UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Deep Learning y Sistemas Inteligentes - Laboratorio 10 -

Semestre II - 2023

Instrucciones:

- Deben estar unido a uno de los grupos de Canvas de nombre “Laboratorio 9,10 # - N”, donde N es un número entre 1 y 15. Los grupos pueden ser de máximo 5 personas.
- Esta actividad debe realizarse en grupos.
- Sólo es necesario que una persona del grupo suba el trabajo a Canvas.
- No se permitirá ni se aceptará cualquier indicio de copia. De presentarse, se procederá según el reglamento correspondiente.

Task 1 - Práctica

Para esta parte estarán resolviendo el problema de CartPole con Deep Q-Learning y una red de destino. Para esto, el objetivo de este ejercicio es entrenar a un agente para que equilibre un poste en un carro (cartpole) en movimiento durante el mayor tiempo posible. Se deberá usar Deep Q-Learning (DQL) con una red objetivo para lograr esto. Para realizar este ejercicio necesitará:

1. Python con las bibliotecas necesarias, incluidas Gymnasium, NumPy y PyTorch (para este caso, pueden usar otro framework de Deep Learning si no se sienten cómodos con PyTorch).
2. El entorno CartPole proporcionado por Gymnasium.

Considere las siguientes instrucciones generales para realizar este ejercicio:

1. **Librerías**: Asegúrese de tener instalado Gymnasium, NumPy y el framework de Deep Learning que haya elegido.
2. **Cree el entorno CartPole**: Utilice la biblioteca Gymnasium para crear el entorno CartPole. Este entorno simula la tarea de equilibrar un poste en un carro en movimiento. :
3. **Definan las redes en línea y de destino**: Cree dos redes neuronales, la red en línea y la red de destino. La red en línea se utiliza para la selección de acciones y se actualiza con más frecuencia, mientras que la red de destino se utiliza para estimar los valores Q y se actualiza periódicamente. Ambas redes deberían tener una arquitectura similar con capas de entrada y salida. Inicialmente, la red de destino debería tener los mismos pesos que la red en línea.
4. **Establecer hiperparámetros**: Defina hiperparámetros como el número de episodios, el tamaño de los batches, el factor de descuento (gamma) y los parámetros de exploración (epsilon, epsilon decay, epsilon mínimo). Ajuste estos hiperparámetros según sea necesario para optimizar el entrenamiento.
5. **Defina la selección de acciones épsilon-greedy**:: Cree una función para la selección de acciones épsilon-greedy. Esta función ayuda al agente a elegir acciones basadas en la política épsilon-greedy.
6. **Defina la reproducción de la experiencia (experience replay)**: Implemente una función para la reproducción de la experiencia, que es una parte crucial de DQL. Esta función ayuda al agente a aprender de una memoria de repetición y a estabilizar el entrenamiento.
7. **Ciclo de entrenamiento**: Cree un ciclo para el entrenamiento del agente. En cada episodio, el agente interactúa con el entorno, recopila experiencias y actualiza sus valores Q mediante la repetición de experiencias (experience replay). La red de destino se actualiza cada N episodios.
8. **Representar el entorno**: Para visualizar el progreso del entrenamiento del agente, use env.render() para mostrar el entorno CartPole durante el entrenamiento. Asegúrese de llamar a env.close() al final para limpiar el renderizado.
9. **Supervisar el entrenamiento**: Supervise el progreso del entrenamiento del agente, incluida la recompensa total obtenida en cada episodio, para esto utilice una gráfica.
10. **Evalúe el rendimiento**: Una vez que se complete el entrenamiento, evalúe el rendimiento del agente probándolo en el entorno CartPole sin renderizar y observe qué tan bien puede equilibrar el poste.
11. **Fine-Tuning**: Experimente con diferentes hiperparámetros, arquitecturas y estrategias de entrenamiento para mejorar el desempeño del agente.

12. **Notas adicionales**: tenga en cuenta que los tiempos de entrenamiento pueden variar y puede ajustar la frecuencia de actualización de la red de destino según los requisitos específicos de su tarea.

Task 2 - Teoría

Defina en qué consiste y en qué clase de problemas se pueden usar cada uno de los siguientes acercamientos en Deep Reinforcement Learning

1. Proximal Policy Optimization
2. Deep Deterministic Policy Gradients (DDPG)
3. Trust Region Policy Optimization (TRPO)
4. Asynchronous Advantage Actor-Critic (A3C)

Entregas en Canvas

1. Jupyter Notebook o el script que usen para resolver el task 1, tanto en PDF como en .ipynb
2. Documento con las respuestas del task 2 en PDF
    a. Pueden responder en el mismo Jupyter Notebook si así prefieren

Evaluación

1. [4 pts.] Task 1 (4 pts)
2. [1 pts.] Task 2 (0.25 pts cada acercamiento)

Total 5 pts.