

Introducción:

El Aprendizaje Reforzado representa un segmento especializado de la Inteligencia Artificial que se dedica al estudio y desarrollo de métodos para que entidades computacionales, denominadas agentes, optimicen su selección de acciones con el fin de elevar la sumatoria de beneficios obtenidos a lo largo de su interacción con un entorno. Esta disciplina se caracteriza por la experimentación y adaptación constante del agente, permitiendo que, mediante la prueba y error, se alcance una mejora progresiva en su desempeño a partir de las experiencias acumuladas.

Conceptos Fundamentales:

- Agente: Se refiere a la entidad computacional que ejecuta acciones en función de lo que percibe de su contorno.
- Entorno: Es el contexto o escenario en el que el agente opera, definido por un conjunto de reglas o un modelo específico.
- Estado: Representa la situación o configuración actual del entorno que el agente evalúa para tomar decisiones.
- Acciones: Son las diversas opciones o movimientos que el agente puede ejecutar.
- Recompensa: Constituye la señal inmediata que recibe el agente tras ejecutar una acción, cuyo total busca maximizar.

El ciclo de aprendizaje en RL se desarrolla en iteraciones, en las que el agente inspecciona el estado vigente, selecciona y efectúa una acción guiada por una política preestablecida, obtiene una recompensa y recibe nueva información del entorno que le indica las consecuencias de sus acciones.

Modelos Decisorios:

- Modelo Basado en Valor: Se enfoca en calcular el potencial de cada posible acción dentro de un estado específico, considerando el potencial como un pronóstico del total de recompensas futuras.
- Modelo Basado en Políticas: Aprende directamente cuál es la acción más adecuada sin tener que estimar valores previamente.
- Modelo de Aprendizaje Actor-Crítico: Este enfoque híbrido utiliza dos componentes: uno, el 'actor', se encarga de seleccionar las acciones, y el otro, el 'crítico', las evalúa y califica.

Explicación breve del código 18_reinforcement_learning

Para este proyecto, primero se analizó un código complejo de Aprendizaje por Refuerzo, que implementa un escenario clásico conocido como "CartPole". Esta tarea implica controlar un carrito en dos dimensiones para mantener un poste verticalmente equilibrado sobre él. Es un problema de control clásico que sirve como banco de pruebas para algoritmos de IA.

El entorno "CartPole" es una representación simplificada de un desafío de control real. En este entorno virtual, el agente (el carrito) aprende a moverse hacia la izquierda o hacia la derecha para evitar que el poste caiga. El código se inicia asegurándose de que se dispone de la versión correcta de Python y TensorFlow, estableciendo así una base sólida y libre de incompatibilidades. La visualización de la tarea se maneja mediante matplotlib, una biblioteca de Python que permite la creación de gráficos estáticos, animados e interactivos.

El proceso de entrenamiento del agente se realiza a través de la interacción con el entorno proporcionado por gymnasium, una biblioteca que ofrece una colección de entornos de prueba para desarrollar y comparar algoritmos de Aprendizaje por Refuerzo. Cada paso que da el agente en el entorno genera una nueva observación (estado), una recompensa (indicando qué tan buena fue la acción) y una señal que indica si la tarea se ha completado o no.

El código hace uso de políticas simples y luego transita a políticas basadas en redes neuronales para tomar decisiones más informadas. Inicialmente, el agente sigue una política básica que simplemente decide las acciones basándose en el ángulo del poste. A medida que el agente experimenta y aprende, recopila las recompensas y ajusta sus acciones para maximizar la recompensa total, lo que, idealmente, conduce a una mejora en la tarea de mantener el poste en equilibrio.

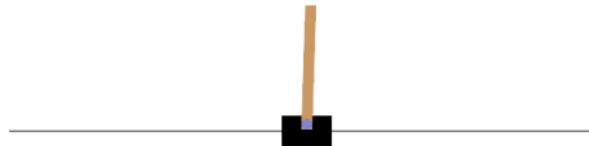
Más adelante, el enfoque cambia al uso de un modelo de red neuronal construido con TensorFlow y Keras, una API de alto nivel para crear modelos de aprendizaje automático. El modelo predice la probabilidad de que moverse a la izquierda sea la mejor acción y decide basándose en un número aleatorio y esa probabilidad. Esto introduce elementos de aleatoriedad y exploración, fundamentales en el Aprendizaje por Refuerzo, ya que permiten al agente descubrir y aprender de nuevos estados que no había experimentado antes.

Para reforzar el aprendizaje, se utiliza un proceso conocido como "gradientes de política", que ajusta las probabilidades de las acciones para incrementar la recompensa total esperada.

Cada episodio de entrenamiento produce un conjunto de recompensas y gradientes que se normalizan y descuentan antes de aplicar los gradientes al modelo.

El entrenamiento se visualiza a través de animaciones que muestran cómo el agente mejora su comportamiento a lo largo de los episodios. Se observa cómo el agente inicialmente deja caer el poste rápidamente, pero con el tiempo, aprende a balancearlo por períodos más largos.

Además de la aplicación práctica de la teoría de Aprendizaje por Refuerzo, este código proporciona una visión clara de cómo los sistemas de IA aprenden de una manera que imita, en cierto grado, el proceso de prueba y error similar al aprendizaje humano. El código final demuestra un agente que, tras miles de iteraciones, es capaz de mantener el poste equilibrado la mayoría de las veces, evidenciando el poder del aprendizaje reforzado y su potencial en tareas de control y decisiones autónomas.



Objetivo del proyecto:

Siguiendo las directrices establecidas para el proyecto, el objetivo es abordar el experimento de Cart-Pole utilizando técnicas de Aprendizaje por Refuerzo. El entorno CartPole v1 es eminentemente sencillo, compuesto por un carro que puede moverse hacia la izquierda o hacia la derecha y un poste situado verticalmente sobre él. La tarea del agente es controlar el carro de tal manera que el poste se mantenga erguido, evitando que se incline más allá de un ángulo crítico que provocaría su caída.

Para lograr este objetivo, el primer paso es **crear el entorno CartPole**. Esta simulación en 2D presenta un desafío de control clásico en el cual se acelera el carro hacia un lado o hacia otro para mantener el equilibrio del poste. Comprender la dinámica de este entorno es crucial, ya que establece las bases para el desarrollo de las políticas de control que el agente empleará.

El proceso se desglosa en varios pasos fundamentales. Inicialmente, debemos **ejecutar una política codificada de manera sencilla**, que puede considerarse como una solución inicial básica al problema. Posteriormente, se avanzará hacia políticas más complejas que

involucran redes neuronales, donde la toma de decisiones se basa en la predicción de probabilidades de acción proporcionadas por un modelo de aprendizaje profundo.

Además de estas políticas basadas en redes neuronales, se explorará el uso **de gradientes de política**, que optimizan directamente la política de acciones que el agente debe tomar. A través de estos métodos, el agente no solo aprende qué acción es la mejor en un estado dado, sino cómo mejorar su selección de acciones para maximizar la recompensa total a lo largo del tiempo.

El experimento no termina con el dominio de las políticas basadas en redes neuronales. Como un paso adicional, y siguiendo el paradigma del aprendizaje por refuerzo, se realizará el mismo experimento utilizando el algoritmo de Q-learning. Este algoritmo es importante para entender en cuántas iteraciones el carro aprende a mantener el poste en una posición vertical sin la ayuda de una política predefinida. Q-learning, que es un método basado en el valor, permitirá al agente aprender a partir de la experiencia, ajustando sus acciones en función de las recompensas acumuladas para alcanzar la meta.

Explicación del código y resultados (Primera parte)

El fragmento de código presentado demuestra la implementación de una política simple en un entorno de aprendizaje por refuerzo, específicamente utilizando el entorno "CartPole-v1" de Gymnasium. La sección de código abarca desde la configuración del entorno hasta la ejecución y evaluación de una política básica. A continuación, se desglosa cada parte del código para una comprensión detallada:

Configuración Inicial y Creación del Entorno:

```
import numpy as np
import gymnasium as gym
import matplotlib.pyplot as plt
import matplotlib.animation
from pathlib import Path

IMAGES_PATH = Path() / "images" / "rl"
IMAGES_PATH.mkdir(parents=True, exist_ok=True)
```

Esta sección importa las bibliotecas necesarias y configura una ruta para almacenar imágenes. numpy se utiliza para operaciones matemáticas, gymnasium (alias gym) para el entorno de aprendizaje por refuerzo, y matplotlib para visualizaciones.

Creación del Entorno CartPole

```
env = gym.make("CartPole-v1", render_mode="rgb_array")
```

Aquí, creo el entorno "CartPole-v1" utilizando Gym. Este entorno simula un péndulo invertido sobre un carro que el agente debe equilibrar. `render_mode="rgb_array"` permite la representación visual del entorno.

Definición de una Política Simple

```
def basic_policy(obs):  
    angle = obs[2]  
    return 0 if angle < 0 else 1
```

Luego defino una política muy básica donde la acción depende del ángulo del poste (`obs[2]`). Si el ángulo es menor que cero, la acción es 0 (mover el carro a la izquierda), y si no, es 1 (mover a la derecha).

Ejecución y Evaluación de la Política

```
totals = []  
for episode in range(500):  
    episode_rewards = 0  
    obs, info = env.reset(seed=episode)  
    for step in range(200):  
        action = basic_policy(obs)  
        obs, reward, done, truncated, info = env.step(action)  
        episode_rewards += reward  
        if done or truncated:  
            break  
    totals.append(episode_rewards)
```

Cálculo de Estadísticas

```
np.mean(totals), np.std(totals), np.min(totals), np.max(totals)
```

Finalmente, se calculan y presentan estadísticas como la media, desviación estándar, mínimo y máximo de las recompensas totales obtenidas en todos los episodios. Estas métricas proporcionan una visión general del rendimiento de la política implementada.

Primeros resultados

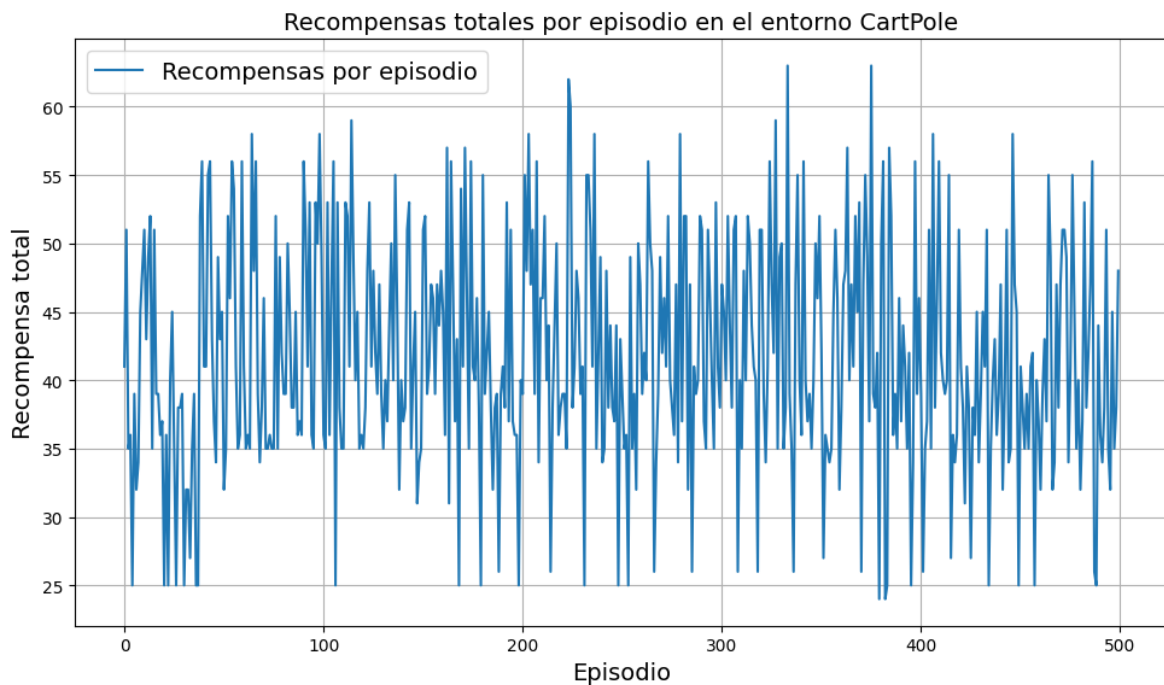
Recompensa promedio: **41.698**. La política básica fue capaz de mantener el poste vertical durante aproximadamente 41.698 pasos antes de que el episodio terminara. En el contexto del CartPole, un "paso" es un intervalo de tiempo durante el cual el agente toma una acción basada en la observación del estado actual.

Desviación estándar: **8.389**. La desviación estándar es una medida de la cantidad de variación o dispersión de un conjunto de valores. Una desviación estándar baja indica que los valores tienden a estar cerca del promedio (media) del conjunto, mientras que una desviación estándar alta indica que los valores están dispersos en un rango más amplio. En este caso,

significa que hubo algunas variaciones en la duración de cada episodio, pero no excesivamente grandes.

Mínimo de recompensa: **24**. Este es el número más bajo de pasos que el agente logró mantener el poste vertical en cualquiera de los episodios. Esto de un episodio particularmente difícil y el resultado de cierta aleatoriedad en la inicialización del entorno.

Máximo de recompensa: **63**. Este es el número más alto de pasos durante los cuales el agente fue capaz de mantener el poste vertical. Esto muestra el mejor caso en tus 500 episodios.



Decho esto, lo podemos visualizar en la siguiente gráfica estilo serie de tiempo, en la cual, tiene movibiliba con forme avanzan los episodios, mateniendo una linealidad de recompensas normal, ciertos picos superar el umbral de 60 recompensas, pero de ahí en fuera muestra un desempeño bueno.

Al fin de acabo, muestran que la política simple tiene un rendimiento bastante básico. El CartPole es considerado "resuelto" cuando el promedio de las recompensas es mayor o igual a 195.0 durante 100 episodios consecutivos, por lo que aún hay bastante margen de mejora.

Explicación del código (segunda parte)

Ahora, la siguiente implementación como parte del proyecto, es un algoritmo de aprendizaje por refuerzo, utilizando el entorno "CartPole-v1" y una red neuronal simple. Aquí analizamos los aspectos clave de este proceso:

Configuración Inicial y Red Neuronal

Pérdida y Optimizador: Se define una función de pérdida de entropía cruzada binaria y un optimizador Adam, ambos fundamentales para el entrenamiento de la red neuronal. La tasa de aprendizaje se fija en 0.01, balanceando la rapidez del aprendizaje con la estabilidad.

Reproducibilidad: La configuración de las semillas aleatorias para NumPy y TensorFlow asegura la consistencia en los resultados del experimento, que ayuda a reproducir exactamente el mismo comportamiento en múltiples ejecuciones.

Arquitectura de Red Neuronal: Se construye una red neuronal secuencial con dos capas densas. La primera capa, con activación ReLU, procesa los estados de entrada (con cuatro características cada uno), y la segunda capa, con activación sigmoide, genera una probabilidad que decide la acción del agente.

Proceso de Entrenamiento

Política de Gradientes: Se implementa una política de gradientes, que utiliza la red neuronal para tomar decisiones en base al estado actual del entorno. La predicción del modelo determina la probabilidad de tomar una acción específica.

Juego por Pasos: La función `play_one_step` ejecuta un paso del entorno, calcula la pérdida y los gradientes, y actualiza el estado y la recompensa para el aprendizaje del modelo.

Episodios Múltiples: Se juegan múltiples episodios (`play_multiple_episodes`), recolectando recompensas y gradientes para cada paso en cada episodio para el entrenamiento posterior del modelo.

Descuento y Normalización: Las recompensas son descontadas y normalizadas, ayuda a estabilizar el aprendizaje al dar más importancia a las recompensas inmediatas y reducir la variabilidad entre episodios.

Visualización y Evaluación

Animación: La función `show_policy_behavior` visualiza el comportamiento del modelo entrenado. La animación resultante muestra cómo el agente toma decisiones en el entorno "CartPole-v1" basado en la política aprendida.

Evaluación del Rendimiento: El rendimiento del modelo se evalúa calculando la recompensa media obtenida por episodio, que proporciona una medida clara de qué tan bien el modelo ha aprendido a equilibrar el poste en el carrito.

Segundos resultados

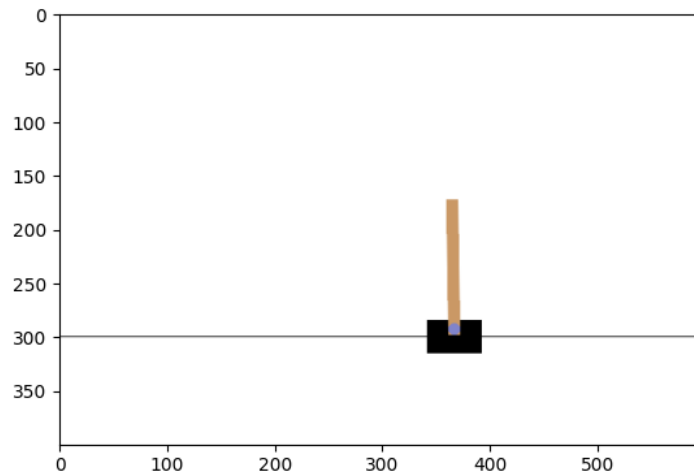
```
obs_tensor: tf.Tensor([[1.9895524  0.92299545 0.13452065 0.3585663 ]], shape=(1, 4), dtype=float32)
obs_tensor: tf.Tensor([[2.0080123  1.1159742  0.14169197 0.11114322]], shape=(1, 4), dtype=float32)
obs_tensor: tf.Tensor([[2.0303316  0.9191362  0.14391483 0.4449595 ]], shape=(1, 4), dtype=float32)
obs_tensor: tf.Tensor([[2.0487144  1.11196   0.15281403 0.20087881]], shape=(1, 4), dtype=float32)
obs_tensor: tf.Tensor([[ 2.0709536  1.3046038  0.1568316  -0.03996667]], shape=(1, 4), dtype=float32)
obs_tensor: tf.Tensor([[2.0970457  1.1076212  0.15603226 0.2978006 ]], shape=(1, 4), dtype=float32)
obs_tensor: tf.Tensor([[2.119198  0.91065913 0.16198827 0.6353421 ]], shape=(1, 4), dtype=float32)
Iteration: 150, mean rewards: 187.8
Estado original: [-0.0442396 -0.02179076  0.03727341  0.02134386]
Forma de obs_array_reshaped: (1, 4)
1/1 [=====] - 0s 84ms/step
Estado original: [-0.04467541 -0.21742688  0.03770028  0.32555005]
Forma de obs_array_reshaped: (1, 4)
1/1 [=====] - 0s 17ms/step
Estado original: [-0.04902395 -0.02286142  0.04421129  0.04499051]
Forma de obs_array_reshaped: (1, 4)

1/1 [=====] - 0s 17ms/step
Estado original: [-0.04948118  0.1715996  0.0451111  -0.2334221 ]
Forma de obs_array_reshaped: (1, 4)
```

Lo que observamos aquí, primero, es la cantidad de impresiones de observación que se toma la red neuronal, esto es parte del proceso de entrenamiento en el aprendizaje por refuerzo, donde el modelo interactúa con el entorno en múltiples episodios y pasos.

La salida "Iteration: 150, mean rewards: 187.0" nos da como resultado que, en promedio, el modelo obtuvo una recompensa total de 187 en los episodios de entrenamiento. En el contexto del problema del CartPole, significa que el modelo ha aprendido a mantener el poste en equilibrio durante los 187 pasos, 13 pasos menos que el de total que es de 200, que es el máximo para este entorno en su configuración estándar. Mostrando un desempeño muy bueno del modelo.

La única desventaja que tenemos aquí es que los tiempos de ejecución pueden ser un poco extensos, para esta parte, se tomo 15 min en ejecutarse, y todo dependerá de las iteraciones que le coloquemos.



Explicación del

código y resultados (Tercera parte)

Ahora, como parte última del proceso, haremos la misma implementación del algoritmo de aprendizaje por refuerzo, pero con el método Q-learning en el entorno "CartPole-v1". Se utiliza una técnica de discretización para convertir el espacio de estado continuo del entorno en un espacio discreto más manejable, facilitando el uso de la tabla Q para almacenar y actualizar los valores Q durante el entrenamiento.

Entorno CartPole: Se crea una instancia del entorno "CartPole-v1" para simular la tarea de equilibrar un poste en un carrito en movimiento.

Discretización del Espacio de Estado: El espacio de estado continuo se divide en una serie de "cubetas" (buckets), que representa un estado continuo como un conjunto de estados discretos. Esta técnica es crucial para aplicar Q-learning en entornos con estados continuos.

Tabla Q: Se inicializa una tabla Q con dimensiones basadas en el número de cubetas y acciones disponibles. La tabla almacena los valores Q para cada par estado-acción.

Función discretize: Esta función convierte un estado continuo en su equivalente discreto, usando los límites definidos y las proporciones correspondientes.

Proceso de Aprendizaje Q-learning

Bucle de Entrenamiento: Durante 10,000 episodios, el agente interactúa con el entorno, tomando decisiones basadas en la política actual y actualizando la tabla Q en función de las recompensas recibidas y las estimaciones futuras de los valores Q.

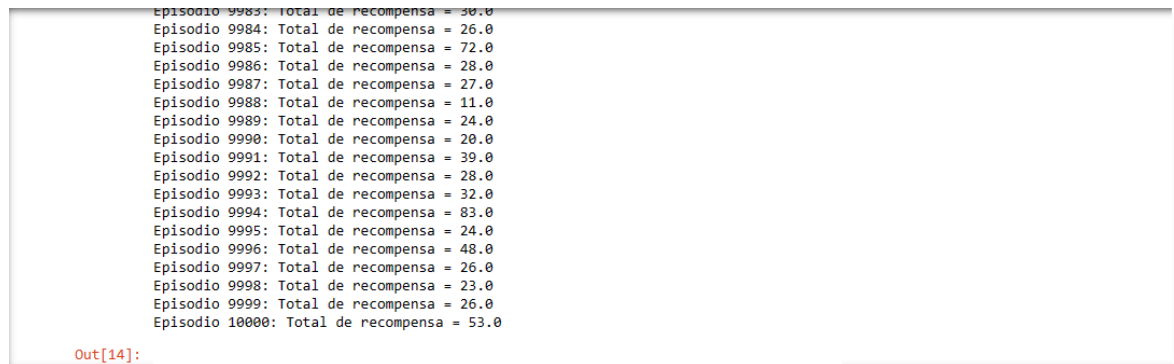
Actualización de Q-values: En cada paso del episodio, se actualizan los valores Q utilizando la fórmula del Q-learning, que incluye un factor de descuento (gamma) y una tasa de aprendizaje (alpha).

Visualización y Evaluación

Función show_policy_behavior: Se utiliza para generar una animación que muestra cómo el agente entrenado actúa en el entorno según la política aprendida.

Resultados

Resultados: La animación muestra la capacidad del agente de mantener el equilibrio del poste, indicando la efectividad del proceso de aprendizaje. Las recompensas totales por episodio dan una idea del rendimiento del agente a lo largo del tiempo.

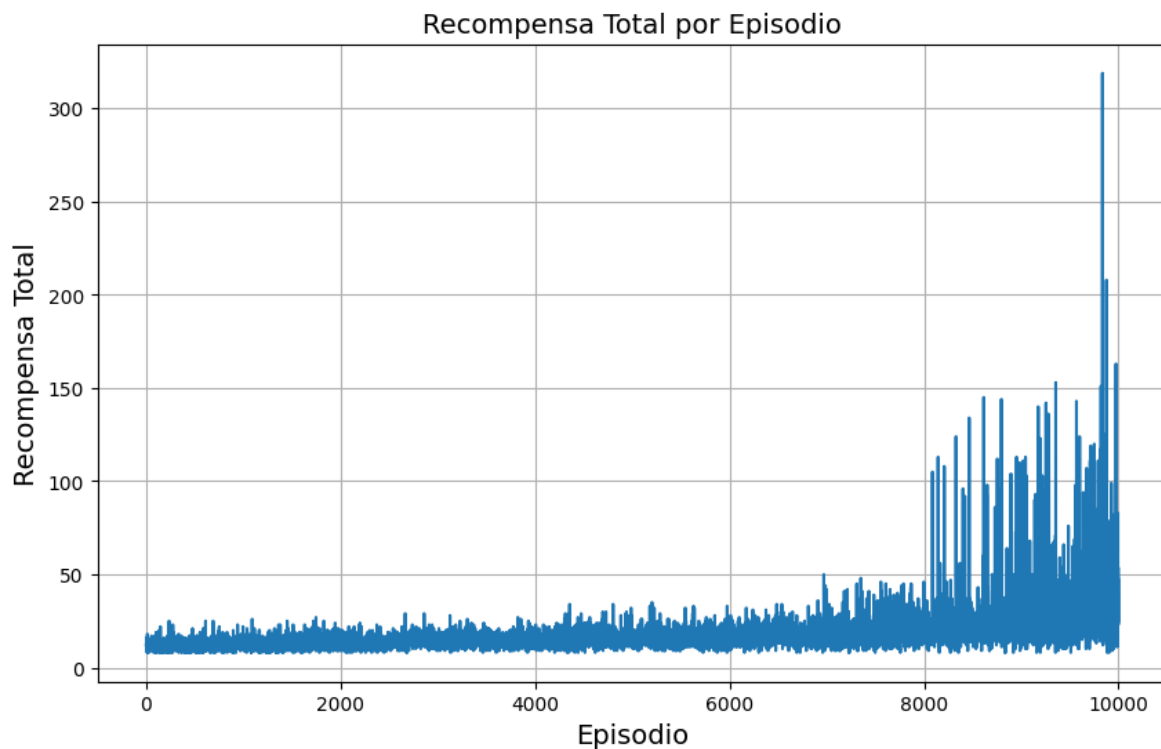


```
Episodio 9983: Total de recompensa = 30.0
Episodio 9984: Total de recompensa = 26.0
Episodio 9985: Total de recompensa = 72.0
Episodio 9986: Total de recompensa = 28.0
Episodio 9987: Total de recompensa = 27.0
Episodio 9988: Total de recompensa = 11.0
Episodio 9989: Total de recompensa = 24.0
Episodio 9990: Total de recompensa = 20.0
Episodio 9991: Total de recompensa = 39.0
Episodio 9992: Total de recompensa = 28.0
Episodio 9993: Total de recompensa = 32.0
Episodio 9994: Total de recompensa = 83.0
Episodio 9995: Total de recompensa = 24.0
Episodio 9996: Total de recompensa = 48.0
Episodio 9997: Total de recompensa = 26.0
Episodio 9998: Total de recompensa = 23.0
Episodio 9999: Total de recompensa = 26.0
Episodio 10000: Total de recompensa = 53.0
```

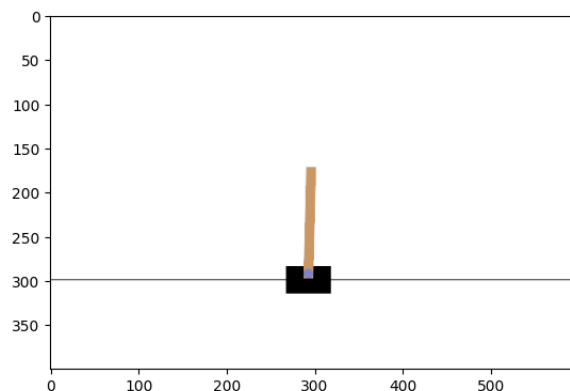
Out[14]:

Cada línea, como "Episodio 9992: Total de recompensa = 28.0", indica el total de recompensas acumuladas que el agente logró en un episodio específico, la variación en las recompensas totales de un episodio a otro da una idea de cómo el agente experimenta con diferentes estrategias para maximizar la recompensa.

Tendencias a Largo Plazo: Al observar los resultados de episodios cercanos al final del entrenamiento (como los episodios 9992 a 10000), podemos inferir si el agente está mejorando en el tiempo. Si las recompensas aumentan o se mantienen consistentes, diciendo que el agente está aprendiendo una política efectiva para mantener el equilibrio del poste en el carrito.



Desafíos del Aprendizaje por Refuerzo: La fluctuación en las recompensas totales también destaca un aspecto inherente al aprendizaje por refuerzo: el balance entre exploración (probar nuevas acciones) y explotación (usar lo aprendido para obtener la máxima recompensa).



Haciéndolo más visual, podemos observar la siguiente grafica de la recompensa total por episodio de la cual se llegó entrenó, mostrando una dispersión muy lineal a partir del episodio 0 al episodio 8000 que no pasa de un arriba de 30 de la recompensa total, a partir del 8000 al 1000, la recompensa total se dispara, mostrando que alrededor del episodio 8000 para arriba,

las recompensas empiezan a ser consistentemente altas, lo que da como resultado que el agente ha aprendido a mantener el poste en posición vertical de manera más eficaz.

Desafíos

Los desafíos primero estuvieron en las importaciones de las librerías, ya que teníamos que usar específicamente versiones que ya las últimas no tenían implementadas este proceso, además, los tiempos de ejecución fueron una lucha para enfrentarse, más que nada con el código original, este si se ejecutaba en colab, pedía una suscripción para más memoria y si se ejecutaba en jupyter no podía correrse ya que necesitaba versiones específicas de librerías, de ahí en fuera, no creo que haya habido más, las implementaciones conforme a lo que se pedía la actividad, las respuestas estaban en el código completo proporcionado, solamente se hicieron modificaciones externas para correr individualmente.

Conclusión

En el método Actor-Crítico, que se puede explorar en TensorFlow , se trabaja con dos componentes clave: el actor, que propone acciones basadas en el estado actual, y el crítico, que evalúa estas acciones. Estos métodos son parte del aprendizaje de diferencias temporales y utilizan una red neuronal para representar tanto la función de política (acciones) como la función de valor (crítico). Siendo eficiente para problemas con estados y acciones continuos, evitando la necesidad de un espacio de estado discreto y complejo.

Por otro lado, el Q-learning, que es un método de control basado en valores, trabaja con una tabla de valores Q para tomar decisiones. Un tutorial práctico sobre cómo balancear un sistema CartPole utilizando Q-learning está disponible en ar5iv . Aunque el Q-learning es poderoso, se enfrenta al problema de la "maldición de la dimensionalidad", especialmente cuando los estados y las acciones son continuos o de alta dimensión. El Deep Q-learning aborda esto utilizando una red neuronal para aproximar la función de valor Q.

En el contexto del entorno CartPole, los métodos Actor-Crítico y Deep Q-learning han demostrado ser efectivos para aprender políticas que mantengan el poste en posición vertical mientras se mueve el carro. Cada enfoque tiene sus ventajas y desventajas, y la elección depende de factores como la complejidad del entorno, la continuidad del espacio de estado y acción, y la disponibilidad de recursos computacionales.

Referencias

TensorFlow Agents. (s. f.). *Train a Deep Q Network with TF-Agents*. Recuperado de [tensorflow.org](https://www.tensorflow.org/agents). Este recurso proporciona una guía completa sobre cómo entrenar una Red Neuronal Profunda Q (DQN) utilizando TF-Agents. Explica la configuración del entorno de aprendizaje por refuerzo, la creación de un agente DQN, y cómo este puede ser entrenado y evaluado en entornos estándar como CartPole.

[handson-ml3/18_reinforcement_learning.ipynb at main · ageron/handson-ml3 \(github.com\)](https://github.com/ageron/handson-ml3/blob/main/handson-ml3/18_reinforcement_learning.ipynb)