KungFuMaster-v5 Agente

Stephany Michell Revuelta Sosa UX23II402

***Resumen*— En este proyecto entrené un agente para jugar *Kung Fu Master* usando aprendizaje por refuerzo profundo. La idea fue entender cómo un modelo puede aprender solo a partir de la experiencia, sin decirle exactamente qué hacer. Para lograrlo utilicé un enfoque tipo DQN que procesa varias imágenes del juego y decide qué acción tomar según los valores Q que va aprendiendo. El entrenamiento se llevó a cabo con un millón de pasos usando RL-Baselines3-Zoo, con un Replay Buffer grande y una tasa de aprendizaje estándar para este tipo de modelos.**

Durante las pruebas se registraron recompensas, duración de los episodios y el comportamiento general del agente. Los resultados mostraron una tendencia clara: mientras más entrena, mejor juega. Las recompensas aumentaron con el tiempo y el agente logró episodios más largos, lo que indica que sí aprendió patrones útiles del entorno. Aun así, el proceso fue lento, principalmente por las limitaciones de hardware y la demanda computacional que tienen estos entornos de Atari.

***Abstract*-- In this project, I trained an agent to play Kung Fu Master using deep reinforcement learning. The goal was to understand how a model can learn on its own from experience, without being explicitly told what to do. To achieve this, I used a DQN-type approach that processes multiple images of the game and decides which action to take based on the Q values it learns. Training was performed with one million steps using RL-Baselines3-Zoo, with a large repetition buffer and a standard learning rate for this type of model.**

During testing, rewards, episode duration, and the agent's overall behavior were recorded. The results showed a clear trend: the more the agent trained, the better it played. Rewards increased over time, and the agent achieved longer episodes, indicating that it learned useful patterns from the environment. Even so, the process was slow, mainly due to hardware limitations and the computational demands of these Atari environments.

I. Introducción

En este proyecto se creó un agente capaz de jugar el juego kung fu master de Antari. El propósito central es analizar cómo un agente inteligente puede aprender a desenvolverse dentro de un entorno desconocido únicamente mediante interacción, reforzando así los conceptos estudiados en aprendizaje profundo y agentes autónomos. Este trabajo además busca vincular la teoría con la práctica, mostrando un ejemplo funcional de toma de decisiones basada en redes neuronales.

Los objetivos específicos son:

1. Reforzar los conocimientos de Deep Learning aplicados al control secuencial.
2. Unir los conceptos de agentes y redes neuronales dentro de un ejemplo real y práctico.
3. Entrenar un agente que logre jugar de manera eficiente optimizando su comportamiento por retroalimentación.

Contexto de Deep learning

En el contexto del juego el Deep learning es la clave del comportamiento del agente ya que la idea es que el agente aprenda del entorno interactuando con el , recibiendo una entrada de 4 frames para que puede tener dirreción del entorno; A través de prueba – error , recibiendo recompensas como retroalimentación pueda realizar acciones.

Una definición formal del aprendizaje profundo descrita en el curso de huggingface es:

"El aprendizaje por refuerzo es un marco para resolver tareas de control (también llamadas problemas de decisión) mediante la construcción de agentes que aprenden del entorno

interactuando con él a través de ensayo y error y recibiendo recompensas (positivas o negativas) como retroalimentación única." [1]

El proceso RL es un bucle de estado , acción , recompensa y siguiente estado y este va ser en pocas palabras el comportamiento interno del agente, por esto mismo se decidió entrenar al agente con redes neuronales y Deep Q-Learning.

En videojuegos, esta dinámica es fundamental: el agente recibe una secuencia de *frames* del juego como entrada y utiliza una red neuronal para estimar el valor de las acciones posibles. Deep Q-Learning permite que el agente no solo imite conductas, sino que desarrolle estrategias emergentes para incrementar su desempeño y alcanzar niveles óptimos.

II. Antecedentes y fundamentos teóricos

2.1 Deep Q-Learning:

En teoría esta es la técnica que usaremos para entrenar la función q, una definición otorgarda por el Hugging Face es:

"Deep Q-Learning utiliza una red neuronal que toma un estado y aproxima los valores Q para cada acción en función de ese estado." [1]

* Revista Argentina de Trabajos Estudiantiles. Patrocinada por la IEEE.

Otro termino que es necesario para comprender el funcionamiento y proceso de aprendizaje es Q función , este es una función de acción-valor que determina el valor de estar en un estado particular y tomar una acción específica en ese estado , Q es por la calidad de la acción en ese estado.

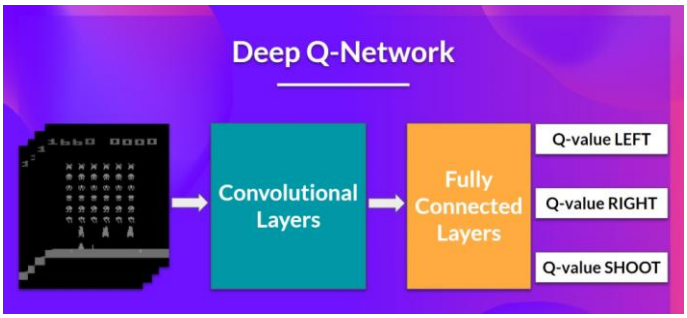
Limitaciones de Q- Learning

Su mayor limitación es que como vimos es un método tabular y eso funciona muy bien en problemas que tienen espacios pequeños o pocos, pero si el entorno tiene un espacio muy grande Q- Learning no es escalable lo cual no es posible representralos en tabla y matrices de forma eficaz.

La solución brindada en este caso es una red neuronal que aproximará dado un estado los valores q para cada acción posible en el estado y basado en esto se elegirá la mejor opciones.

2.2 Deep network

La network tiene la siguiente estructura: entrada de frames – red con capaz convolutivas - capaz conectivas y la salida de Q valores obtenidos:



Como entrada toma una pila de 4 frames para luego pasar por la red y crear un vector con las Q de las posibles acciones para el estado de cada frame, cuando inicia la red es ineficiente pero conforme avanza el entrenamiento el agente de la red q aprende de las acciones y con el tiempo jugara de forma más eficiente.

2.3 Función de perdida

La diferencia en Q- Learning es que en lugar de actualizar el valor q con par- acción , se actualiza con una función de perdida.

Esta función compara el valor del Q y el objetivo Q, lo que en una estructura teórica de redes neuronales sería d y y , el valor obtenido y el valor esperado. también utiliza el descenso degradiente para actualizar los pesos de la Deep network para aproximar mejor los valores Q.

2.4 Fases de Deep learning

El Deep learning se divide en dos fases la primera es el muestreo en el cual se realizan acciones y se almacenan las tuplas de experiencias observadas en una memoria de repetición. La segunda fase es el entrenamiento donde se selecciona un lote pequeño de tuplas al azar, para que aprenda de este lote utilizando un paso de actualización de descenso de gradiente .

2.5 Técnicas para estabilizar el entrenamiento

El Deep learning puede demostrar inestabilidad, para esto se pueden usar tres soluciones , la primera es reproducir experiencias para hacer uso de las experiencias , la segunda es utilizar Q- target .

Concepto de Q-target : Es una copia congelada de la red principal que no se actualiza constantemente, solo cada cierto número de pasos, esto hace a la red más estable. [2]

La tercera solución es usar double Deep learning para manejar la sobreestimación de valores Q, en esta técnica se calcula el objetivo Q, utilizamos dos redes para disociar la selección de la acción de la generación del valor Q objetivo, el double deep learning ayuda a entrenar más rapido y de forma más estable.

2. 6 Replay Buffer

Es un componente fundamental en el Deep learning su uso es almacenar experiencias pasadas para que el agente pueda reutilizarlas durante el entrenamiento, esta memoria se guarda en tuplas .

También se usa para evitar el olvido catastrófico este problema surge cuando la red olvida las experiencias pasadas a medida que obtiene nuevas.

III. Metodología

3.1 Descripción del entorno

El entorno donde se encuentra el agente es el juego de antari kung fu master, que se muestra en la siguiente imagen

Donde el agente es el personaje blanco (el maestro que pelea contra los ladrones), en este se tienen 14 acciones posibles las cuales son:

0 noop	8 fuego izquierdo
1 arriba	9 fuego bajo
2 bien	10 fuego lateral
3 izquierda	11 fuego vertical
4 abajo	12 fuego arriba
5 completamente	13 fuego directo
6 abajo izquierda	14 fuego abajo izquierdo
7 fuego derecho	

3.1.2 Observaciones del agente

El entorno de antari tiene trees tipos de observaciones posibles:

1) obs_type="rgb">observation_space=Box(0, 255, (210 , 160, 3), np.uint8) cada imagen es de 210×160 píxeles con 3 canales de color (RGB) con enteros de 8 bits.

2) obs_type="ram">observation_space=Box(0, 255, (12 8,), np.uint8) en esta observabilidad no se ve una imagen si no un vector de 128, que corresponde al estado.

3) obs_type="grayscale"-> , una versión en escala de grises tipo q"rgb"Box(0, 255, (210, 160), np.uint8)

3.1.3 Recompensas

El agente recibe una recompensa de 100 puntos cuando logra golpear o a su enemigo, mientras no lo logra continua con 0 de recompense.

```
Recompensa por movimiento: 0.0
Recompensa por movimiento: 0.0
Recompensa por movimiento: 0.0
Recompensa por movimiento: 0.0
Recompensa por movimiento: 0.0
Recompensa por movimiento: 100.0
Recompensa por movimiento: 0.0
```

3.2 Configuración del agente

Como se mencionó anteriormente la red tiene la siguiente estructura :

1) De entrada 4 frames
2) Capas convolucionales
3) Capas totalmente conectadas
4) Salidas de Q valores

El agente se entrenó utilizando DQN con una política convolucional (CnnPolicy), adecuada para observar el entorno mediante imágenes.

3.2.2 Hiperparametros

1) Pasos de entrenamiento n_timesteps , se utilizaron 1 e6 pasos ya que antari necesita muchas experiencias.
2) Replay buffer, para su memoria se utilize un tamaño de 1,00,0000.
3) La taza de aprendizaje learning_rate , se fijo en 1 e4.
4) El batch-size fue de 32, equilibrando consumo de GPU/memoria y estabilidad en el gradiente.
5) Antes de inciar el entrenamiento llena un millón de experiencias learning_starts.
6) Se entrenó cada cuatro pasos train_freq=4, con un solo paso de gradiente por actualización, manteniendo un aprendizaje estable y controlado.
7) Para la fracción de la exploración exploration_fraction, 10% del entrenamiento fue exploración.

A continuación se muestra el archive de configuración de hiperparametros donde se pueden visualizar todas las metricas antes mencionadas:

dqn.yml

```yaml
atari:
  env_wrapper:
    - stable_baselines3.common.atari_wrappers.AtariWrapper
  frame_stack: 4
  policy: 'CnnPolicy'
  n_timesteps: !!float 1e6
  buffer_size: 100000
  learning_rate: !!float 1e-4
  batch_size: 32
  learning_starts: 100000
  target_update_interval: 1000
  train_freq: 4
  gradient_steps: 1
  exploration_fraction: 0.1
  exploration_final_eps: 0.01
  optimize_memory_usage: False
```

Los hiperparámetros utilizados se seleccionaron buscando un equilibrio entre calidad de aprendizaje y capacidad de cómputo disponible. Se entrenó el agente durante 1e6 pasos porque Atari requiere una gran cantidad de experiencias para aprender estrategias útiles, pero valores mayores habrían resultado demasiado pesados para la computadora utilizada. De igual forma, se eligió un Replay Buffer de 1e6 experiencias, lo suficiente para almacenar variedad de estados sin saturar memoria. La tasa de aprendizaje 1e-4 permitió actualizaciones estables, mientras que un batch de 32 equilibró velocidad y consumo de recursos.

3.2.3 Implementación con RL-Baselines3-Zoo

Para utilizar RL Baselines3 Zoo fue necesario instalar previamente varias dependencias. El primer paso fue instalar y habilitar Git, ya que esta librería se obtiene directamente desde

el repositorio oficial en GitHub y no desde un paquete estándar de pip.

Como Segundo paso fue necesario instalar el paquete Atari y aceptar la licencia de la ROM para descargar los archivos de los juegos.

Otro punto importante fue decargar pytorch con cuda, para que usara el gpu al momento de entrenar, con esta version el entrenamiento tardaba unas 2 horas aproximadamente pero sin usar el gpu tardaba de 6 9 horas.

3.2.4 Entorno personalizado

Para no mezclar el proyecto con otros y no alteral la configuración de otras librerías antes instaladas , se creó un entorno virtual llamado juego -evn , en este se realizó la configuración mencionada en el punto 3.2.3

En este entorno también se creo el archive dnq.yml en el cual se realizó la configuración de los hiperparametros.

3.2.5 Comandos para entrenar y Probar

Para entrenar se creó una carpera logs donde se guardan los modelos, el commando para entrenar fue:

Este comando ejecuta el módulo de entrenamiento de RL-Baselines3-Zoo, especificando que se trabajará con el algoritmo dqn, donde y se ingresa el nombre del juego Kung fu Master y se especifica la version.

Para probar el modelo se utilizó el siguente commando:

Ejecuta el agente previamente entrenado con el algoritmo dqn en el entorno del kung fu .

La opción --folder logs/ indica la ubicación donde se encuentran los modelos guardados, --n-timesteps 5000 define el número de pasos que el agente ejecutará durante la prueba, y --no-render permite correr la evaluación sin visualizar la pantalla del juego, reduciendo el uso de recursos.

3.3 Estrategia de evaluación

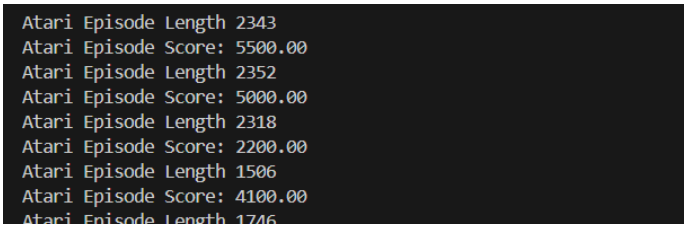
Para medir el desempeño del agente entrenado se realizó una fase de evaluación utilizando el comando rl_zoo3.enjoy, donde el modelo jugó múltiples episodios del entorno ALE/KungFuMaster-v5. Durante esta etapa se registraron dos métricas principales:

Score por episodio, que representa la recompensa total acumulada durante el juego.

Longitud del episodio, es decir, la cantidad de pasos que el agente sobrevivió o permaneció activo dentro del entorno.

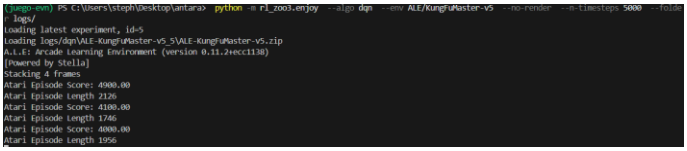
La prueba se ejecutó por N timesteps = 5000, lo que permitió observar el comportamiento del agente a lo largo de varios episodios consecutivos.

En los resultados se muestran puntuaciones que oscilaron entre 2200 y 6100 puntos, junto con longitudes de episodio entre 1506 y 2747 pasos, lo cual permitió evaluar la consistencia del aprendizaje y la estabilidad de la política entrenada.



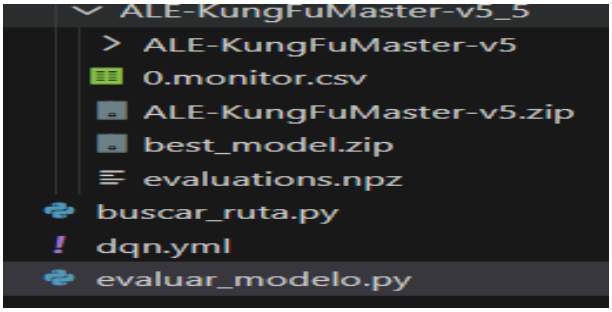
3.3.1 Metricas de evalución

Para evaluar el desempeño del agente durante el entrenamiento sin necesidad de renderizar la imagen del entorno, se utilizó el siguiente comando:



El parámetro --no-render permite ejecutar la simulación sin mostrar la interfaz visual del juego, lo cual reduce el consumo de recursos y facilita la obtención de métricas como puntajes por episodio, longitud temporal y comportamiento del modelo durante la interacción con el entorno.

Por último se crearón graficas de recompeza por epoca y otra por duración del episodio , tomando de datos la información guardada en el archivo 0.motor.csv:



Ya en este archivo se registran todas las recompensas de cada epoca durante el entrenamiento, y este contiene los siguentes valores por ejemplo:
#{"t_start": 0.0, "env_id": "ALE-KungFuMaster-v5"}

10, 450.0, 5.42
9, 380.0, 6.21

Donde por orden de fila l es la longitude de los pasos, r la recompenza por episodio y t el tiempo de duración.

IV. RESULTADOS

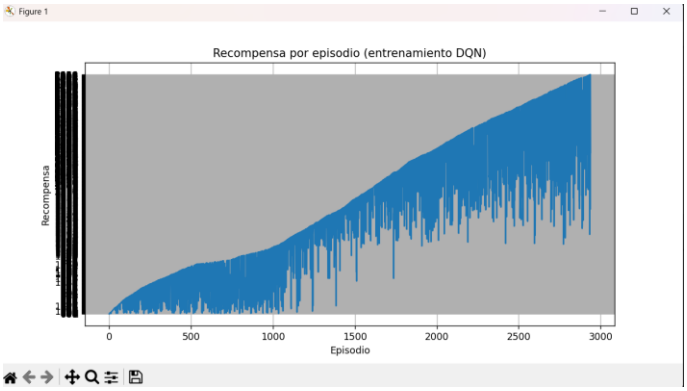
En esta sección se presentan los resultados obtenidos durante la evaluación del agente entrenado. Primero, se muestran las curvas de aprendizaje generadas a partir de los registros de entrenamiento, donde se analiza la evolución de la recompensa y la duración de los episodios a lo largo del tiempo.

Finalmente, se presentan capturas e imágenes del agente interactuando con el entorno, proporcionando evidencia visual de su comportamiento durante la ejecución.

4.1 Gráficos de aprendizaje

4.1.1 Recompensa por episodio

En la siguiente gráfica se muestra la evolución de la recompensa obtenida durante cada episodio del entrenamiento. Se observa que, a medida que aumenta el número de episodios, la recompensa promedio también incrementa, lo que indica que el agente mejora progresivamente su aprendizaje.



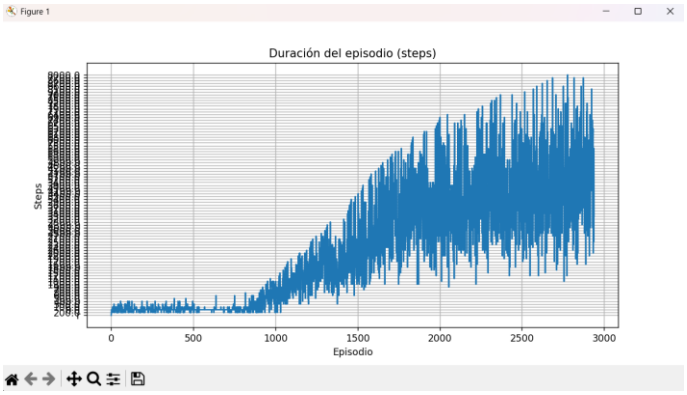
Como ya sabes que recibe 100 de recompensa cuando le pega de forma correcta al ladron, entonces conforme aumentan los episodios va mejorando en kung fu.

4.1.2 Duración por episodio

En esta gráfica se observa la evolución de la duración de cada episodio, medida en número de *steps*. Al inicio del entrenamiento, los episodios son muy cortos, eso quiere decir que el agente no jugaba de forma eficiente, sin pegarle al ladron y pierde muy rapido.

Conforme avanza el entrenamiento, la duración de los episodios aumenta de manera progresiva, mostrando un patrón ascendente y más variable.

Este comportamiento sugiere que el agente va aprendiendo a sobrevivir más tiempo en el entorno, lo cual es un indicador positivo del proceso de aprendizaje.



4.2 Métricas cuantitativas de desempeño

Basandonos en las gráficos y analizandolas más a fondo Podemos obtener las siguientes metricas:

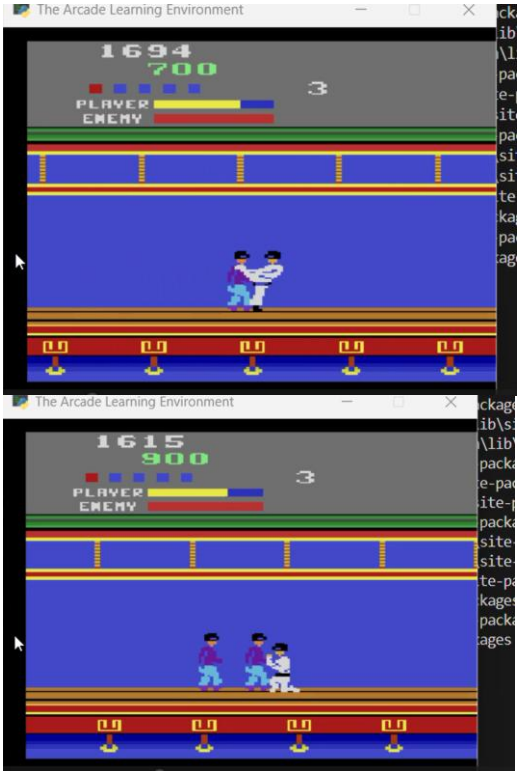
Recompensa más pequeña: 700

Recompense más grande: 3160

Recompensa promedio : 1600

4.3 Agente jugando

A contuación se muestran algunas capturas del agente jugando:



V. Discusión

Los resultados obtenidos muestran una tendencia clara: a medida que el agente acumula más episodios de entrenamiento, su desempeño mejora de manera consistente.

Esto se observa tanto en el aumento progresivo de la recompensa como en la mayor duración de los episodios, lo cual indica que el agente sobrevive más tiempo y toma decisiones más eficientes dentro del entorno.

En las primeras etapas, el agente presenta recompensas bajas y episodios muy cortos, lo que es esperable debido a su fase exploratoria inicial. Sin embargo, conforme avanza el entrenamiento, se aprecia una curva de aprendizaje ascendente, con fluctuaciones propias del uso de técnicas como epsilon-greedy y del carácter estocástico del entorno.

Esta tendencia confirma que el modelo sí está aprendiendo patrones útiles y generalizando estrategias.

Aun así, los resultados también revelan ciertas limitaciones. El entrenamiento es relativamente lento y Ruidoso (2 o 6 horas) , en parte debido a la magnitud de la búsqueda Q y al número elevado de pasos necesarios para que el agente converja hacia una política más estable.

Además, las restricciones computacionales del equipo (memoria, GPU ausente o limitada, velocidad de procesamiento) impactan directamente la calidad y el ritmo de aprendizaje: con hardware más potente o con implementaciones optimizadas, es probable que el agente pudiera alcanzar un desempeño superior o converger en menos tiempo.

VI. Conclusiones y trabajo futuro

Los resultados obtenidos permiten concluir que el modelo dqn sí logró aprender una política efectiva dentro del entorno, aunque el proceso de entrenamiento fue considerablemente lento.

Este proyecto representa la primera vez que implemento un modelo cuyo entrenamiento toma varias horas, pero el progreso es evidente: al inicio el agente no obtenía prácticamente recompensas, mientras que conforme avanzaron los episodios su desempeño mejoró de manera continua.

A pesar de las limitaciones computacionales, el agente mostró una tendencia clara de aprendizaje, aumentando tanto sus recompensas como la duración de los episodios, lo cual confirma que el modelo es capaz de adquirir comportamientos más eficientes con suficiente experiencia.

A partir de este proyecto surgen diversas oportunidades de mejora. Una de ellas consiste en entrenar el modelo en un equipo de cómputo con mayor capacidad, lo que permitiría aumentar considerablemente el número de episodios pasando de un entrenamiento de 1,000,000 a 10,000,000 o más.

Analizar cómo escala el desempeño del agente en términos de estabilidad y calidad de la política aprendida. También sería valioso ajustar y comparar distintos hiperparámetros, como incrementar el tamaño de la red neuronal, modificar el *batch size* o experimentar con diferentes valores de *learning rate*, con el fin de evaluar su impacto en la velocidad de convergencia y en la eficiencia del aprendizaje.

Finalmente, un trabajo futuro especialmente enriquecedor sería desarrollar el entorno y la implementación completa desde cero, sin depender de librerías de alto nivel, lo que implicaría construir manualmente la red neuronal, diseñar funciones de preprocesamiento para redimensionar imágenes de forma óptima y programar toda la lógica del juego.

Este enfoque permitiría comprender en mayor profundidad el funcionamiento interno de cada componente del algoritmo y fortalecer la base conceptual del aprendizaje por refuerzo.

VII. Referencias

[1] *What is reinforcement learning? - hugging face deep RL course*. (s/f). Huggingface.co. Recuperado el 26 de noviembre de 2025, de https://huggingface.co/learn/deep-rl-course/unit1/what-is-rl

[2] (S/f). Medium.com. Recuperado el 27 de noviembre de 2025, de https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae

[3] Librería RL Baselines3 Zoo recuperado 26 de noviembre 2025 https://stable-baselines3.readthedocs.io/en/master/guide/rl_zoo.html

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. En *arXiv [cs.LG]*. http://arxiv.org/abs/1312.5602