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

Nombre notebook: comparativa_algoritmos_bipedal.ipynb

Contenido

1. 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 una 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.

2. Enfoque de Solución

2.1. 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:

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

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:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t)]$$

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:

$$J(\pi) = \sum_t E_{(st,at)\sim\rho\pi}[r(s_t,a_t) + \alpha H(\pi(\cdot|s_t))]$$

donde α regula la importancia de la entropía (exploración).

2.2. 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 ($\tau = 0.02$) 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 aumentó el tamaño del buffer a 500.000 muestras para evitar que se refrescara demasiado seguido, 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.

3. 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.

3.1. 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 $\longrightarrow 3\times10^{-5}$
  - SAC $\longrightarrow 3\times10^{-4}$
- **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]/.

4. 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 en SAC) para acelerar la estabilización.

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.

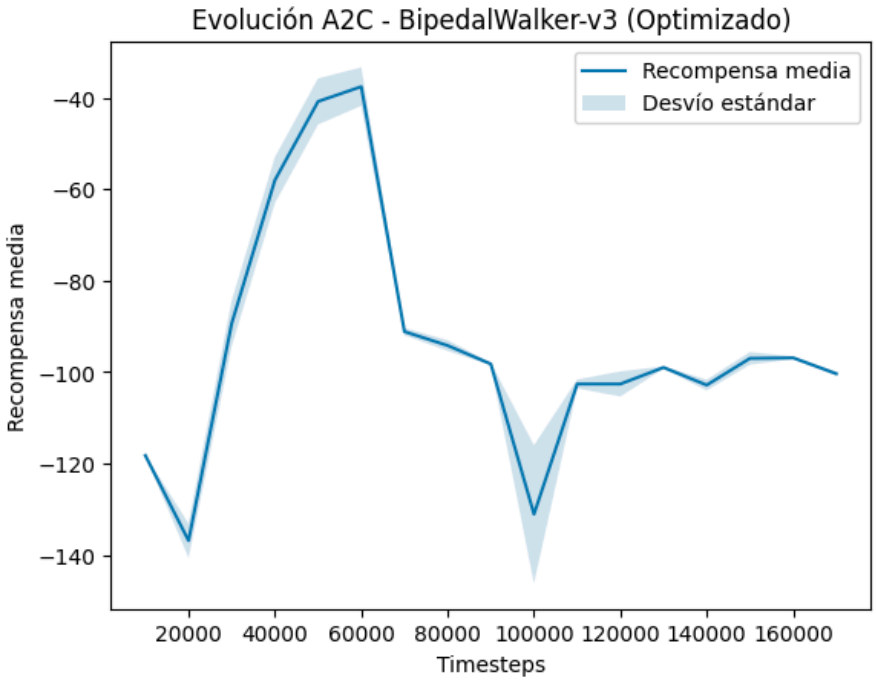
Reentrenamiento inconsistente

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.

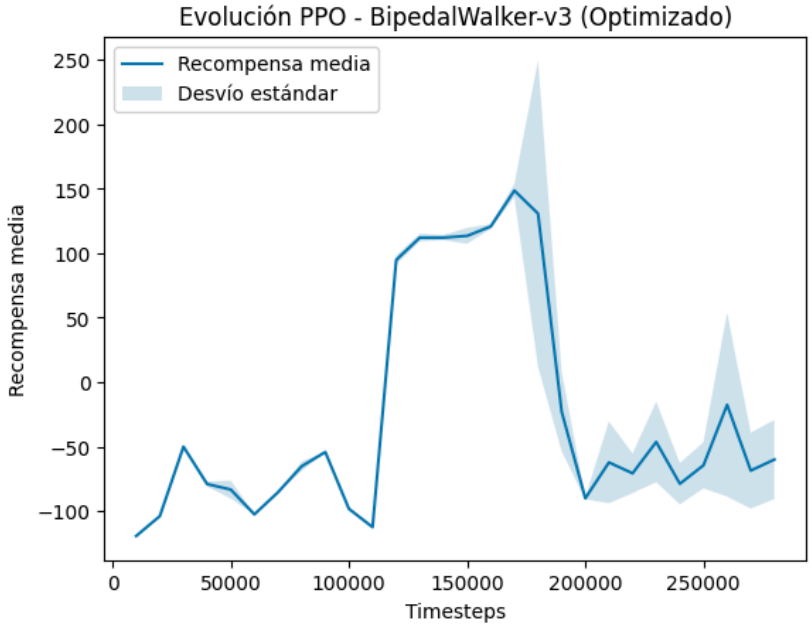
5. Resultados y análisis

5.1. Gráficos de Convergencia

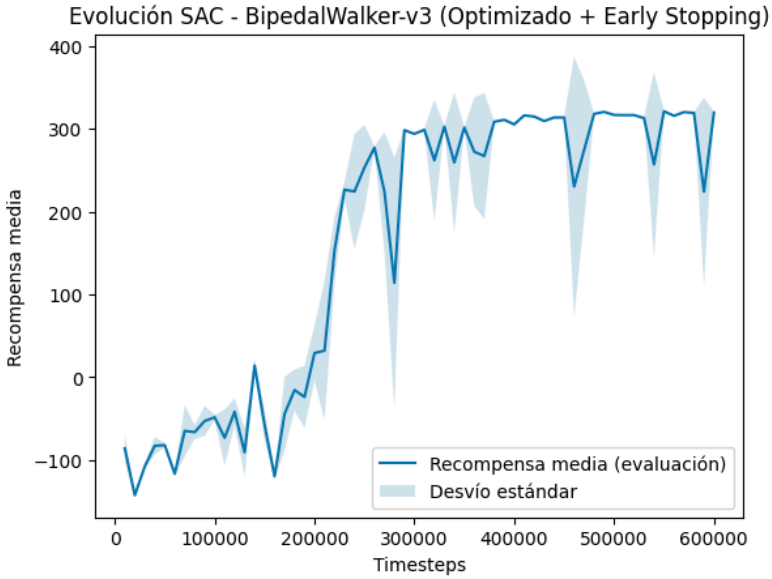
El análisis incluye un gráfico de recompensas promedio por episodio en función de los timesteps de entrenamiento, evaluadas cada 10.000 pasos.



Para A2C, se observa que luego de un incremento inicial de la recompensa, cae nuevamente, y finalmente se produce un Early Stopping luego de 170.000 timesteps. La recompensa siempre es negativa, mostrando que el agente no logra dar pasos sin caerse. Esto puede deberse a ruido en la estimación de Q, al ser un entorno continuo.



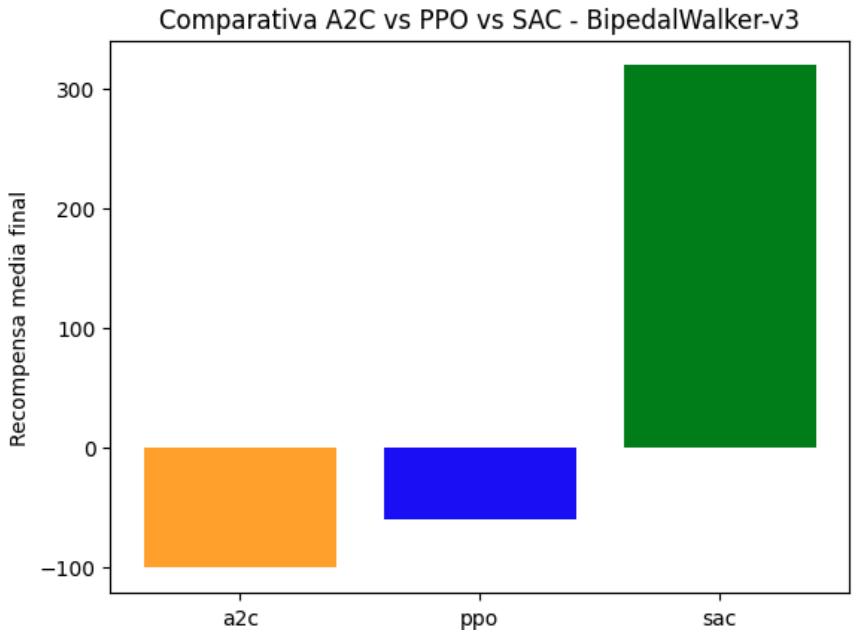
Para PPO, el agente nuevamente logra una mejora en la recompensa y vuelve a caer, hasta activarse el Early Stopping. En este caso, para algunos episodios la recompensa es positiva, indicando que el agente logra dar algún paso sin caerse.



Para SAC, la recompensa crece de forma constante, llegando a valores positivos, lo que muestra que el agente logra aprender a dar pasos. No se activa el Early Stopping, y completa los 600.000 timesteps.

Comparativa de resultados

A continuación se muestra un gráfico comparativo de la última evaluación:



Se comprueba que PPO logra mejoras con respecto a A2C, mientras que SAC es muy superior a ambos.
Esto también se comprueba en los videos guardados (ver carpeta “videos” del repositorio):

- En A2C nunca se logra que el agente de un paso.
- En PPO, el agente camina de forma torpe, y en uno de los episodios da algunos pasos antes de caer, mientras que en otro da varios pasos y se cae sobre el final.
- En SAC, el agente logra caminar de forma continua, y ambos videos concluyen sin que el mismo se caiga.

Tanto a nivel de recompensas, como visualmente, se concluye que el algoritmo SAC da resultados muy superiores al resto.

6. Conclusiones

Los resultados confirman una progresión clara de rendimiento: A2C falla completamente sin lograr ningún paso efectivo, PPO muestra mejoras significativas con locomoción torpe pero inestable, mientras que SAC demuestra superioridad tanto cuantitativa como cualitativa, logrando caminar de forma continua sin caídas. Esta ventaja de SAC se debe principalmente a su naturaleza off-policy, que le permite reutilizar experiencias pasadas de su buffer de replay, combinada con su principio de entropía máxima que mantiene exploración constante. A diferencia de A2C y PPO que descartan experiencias anteriores por ser on-policy, SAC aprovecha eficientemente toda la información disponible, generando políticas más robustas y estables para el control continuo en entornos complejos como BipedalWalker.

Referencias

- Documentación Gymnasium. Disponible en: https://gymnasium.farama.org/
- Documentación Stable Baselines 3. Disponible en: https://stable-baselines3.readthedocs.io/en/master/
- Material del cursado: Aprendizaje por Refuerzo II, FIUBA.