*Universidad de Buenos Aires*

*Facultad de Ingeniería*

Especialización en Inteligencia Artificial

**Aprendizaje por Refuerzo II 2025**

**Desafío Práctico**

Octubre 2025

**Autores:**

* Agustin Lopez Fredes

**Asignatura:** Aprendizaje por Refuerzo II

**Tema:** Desafío Práctico

**URL Repositorio:**

https://github.com/Agustinlopezf/CEIA-AprendizajeRefuerzoII

Contenido

[1. Introducción al Problema 3](#_Toc206531416)

[Descripción del Entorno 3](#_Toc206531417)

[2. Enfoque de Solución 4](#_Toc206531418)

[2.1. Algoritmo Seleccionado: Q-Learning 4](#_Toc206531419)

[2.2. Modificaciones al Entorno 4](#_Toc206531420)

[3. Implementación del Código 5](#_Toc206531421)

[Wrapper para modificar recompensas: 5](#_Toc206531422)

[Elección de acción balanceando exploración con ε-greedy: 5](#_Toc206531423)

[Actualización off-policy en base a mejor acción: 6](#_Toc206531424)

[3.1. Parámetros y Métricas 6](#_Toc206531425)

[4. Inconvenientes Encontrados y Soluciones 6](#_Toc206531426)

[5. Resultados y análisis 6](#_Toc206531427)

[5.1. Gráficos de Convergencia 6](#_Toc206531428)

[5.2. Análisis: 8](#_Toc206531429)

[5.3. Métricas Finales 8](#_Toc206531430)

[5.4. Ejecución de política aprendida 8](#_Toc206531431)

[Leyenda: 8](#_Toc206531432)

[Configuración: 8](#_Toc206531433)

[Secuencia de pasos: 9](#_Toc206531434)

[6. Conclusiones 10](#_Toc206531435)

[Referencias 10](#_Toc206531436)

# Introducción al Problema

El desafío práctico consiste en desarrollar un agente de Aprendizaje por Refuerzo Profundo (Deep Reinforcement Learning, DRL) capaz de controlar un bípedo en el entorno **BipedalWalker-v3** de la biblioteca **Gymnasium**.

Este entorno representa un problema continuo de locomoción, donde un agente debe aprender a caminar sobre un terreno irregular controlando sus articulaciones para maximizar la distancia recorrida sin caer.

## Descripción del Entorno

* **Tipo de entorno:** continuo (acciones y observaciones continuas).
* **Observaciones:** vector de 24 dimensiones que incluye posición, velocidad, ángulos articulares y sensores de contacto con el suelo.
* **Acciones:** vector continuo de 4 valores (pares de motores de piernas y caderas).
* **Recompensas:**
  + +300 aprox. si el agente camina correctamente hasta la meta.
  + Penalizaciones por gasto energético o pérdida de equilibrio.
  + -100 aprox. si el agente cae antes de tiempo (episodio termina).
* **Duración máxima de episodio:** 1600 timesteps.
* **Objetivo:** maximizar la recompensa acumulada, aprendiendo una locomoción estable y eficiente.

El problema requiere técnicas de Aprendizaje por Refuerzo continuo con función de valor y política parametrizada.

Se seleccionaron tres algoritmos representativos para realizar la comparativa:

* **A2C (Advantage Actor-Critic)** – método síncrono basado en gradiente de política.
* **PPO (Proximal Policy Optimization)** – política proximal con restricciones KL para mayor estabilidad.
* **SAC (Soft Actor-Critic)** – actor-crítico con entropía máxima para promover exploración adaptativa.

# Enfoque de Solución

## Algoritmo Testeados

### A2C (Advantage Actor-Critic)

## Método actor-crítico síncrono que actualiza la política y la función de valor a partir de estimaciones de ventaja:

Utiliza múltiples workers (entornos paralelos) para estimar gradientes de forma más estable. Es un método *on-policy*, por lo que requiere recopilar nuevos datos tras cada actualización.

## PPO (Proximal Policy Optimization)

Extiende A2C introduciendo una función objetivo “recortada” (*clipped objective*) que restringe las actualizaciones de política para evitar desviaciones abruptas:

Utiliza múltiples workers (entornos paralelos) para estimar gradientes de forma más estable. Es un método *on-policy*, por lo que requiere recopilar nuevos datos tras cada actualización.

**SAC (Soft Actor-Critic)**

Método *off-policy* basado en actor-crítico y entropía máxima.  
Optimiza tanto la recompensa esperada como la entropía de la política, favoreciendo exploración:

donde regula la importancia de la entropía (exploración).

## Modificaciones al entrenamiento

Durante el desarrollo del proyecto se llevaron a cabo diversas modificaciones experimentales sobre los algoritmos A2C, PPO y SAC, con el objetivo de mejorar la estabilidad, la velocidad de aprendizaje y el desempeño final en el entorno continuo BipedalWalker-v3. Cada cambio fue evaluado de forma incremental, analizando su impacto en la convergencia, la estabilidad de las recompensas y el comportamiento del agente durante la simulación.

### Normalización de observaciones y recompensas (VecNormalize)

En una primera etapa, se implementó la clase **VecNormalize** de Stable-Baselines3 para realizar **normalización adaptativa** tanto de observaciones como de recompensas. El objetivo era estabilizar los gradientes y evitar que magnitudes muy grandes o muy pequeñas afectaran de manera desbalanceada el proceso de optimización.

Sin embargo, tras múltiples pruebas se observó que la **normalización de recompensas** (norm\_reward=True) introducía inestabilidad, en particular para los algoritmos off-policy como SAC. Esto se debe a que el valor esperado de las recompensas normalizadas cambia dinámicamente a lo largo del entrenamiento, lo que puede dificultar la correcta estimación de los valores Q.

Por este motivo, se decidió mantener únicamente la normalización de observaciones (norm\_obs=True), y finalmente se descartó completamente el uso de VecNormalize. Los experimentos mostraron que el entrenamiento en el **entorno original sin normalización** producía mejores resultados, especialmente en términos de coherencia de las acciones y recompensas más estables.

**Arquitectura de las redes neuronales**

Con el fin de mejorar la capacidad de representación de las políticas y funciones de valor, se modificaron las arquitecturas internas de las redes neuronales utilizadas por cada agente. Se pasó de configuraciones simples a estructuras más profundas de 256×256 neuronas por capa.

Esta modificación permitió al agente capturar mejor las relaciones no lineales presentes en el entorno continuo y modelar dinámicas más complejas, como el equilibrio y la coordinación de movimiento en múltiples articulaciones. En comparación con redes más pequeñas, las arquitecturas profundas redujeron la varianza de las recompensas y produjeron trayectorias más suaves y consistentes. Los beneficios fueron especialmente evidentes en los modelos PPO y SAC, donde la capacidad de generalización aumentó sin afectar la estabilidad numérica.

**Duración del entrenamiento y convergencia**

El número de timesteps fue otro parámetro crítico en el desempeño de los agentes. En las primeras pruebas se utilizaron 300 000 pasos, suficientes para validar la configuración general y detectar posibles errores de implementación.

No obstante, los resultados mostraron que el agente apenas comenzaba a aprender patrones de equilibrio en esa etapa temprana. Por ello, se amplió la duración del entrenamiento a 600 000 timesteps, en donde logró **movimientos coordinados y marcha estable**. Este comportamiento refleja la naturaleza del aprendizaje por refuerzo continuo, donde la exploración requiere un número considerable de interacciones para alcanzar una política efectiva.

**Tasas de aprendizaje diferenciadas por algoritmo**

Cada algoritmo utiliza estrategias de actualización distintas, por lo que se ajustaron las tasas de aprendizaje (learning\_rate) según su naturaleza. Para los métodos on-policy (A2C y PPO) se empleó un valor bajo de 3×10⁻⁵, priorizando la estabilidad de la política y evitando oscilaciones en la función de valor. En cambio, para SAC, al tratarse de un método off-policy más robusto y con aprendizaje más agresivo, se utilizó una tasa mayor de 3×10⁻⁴.

Este ajuste permitió un equilibrio adecuado entre estabilidad y velocidad de convergencia. En SAC, el aprendizaje más rápido se compensó con una tasa de target smoothing (τ = 0.02) que evitó actualizaciones bruscas de los valores Q, mejorando la consistencia entre la política y el valor crítico.

**Parámetros del buffer de experiencia y ritmo de actualización**

En el caso de SAC, se ajustaron los parámetros del replay buffer y la frecuencia de actualización del modelo. Se redujo el tamaño del buffer a 150 000 muestras para evitar una mezcla excesiva de experiencias antiguas durante los entrenamientos cortos de 300 000 timesteps, y se configuraron las tasas de actualización en:

* train\_freq = 32
* gradient\_steps = 32

Esta configuración 1:1 entre pasos y actualizaciones produjo una dinámica de entrenamiento más estable y eficiente. Se comprobó que valores mayores aumentaban el tiempo de cómputo sin mejorar las recompensas obtenidas.

**Early stopping y reentrenamiento controlado**

Para evitar sobreentrenamiento y reducir el tiempo de ejecución innecesario, se implementó un callback de parada temprana basado en evaluaciones periódicas:

*StopTrainingOnNoModelImprovement(max\_no\_improvement\_evals=10)*

Este mecanismo detuvo automáticamente el entrenamiento cuando el agente no mostraba mejoras en la recompensa media durante diez evaluaciones consecutivas, tras un mínimo de cinco iteraciones iniciales.

Adicionalmente, se introdujo un parámetro manual reentrenar = True/False, que permitía continuar el aprendizaje desde un modelo previamente guardado o reiniciar desde cero. Cuando reentrenar=False, el script cargaba automáticamente el modelo más reciente.

# Implementación del Código

El repositorio se ubica en :

<https://github.com/Agustinlopezf/CEIA-AprendizajeRefuerzoII>

Los puntos más importantes son:

### Estructura general del entorno::

def make\_env():

env = gym.make("BipedalWalker-v3")

env = Monitor(env, filename=os.path.join(log\_dir, "monitor.csv"))

return env

train\_env = DummyVecEnv([make\_env])

train\_env = VecNormalize(train\_env, norm\_obs=True, norm\_reward=True)

Durante las primeras pruebas se utilizó VecNormalize para normalizar observaciones y recompensas, pero finalmente se trabajó con el entorno original sin normalización, ya que los resultados mostraron un mejor desempeño.

Cada algoritmo (A2C, PPO y SAC) se implementó en un script independiente, con configuración de callbacks, registro en TensorBoard y almacenamiento automático de modelos, métricas y gráficos de evolución.

## Parámetros y Métricas

El entrenamiento final se realizó con 600 000 timesteps para cada agente, suficiente para observar convergencia sin extender excesivamente el tiempo de cómputo.

Los parámetros más relevantes fueron:

* **Arquitectura de red**: dos capas ocultas de 256 neuronas (256×256).
* **Tasa de aprendizaje:**
  + A2C y PPO → 3×10⁻⁵
  + SAC → 3×10⁻⁴
* **Tamaño de batch**: 256
* **Buffer de replay (SAC):** 500 000 muestras
* **Evaluación periódica**: cada 10 000 pasos, con early stopping tras 10 evaluaciones sin mejora.
* **Registro de métricas**: recompensas medias, longitudes de episodios y tasas de éxito, registradas tanto en progress.csv como en TensorBoard.
* **Guardado automático**: modelos en ./models/[algoritmo]/best\_model.zip y resultados en ./logs/[algoritmo]/.

# Inconvenientes Encontrados y Soluciones

### Convergencia lenta inicial

En los primeros 100 000 pasos los agentes tendían a caerse con frecuencia debido a exploración excesiva.

Solución: incremento progresivo de la duración del entrenamiento y ajuste de las tasas de actualización (train\_freq y gradient\_steps) para acelerar la estabilización.

### Recompensas ruidosas

Las recompensas crudas del entorno presentaban alta varianza, dificultando el análisis.

Solución: aplicación de una media móvil con ventana de 1 000 pasos en los gráficos de entrenamiento para visualizar tendencias de mejora.

### Sobreajuste y estancamiento

En entrenamientos prolongados algunos agentes mostraban degradación tras alcanzar máximos de recompensa.

Solución: incorporación del callback de early stopping (StopTrainingOnNoModelImprovement) que detiene el entrenamiento tras 10 evaluaciones sin mejora, evitando consumo innecesario de recursos.

### Reentrenamiento inconsistente

Al cargar modelos antiguos, las recompensas no coincidían con las del entrenamiento original debido a diferencias de entorno.

Solución: unificación de la configuración mediante el parámetro reentrenar y eliminación del VecNormalize para garantizar reproducibilidad.

# Resultados y análisis

## Gráficos de Convergencia

Se incluye un gráfico de recompensas vs. Episodios, así como la tasa de éxitos y fracasos (caída en agujeros) cada 1000 episodios. Usamos media móvil para suavizar.

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

## Análisis:

* Recompensas: Aumentan de negativas a aproximadamente 0.94 (1 menos penalizaciones por 6 pasos óptimos). Convergencia alrededor de 50k episodios.
* Tasa de Éxito: Asciende a aproximadamente 0.99, indicando política óptima.
* Caídas: Descienden a aproximadamente 0.01, mostrando aprendizaje de evitación.

Estos gráficos demuestran convergencia efectiva del algoritmo.

## Métricas Finales

Ejemplo de salida (de una ejecución):

* Éxitos: 198,500/200,000 (99.25%).
* Tasa de éxito: 99.25%.
* Caídas: 1,200/200,000 (0.60%).
* Longitud promedio: 6.2 pasos (óptimo 6 en grilla 4x4).

Tasa de éxito en pruebas finales: 100%.

## Ejecución de política aprendida

A continuación, se muestra la ejecución de la política aprendida por el agente, en formato ansi:

### Leyenda:

* **S**: Inicio (start)
* **F**: Superficie congelada (frozen)
* **H**: Agujero (hole)
* **G**: Objetivo (goal)
* **X**: Posición del agente
* **Acciones**: 0 = Izquierda, 1 = Abajo, 2 = Derecha, 3 = Arriba

### Configuración:

* Penalización por paso: -0.01
* Penalización por caer en un agujero: -0.3
* Recompensa por alcanzar el objetivo: 1.0

### Secuencia de pasos:

| **Paso** | **Estado Actual** | **Acción** | **Recompensa** | **Próximo Estado** | **Tablero (Posición del Agente)** |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | - | - | - | **S**FFF  FHFH  FFFH  HFFG |
| 1 | 0 | Abajo (1) | -0.01 | 4 | SFFF  **X**HFH  FFFH  HFFG |
| 2 | 4 | Abajo (1) | -0.01 | 8 | SFFF  FHFH  **X**FFH  HFFG |
| 3 | 8 | Derecha (2) | -0.01 | 9 | SFFF  FHFH  F**X**FH  HFFG |
| 4 | 9 | Derecha (2) | -0.01 | 10 | SFFF  FHFH  FF**X**H  HFFG |
| 5 | 10 | Abajo (1) | -0.01 | 14 | SFFF  FHFH  FFFH  HF**X**G |
| 6 | 14 | Derecha (2) | 1.0 | 15 | SFFF  FHFH  FFFH  HFF**X** |

Se observa que llega correctamente al objetivo en una secuencia mínima de pasos.

# Conclusiones

Se resolvió exitosamente el problema de FrozenLake usando Q-Learning, logrando alta tasa de éxito y convergencia rápida. Las modificaciones al entorno y decaimientos mejoraron el aprendizaje. También se observa que el agente aprendió una política óptima de pasos.

Este trabajo demuestra aplicación práctica de RL en entornos discretos.

# Referencias

* Gymnasium Documentation. Disponible en: https://gymnasium.farama.org/
* Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.
* Material del cursado: Aprendizaje por Refuerzo I, FIUBA.