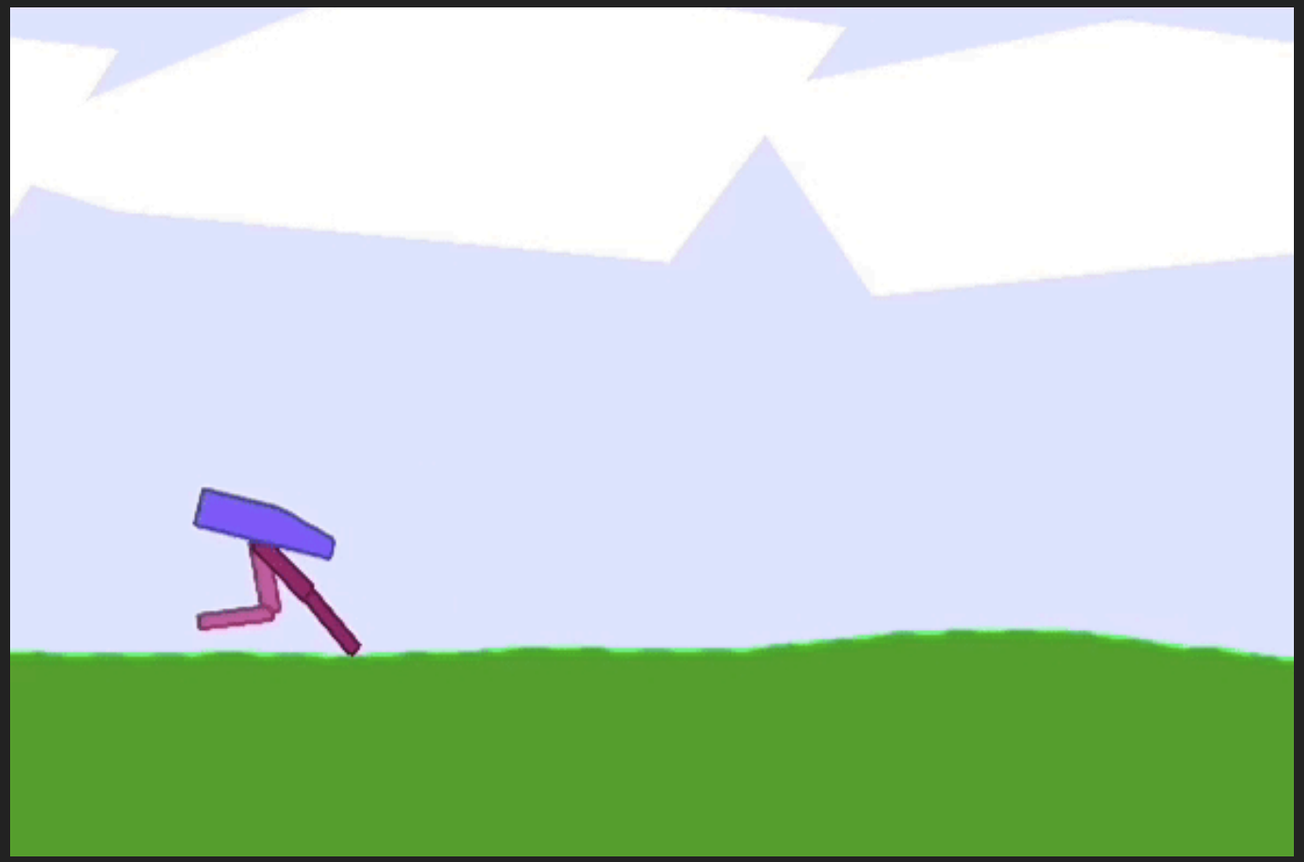
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

CUANDO USAR ACTOR-CRITIC?

- Cuando se maneja espacios de estado continuos (las RNA como aproximadoras de función toman vectores de números reales tales como posiciones, velocidades, ángulos, etc., como entrada).
- La variante de Actor-Critic A2C también se adapta a estados discretos.
- El Actor puede manejar acciones continuas o los parámetros de una distribución de probabilidad de acciones continuas.
- Cuando la exploración es crítica (ej: entornos con recompensas escasas).
- Cuando la estabilidad es más importante que el rendimiento. Es menos eficiente que sus variantes pero es más estable en algunos casos porque evita complejidades como Replay buffers y Target networks.

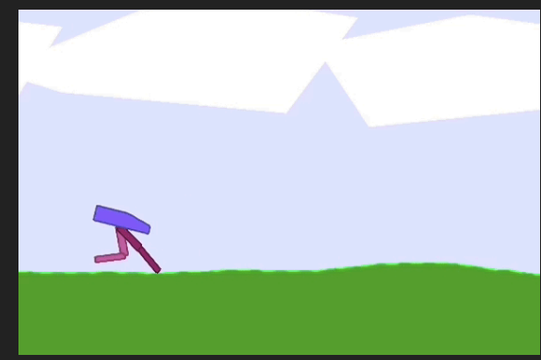
ACTOR-CRITIC (EJEMPLO)

- Actor-Critic one-step aplicado a BipedalWalker de Gymnasium.



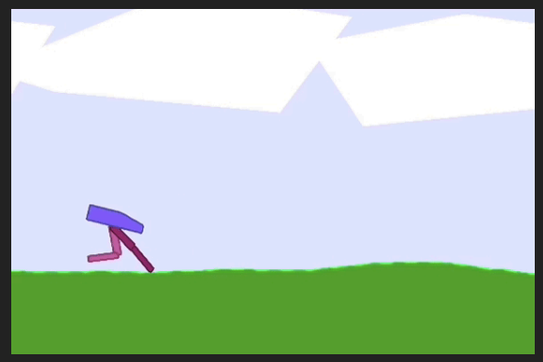
CARACTERÍSTICAS

- Objetivo: aprender a caminar hacia adelante sin caerse, superando obstáculos en un terreno irregular.
- Espacio de estados (usa Lidar, Laser Distance Sensor, que mide la forma del terreno):
  - ✓Posición y ángulo del torso.
  - ✓Velocidades articulares (hombros y rodillas).
  - ✓Contacto con el suelo (para detectar caídas).
- Espacio de acciones (tiene 4 articulaciones, 2 en cada pierna: cadera y rodilla):
  - ✓Acciones son valores continuos en [-1, 1], donde:
  - ✓1 = máxima fuerza en una dirección.
  - ✓-1 = máxima fuerza en la dirección opuesta.



CARACTERÍSTICAS

- Recompensas y Penalizaciones
  - ✓Recompensa positiva: Avanzar hacia adelante sin caerse.
  - ✓Caer (ángulo del torso fuera de límites) -100.
  - ✓Consumo excesivo de energía (fuerza aplicada a los motores).
  - ✓Recompensa total = distancia recorrida - penalización por energía.
- El episodio termina si:
  - ✓El torso toca el suelo (ángulo > 30°).
  - ✓Se alcanza el límite de pasos (1600).



PSEUDOCÓDIGO

Crear Actor

Crear Crítico

Bucle (por cada episodio)

    Inicializar estado

    Inicializar recompensa_acumulada

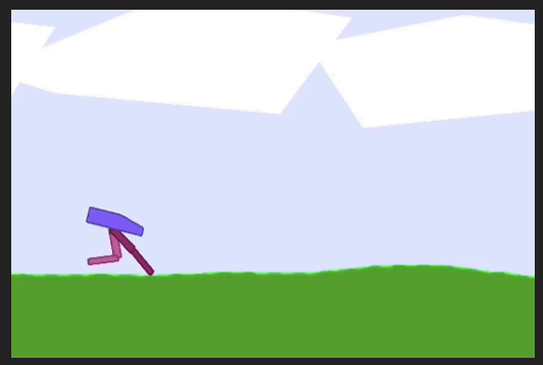
    Bucle (por cada paso)

Acción <– actor(estado)

Nuevo_estado, recompensa <– entorno(accion)

Recompensa_acumulada <– Recompensa_acumulada + recompensa

Valor <– critico(estado)



ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC (A3C)

- Este método publicado con el título “Métodos asincrónicos para aprendizaje por refuerzo profundo” (Mnih et al., 2016).

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU. Furthermore, we show that asynchronous actor-critic succeeds on a wide variety of continuous motor control problems as well as on a new task of navigating random 3D mazes using a visual input.

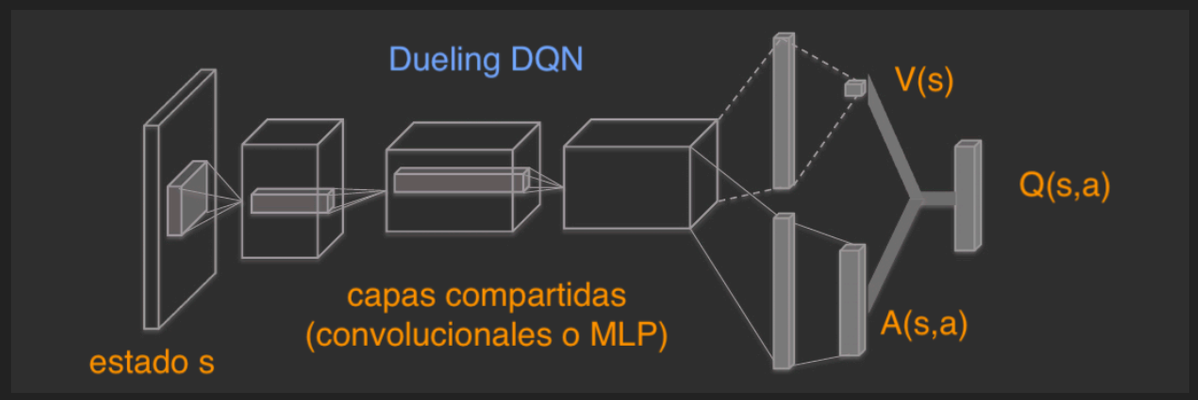
line RL updates are strongly correlated. By storing the agent’s data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to off-policy reinforcement learning algorithms.

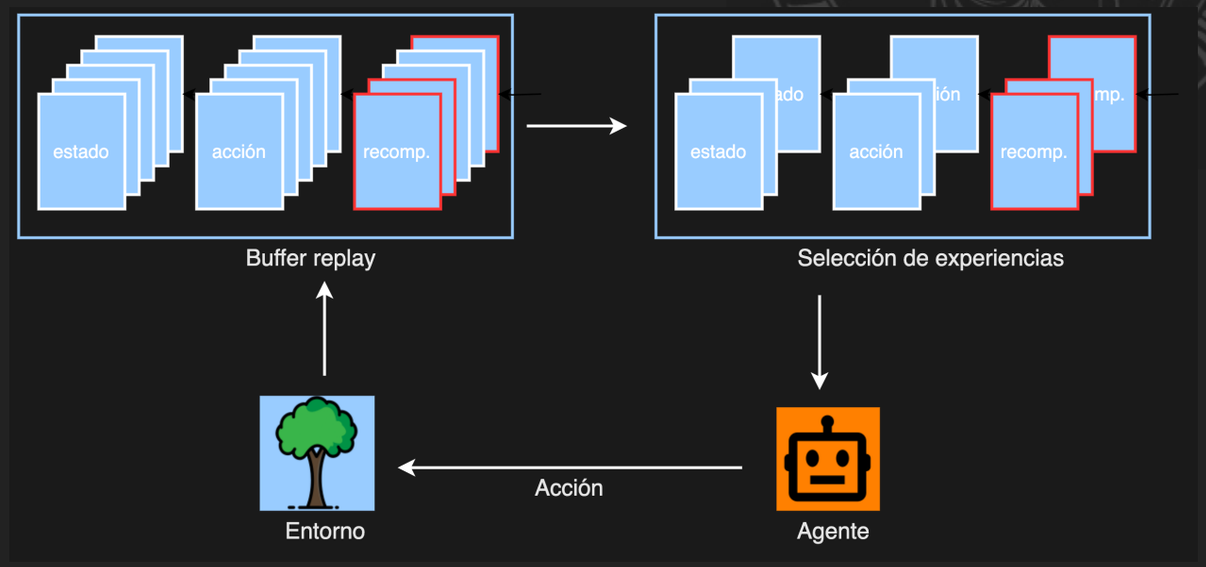
Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

In this paper we provide a very different paradigm for deep reinforcement learning. Instead of experience replay, we asynchronously execute multiple agents in parallel, on multiple instances of the environment. This parallelism also decorrelates the agents’ data into a more stationary process,

COMO FUNCIONA A3C?

- A3C se basa (parcialmente) en la variante Dueling DQN:
  - ✓ Usa la función de ventaja A(s, a).
  - ✓ Reemplaza Replay Buffer por una arquitectura de varios actores en lugar de uno.



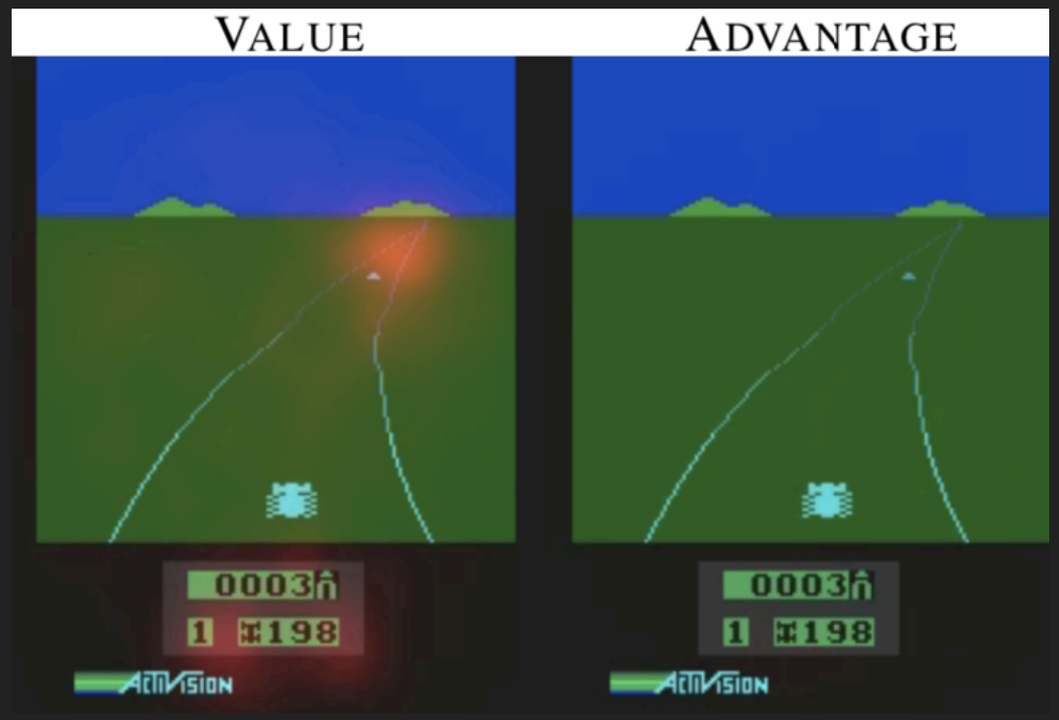


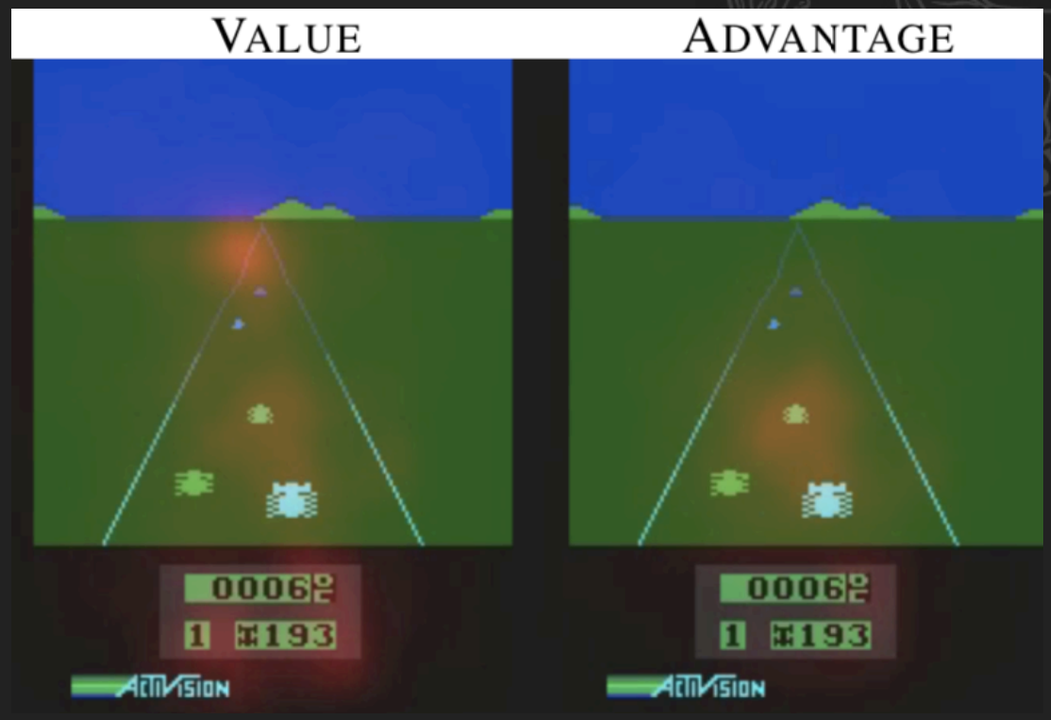
FUNCIÓN DE VENTAJA A(S, A)

- En Dueling DQN la solución consistía en desacoplar la función de valor Q en dos términos que respondían si:

  ✓ ¿cuál es el valor de estar en el estado s?

  ✓ ¿cuál es la ventaja de tomar una acción a estando en ese estado s?



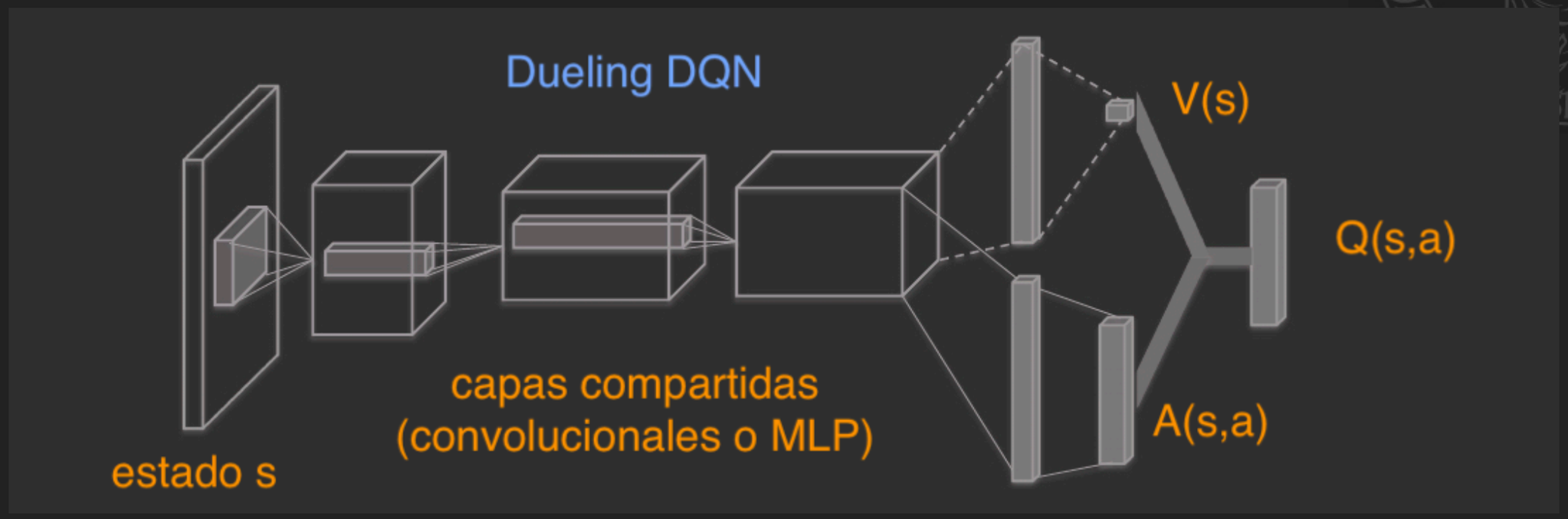


FUNCIÓN DE VENTAJA A(S, A)

- El desacople de Q(s, a) se reflejaba matemáticamente como:

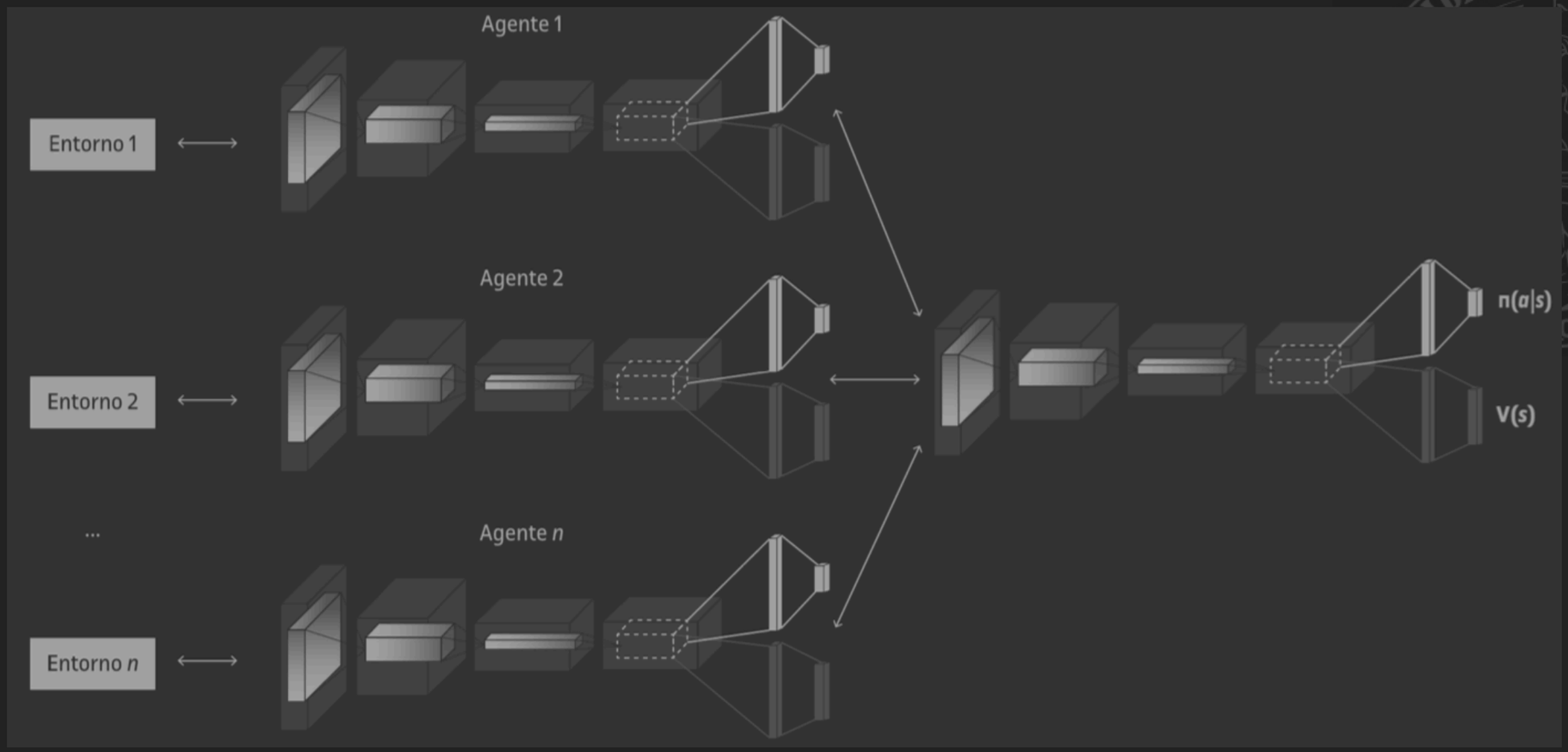
$$Q(s, a) = A(s, a) + V(s)$$

- y también se reflejaba en la arquitectura de la red neuronal.
- A3C toma esta arquitectura de Dueling DQN para su implementación.



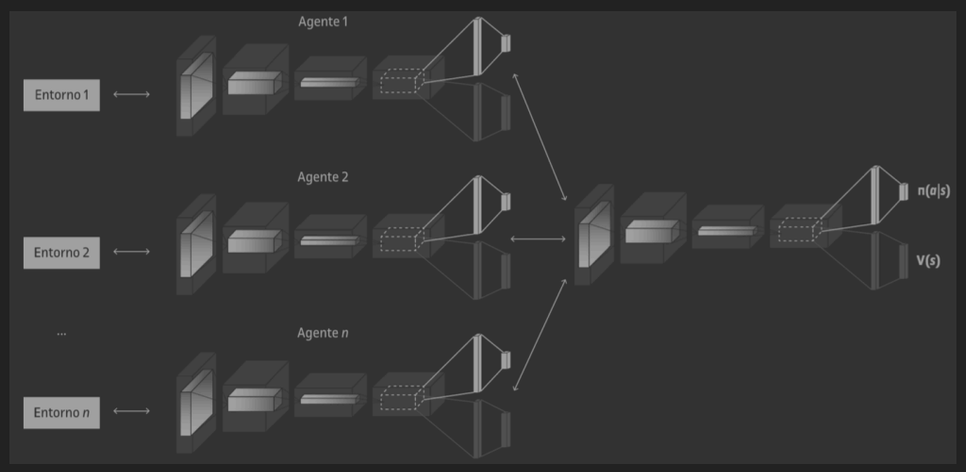
ARQUITECTURA DE VARIOS ACTORES

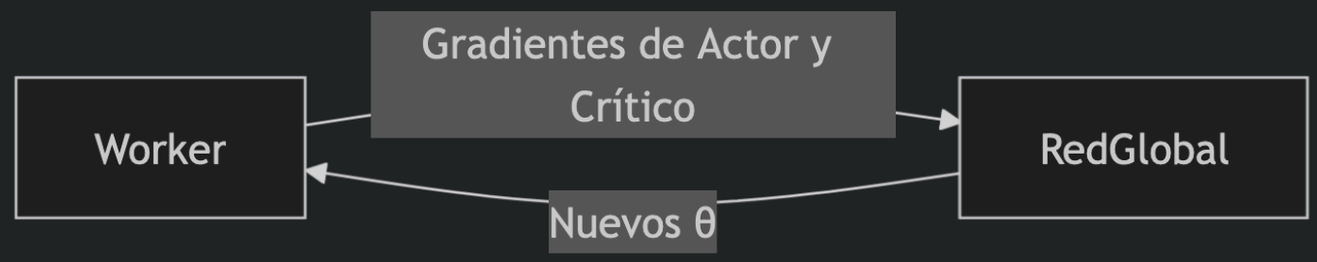
- Dado que el Replay Buffer consume mucha memoria y cómputo A3C propone usar múltiples agentes (actor-learners o workers) ejecutándose en paralelo de forma asíncrona en diferentes hilos (threads) de una CPU estándar.



ARQUITECTURA DE VARIOS ACTORES DE A3C

- Los agentes en forma periódica, asíncrona e independiente, actualizan la función de valor global (global network).
- Después de que se actualiza la global networking, los agentes actualizan sus propios gradientes con los de la red global y continúan normalmente su proceso de aprendizaje n-step hasta la siguiente actualización.
- Toda vez que un agente se actualiza está transfiriendo su información a la global networking.





ARQUITECTURA DE VARIOS ACTORES DE A3C

- El Advantage A(s,a) es un cálculo temporal que ocurre en cada worker, derivado de sus propias experiencias y su copia local del Crítico.

Worker (Local):

|--> Genera (s, a, r, s')

|--> Calcula V(s) con Crítico local

|--> Calcula Recompensa_descontada = Σγr

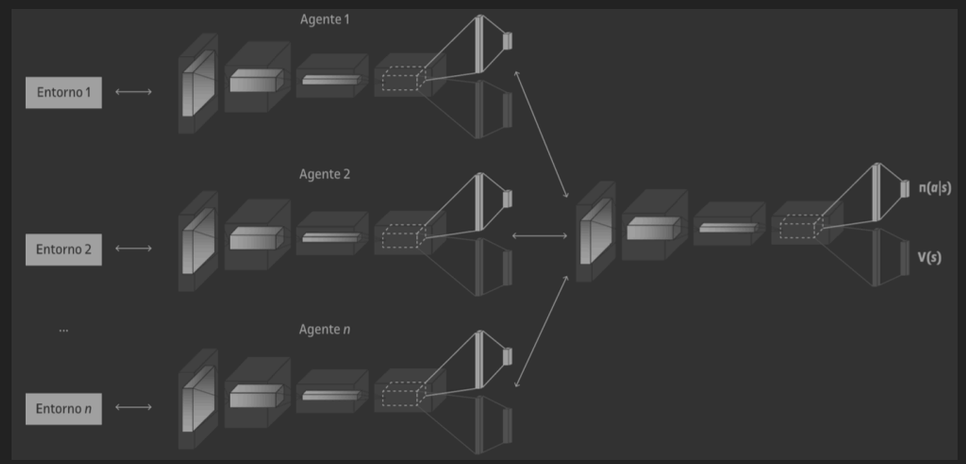
|--> Calcula A(s,a) = Recompensa_descontada - V(s)

|--> ∇θ_actor = -∇logπ(a|s) * A(s,a)

|--> ∇θ_critic = ∇(A(s,a)^2

|--> Envía ∇θ_actor + ∇θ_critic a la red global

(envía gradientes a la red global)

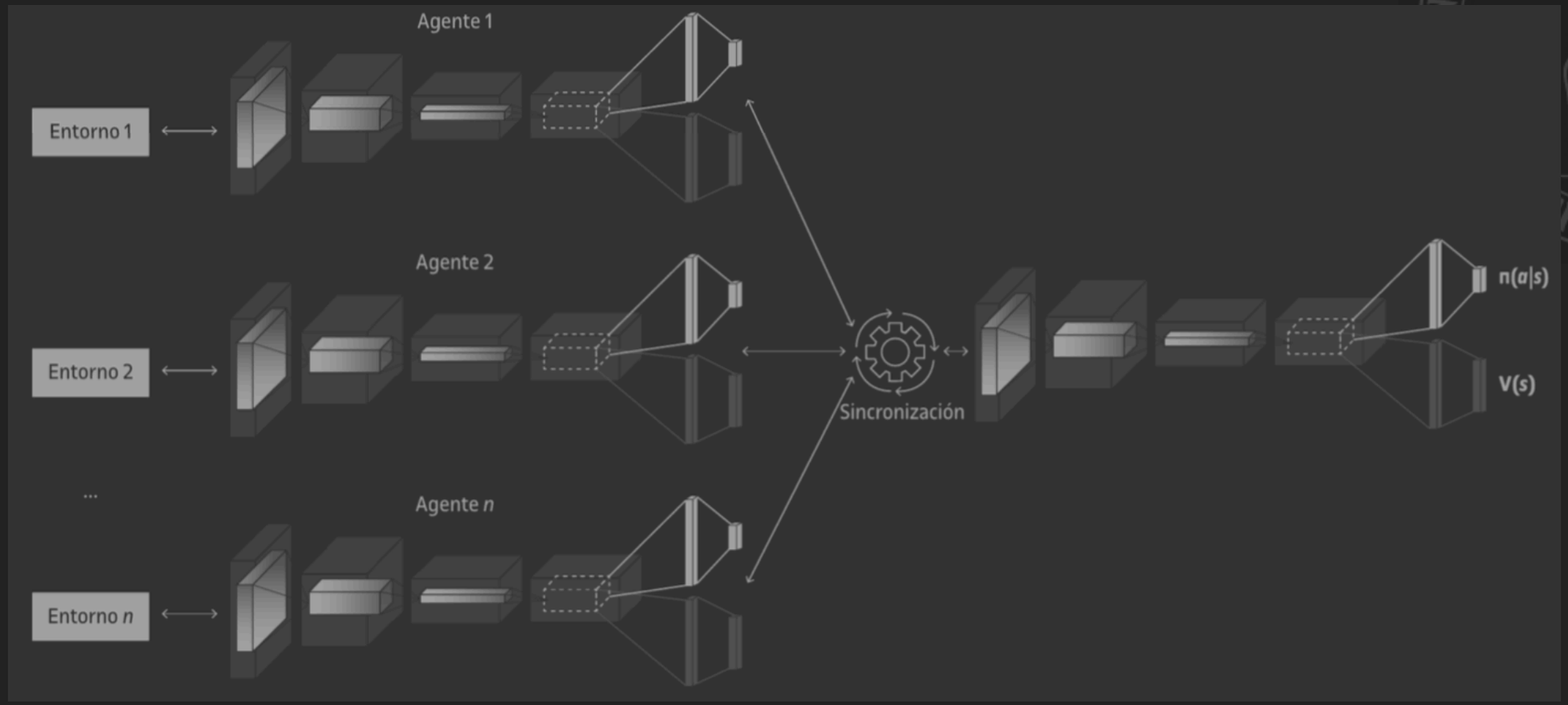


PROBLEMAS DE A3C

- A3C tiene problemas con el overlapping de gradientes (por su cualidad asíncrona).
- A3C requiere ajustes complejos para funcionar bien en entornos modernos.
- A3C fue parte de la biblioteca RLlib hasta la aparición de la versión Ray 2.0 en 2023 (la versión actual de Ray es 2.40, Mayo 2025).
- A3C es una variante de Actor-Critic obsoleta y en creciente desuso, no así A2C.
- A3C es menos eficiente y menos estable que A2C.

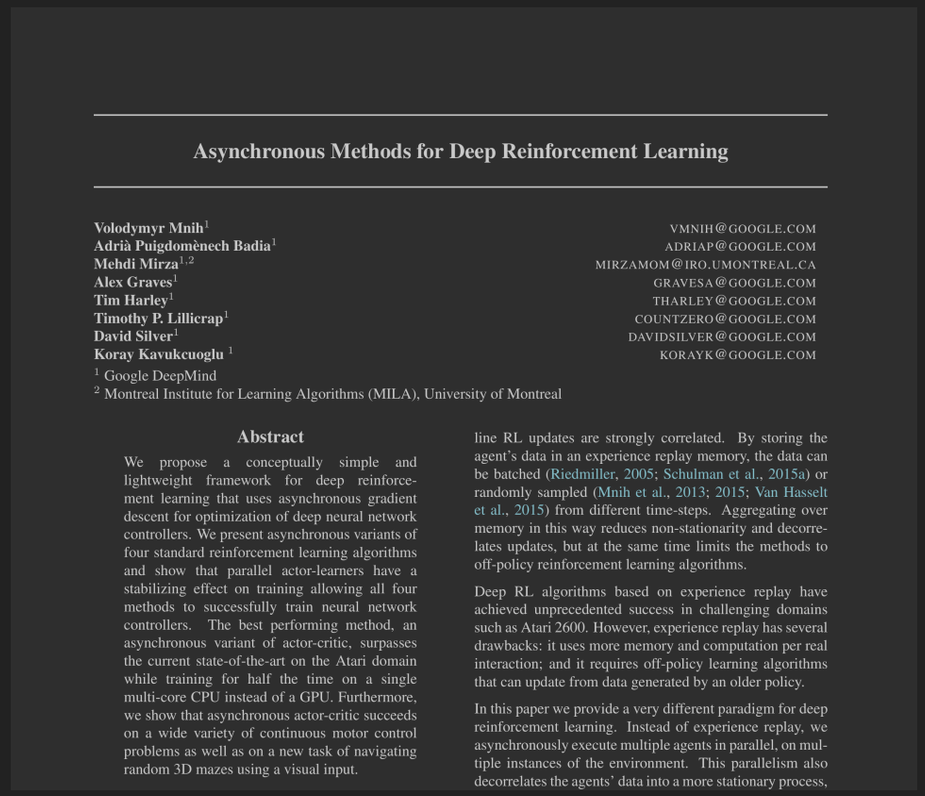
ADVANTAGE ACTOR-CRITIC (A2C)

- A2C es el equivalente síncrono de A3C.
- A2C también tiene varios agentes que se entrenan a la vez, pero luego de que todos finalizan su proceso de aprendizaje (luego de n pasos) se sincronizan todos sus parámetros y la global networking se actualiza.



CONTRIBUCIONES DE A2C

- Reducción de la varianza en las actualizaciones del actor => acelera la convergencia.
- Combina paralelización sincrónica + estimación de ventaja para lograr un aprendizaje más estable y eficiente que los métodos clásicos.



Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM
[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU. Furthermore, we show that asynchronous actor-critic succeeds on a wide variety of continuous motor control problems as well as on a new task of navigating random 3D mazes using a visual input.

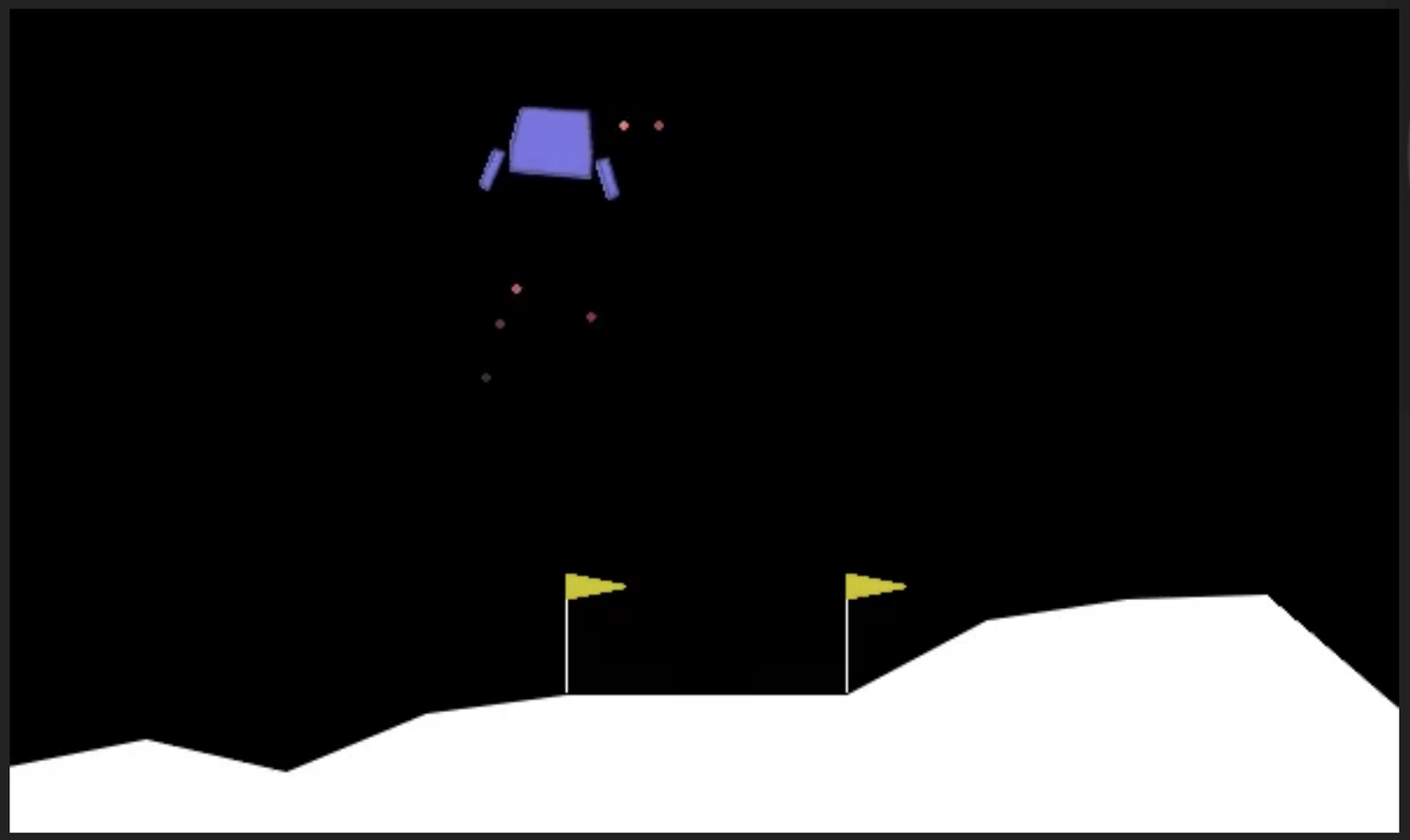
line RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to off-policy reinforcement learning algorithms.

Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

In this paper we provide a very different paradigm for deep reinforcement learning. Instead of experience replay, we asynchronously execute multiple agents in parallel, on multiple instances of the environment. This parallelism also decorrelates the agents' data into a more stationary process,

EJEMPLO

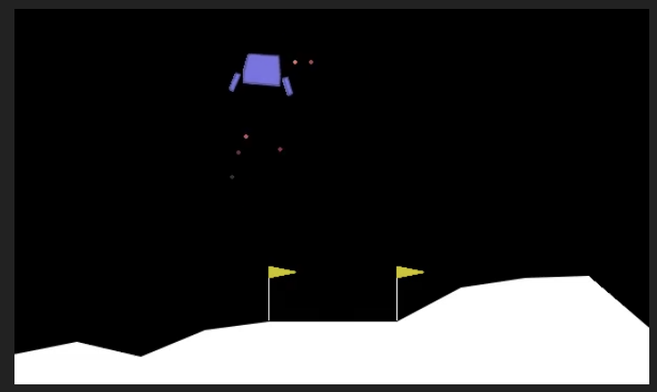
- Aprendizaje del alunizaje con LunaLander de Gymnasium y con la biblioteca de agentes Stable Baseline 3.



PARÁMETROS EN SB3

- policy='MlpPolicy': Especifica la arquitectura de las redes neuronales para el Actor y el Crítico. 'MlpPolicy' significa que se usarán Redes Neuronales Perceptrón Multicapa (Multi-Layer Perceptrons).
- env=train_env: El entorno de entrenamiento con el que interactuará el agente (LunarLander).
- n_steps=5: El número de pasos que cada "worker" (o el único worker en A2C) ejecuta en el entorno para recolectar experiencias antes de calcular las actualizaciones de los modelos. Se recolectan 5 transiciones (s, a, r, s') antes de hacer un paso de optimización.
- gamma=0.99: El factor de descuento (γ).

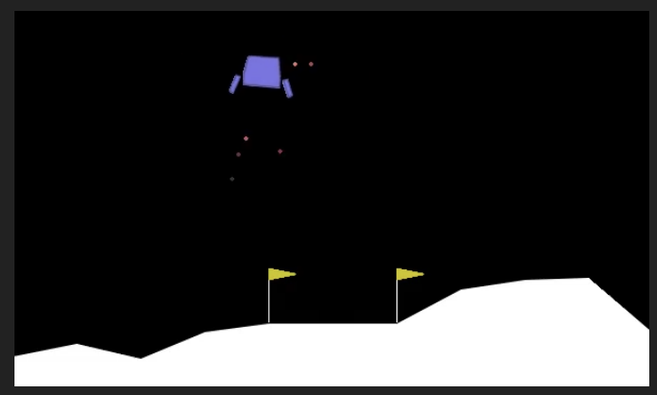
```
# creo el agente A2C
model = A2C(
    policy='MlpPolicy',
    env=train_env,
    n_steps=5,
    gamma=0.99,
    gae_lambda=1.0,
    ent_coef=0.0,
    vf_coef=0.5,
    max_grad_norm=0.5,
    learning_rate=7e-4,
    use_rms_prop=True,
    verbose=1,
    seed=42
)
```



PARÁMETROS EN SB3

- gae_lambda=1.0: Parámetro Lambda para Generalized Advantage Estimation (GAE). GAE es una técnica para estimar la función de ventaja (A(s, a) = Q(s, a) - V(s)) que balancea sesgo y varianza. lambda=1.0 significa que la estimación de la ventaja se parecerá mucho a la ventaja calculada con Monte Carlo (usando retornos completos dentro del lote de n_steps), lo que generalmente reduce el sesgo pero puede aumentar la varianza. lambda=0 se parecería más a la ventaja TD(0).
- ent_coef=0.0: Coeficiente de entropía (c_ent). Es el peso que se le da al término de regularización por entropía en la función de pérdida del Actor. La regularización por entropía anima a la política a ser más estocástica (exploratoria). Un valor de 0.0 significa que no se está usando explícitamente esta regularización (lo cual es un poco inusual para A2C, a menudo se usa un valor pequeño como 0.01).

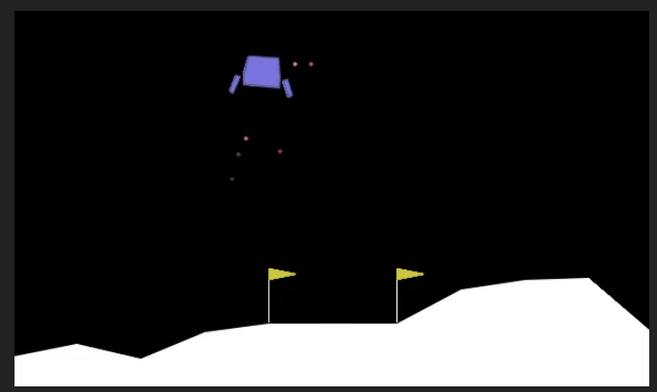
```python
# creo el agente A2C
model = A2C(
    policy='MlpPolicy',
    env=train_env,
    n_steps=5,
    gamma=0.99,
    gae_lambda=1.0,
    ent_coef=0.0,
    vf_coef=0.5,
    max_grad_norm=0.5,
    learning_rate=7e-4,
    use_rms_prop=True,
    verbose=1,
    seed=42
)
```



PARÁMETROS EN SB3

- vf_coef=0.5: Coeficiente de la función de valor (c_vf). Es el peso que se le da a la pérdida del Crítico (value loss) en la función de pérdida total. Ayuda a balancear el aprendizaje de la política (Actor) y la función de valor (Crítico). Si vf_coef es muy alto, el Crítico dominará y el Actor aprenderá lentamente. Si es muy bajo, el Crítico no guiará bien al Actor (política inestable).
- max_grad_norm=0.5: Norma máxima del gradiente. Se utiliza para el recorte de gradientes (gradient clipping). Si la norma L2 total de los gradientes excede este valor, los gradientes se reescalan para que su norma sea igual a max_grad_norm. Esto previene que gradientes muy grandes desestabilicen el aprendizaje. Si los gradientes son [0.3, 0.4], su norma L2 es raiz(0.3^2+0.4^2)=0.50.32+0.42=0.5. Si max_grad_norm=0.5, estos gradientes no se recortan (ya que 0.5≤0.50.5≤0.5). Si fueran [0.6, 0.8] (norma 1.01.0), se reescalarían a [0.3, 0.4].

```python
# creo el agente A2C
model = A2C(
    policy='MlpPolicy',
    env=train_env,
    n_steps=5,
    gamma=0.99,
    gae_lambda=1.0,
    ent_coef=0.0,
    vf_coef=0.5,
    max_grad_norm=0.5,
    learning_rate=7e-4,
    use_rms_prop=True,
    verbose=1,
    seed=42
)
```

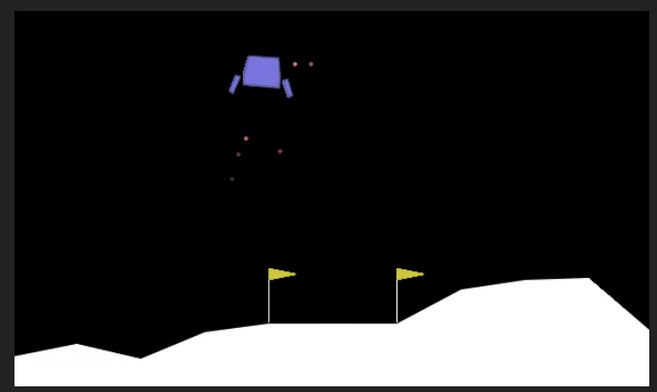


PARÁMETROS EN SB3

- learning_rate=7e-4: Tasa de aprendizaje (α o lr). Controla el tamaño de los pasos que da el optimizador (e.g., Adam, RMSProp) al actualizar los pesos de las redes neuronales. 7e-4 (o 0.0007) es un valor común.

- use_rms_prop=True: Indica si se debe usar el optimizador RMSProp. Si es False, Stable Baselines 3 suele usar ell optimizador adaptativo Adam por defecto.

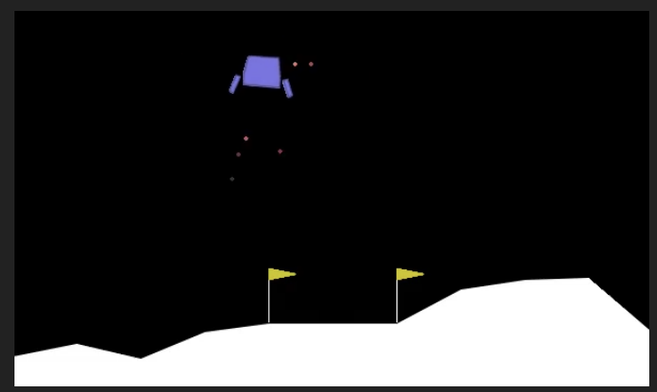
```python
# creo el agente A2C
model = A2C(
    policy='MlpPolicy',
    env=train_env,
    n_steps=5,
    gamma=0.99,
    gae_lambda=1.0,
    ent_coef=0.0,
    vf_coef=0.5,
    max_grad_norm=0.5,
    learning_rate=7e-4,
    use_rms_prop=True,
    verbose=1,
    seed=42
)
```



PARÁMETROS EN SB3

- verbose=1: Nivel de verbosidad. 1 generalmente significa que imprimirá información de progreso durante el entrenamiento. 0 sería silencioso, 2 podría dar más detalles.
- seed=42: Semilla aleatoria. Fija la semilla para las operaciones aleatorias (inicialización de pesos, muestreo de acciones, aleatoriedad del entorno si la soporta) para que los resultados del entrenamiento sean reproducibles.

```
# creo el agente A2C
model = A2C(
    policy='MlpPolicy',
    env=train_env,
    n_steps=5,
    gamma=0.99,
    gae_lambda=1.0,
    ent_coef=0.0,
    vf_coef=0.5,
    max_grad_norm=0.5,
    learning_rate=7e-4,
    use_rms_prop=True,
    verbose=1,
    seed=42
)
```



REFERENCIAS BIBLIOGRÁFICAS Y WEB (I)

- R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
- https://www.kaggle.com/discussions/getting-started/162377
- Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016, June). Dueling network architectures for deep reinforcement learning. In International conference on machine learning (pp. 1995-2003). PMLR.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PmLR.
- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018, July). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning (pp. 1861-1870). Pmlr.
- Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., ... & Levine, S. (2019). Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905.
- https://github.com/ray-project/ray/tree/master/rllib/tuned_examples/sac