Artificial Intelligence

P24-LIS3082-1

Proyect:

Reinforcement Learning

Mariana García Rodriguez 168521

Introducción:

El Aprendizaje por Refuerzo (Reinforcement Learning) es una rama de la Inteligencia Artificial (IA) que se enfoca en cómo los agentes deberían tomar decisiones para maximizar alguna noción de recompensa acumulativa a través del tiempo. En el aprendizaje por refuerzo, un agente interactúa con su entorno, toma decisiones, comete errores y aprende de ellos con el objetivo de mejorar su comportamiento basado en la experiencia.

Conceptos Clave:

- **Agente:** Entidad que toma decisiones basadas en observaciones de su entorno.
- **Entorno:** El mundo, definido por un modelo, con el que el agente interactúa.
- **Estado**: Una representación del entorno en un momento dado. Puede ser todo el entorno o solo una parte relevante para el agente.
- **Acciones:** Todas las posibles decisiones o movimientos que el agente puede tomar.
- **Recompensa:** Una señal inmediata que el agente recibe después de tomar una acción. El objetivo del agente es maximizar la suma de estas recompensas.

El proceso de aprendizaje en RL puede ser visto como un bucle donde, en cada paso, el agente observa el estado actual del entorno, decide y realiza una acción basada en una política, recibe una recompensa y una nueva observación que refleja el cambio en el entorno debido a su acción

Modelos de Decisión

Modelo Basado en Valor: Se centra en aprender el valor de cada acción en un estado dado, donde el valor es una estimación de las recompensas futuras.

Modelo Basado en Políticas: Directamente aprende la política de acción sin necesidad de aprender los valores primero.

Modelo de Aprendizaje Crítico: Combina los dos anteriores, utilizando una estructura de "actor" para elegir acciones y una de "crítico" para evaluarlas.

Aplicaciones

El aprendizaje por refuerzo se ha aplicado con éxito en una variedad de campos, incluidos:

- **Juegos**: Desde juegos simples hasta complejos como Go, donde el programa AlphaGo de DeepMind venció a campeones humanos.
- **Robótica:** Para enseñar a los robots a realizar tareas complejas como caminar y manipular objetos.
- **Sistemas de recomendación:** Mejorando las sugerencias basadas en la interacción del usuario.
- Optimización de recursos en redes de computadoras y sistemas de distribución de energía.

El aprendizaje por refuerzo ofrece un marco poderoso para enseñar a las máquinas a tomar decisiones de manera autónoma, aprendiendo a través de la experiencia y mejorando con el

tiempo, lo que abre amplias posibilidades para aplicaciones avanzadas en inteligencia artificial.

The simple hard coded policy:

Podemos observar que este código esta diseñando para poder demostrar como se opuede aplicar una política de decisiones de aprendizaje por refuerzo usando CartPole-v1 de OpenAI Gym

Ejecución de la Política Básica:

Update scene:

Esta función actualiza la escena en la animación. Recibe el número de frame actual (num), la lista de todos los frames y un patch, que es el objeto que se está mostrando en matplotlib (en este caso, una imagen del entorno CartPole.Patch.set_data(frames[num]) actualiza la imagen mostrada con el frame actual, donde la función retorna el patch actualizado, que es necesario para la función de animación de matplotlib.

Plot animation:

Esta función crea y devuelve una animación de los frames capturados durante un episodio de simulación.Inicializa una figura de matplotlib y muestra el primer frame y desactiva los ejes con plt.axis('off') para que solo se vea la imagen. También crea una animación (FuncAnimation) que actualiza los frames llamando a update_scene con un intervalo y cantidad de repeticiones especificadas.

Por lo que cierra la figura con plt.close()para evitar que se muestre automáticamente al crearla (la animación se mostrará más adelante, cuando se solicite). Y Finalmente devuelve el objeto de animación creado.

Show one episode:

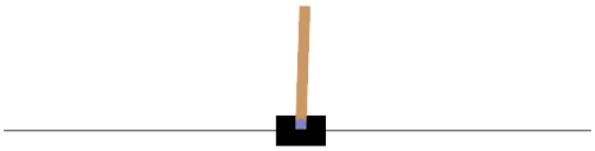
Esta función simula y muestra un episodio completo del entorno "CartPole-v1", donde vemos como nicializa una lista vacía frames para almacenar cada frame del episodio y puede crear un entorno de CartPole con un modo de renderizado que devuelve imágenes RGB de cada estado del entorno. Por lo que establece una semilla para el generador de números aleatorios para hacer reproducibles los episodios.

Reinicia el entorno a su estado inicial y comienza a simular pasos en un bucle. En cada paso, se captura el frame actual usando env.render(), se decide una acción con la política (policy(obs)), y luego se aplica esa acción al entorno con env.step(action). Si el episodio termina (done o truncated es verdadero), se rompe el bucle.Finalmente, cierra el entorno y devuelve una animación de todos los frames capturados usando plot animation.

La función show one episode requiere como argumento una función policy, que define cómo se toman decisiones en el entorno. La política recibe las observaciones del entorno (por

ejemplo, la posición y velocidad del carro, y la posición y velocidad angular del poste) y devuelve una acción (mover el carro a la izquierda o a la derecha).

El código completo es un ejemplo de cómo visualizar y entender el comportamiento de un agente en un entorno simulado, permitiendo la evaluación y mejora de políticas de decisión.



The simple hard coded policy

En este entorno, el objetivo es mantener un poste vertical equilibrado en un carro que se puede mover hacia la izquierda o hacia la derecha a lo largo de una pista horizontal.

Carro:

Es el bloque negro cuadrado en la parte inferior de la imagen. Este carro puede moverse hacia la izquierda o hacia la derecha a lo largo de la pista para intentar mantener el poste en equilibrio.

Poste:

Es la barra vertical marrón. El objetivo es evitar que el poste se caiga moviendo el carro debajo de él.

Pista:

Es la línea negra horizontal. Representa el camino sobre el que el carro puede moverse.

Dinámica del entorno:

Al inicio de cada episodio, el poste está ligeramente inclinado, y el objetivo es usar una política (una función que decide si mover el carro a la izquierda o a la derecha en función del estado actual del entorno) para mantener el poste tan vertical como sea posible durante el mayor tiempo posible.

Estado del entorno:

Generalmente incluye la posición del carro, la velocidad del carro, el ángulo del poste y la velocidad angular del poste en la parte superior.

Acciones:

En cada paso de tiempo, la política decide una acción basada en el estado actual. Las acciones posibles son mover el carro a la izquierda o a la derecha.

Recompensas:

Por cada paso de tiempo que el poste se mantiene en pie, se otorga una recompensa (generalmente +1), incentivando al agente a mantener el poste equilibrado el mayor tiempo posible.

Terminación:

El episodio termina (y se considera que el agente ha fallado) si el poste se cae demasiado (generalmente más de 15 grados desde la vertical), si el carro se mueve más allá de los límites de la pista, o si el episodio alcanza un límite de pasos de tiempo.

The Neural Network Policies:

Importa TensorFlow:

Tenemos que importa la biblioteca TensorFlow, que es una herramienta ampliamente usada para aprendizaje automático y redes neuronales. Y establece una semilla para la generación de números aleatorios Esto garantiza que la inicialización de los pesos del modelo y cualquier otro proceso aleatorio en TensorFlow sean reproducibles.

Construye un modelo de red neuronal secuencial con Keras:

Primera capa: Una capa densa con 5 neuronas y la función de activación ReLU donde las capas densas son capas de red neuronal totalmente conectadas donde cada entrada está conectada a cada neurona de la capa.

Segunda capa: Una capa densa con 1 neurona y la función de activación sigmoidal. Esta capa se utilizará para producir la probabilidad de una acción en el entorno CartPole.

Define una función de política para el aprendizaje por refuerzo (pg policy):

La función pg_policy toma una observación del entorno (obs) y la pasa a través del modelo para predecir la probabilidad de tomar la acción "izquierda". La función obs[np.newaxis] agrega una dimensión extra a obs para que tenga la forma requerida por model.predict (es decir, un batch de observaciones aunque sea de tamaño 1).Luego genera un número aleatorio entre 0 y 1. Si este número es mayor que la probabilidad predicha de ir a la izquierda, la función devuelve 1 (lo que podría representar la acción de moverse a la derecha), de lo contrario devuelve 0 (moverse a la izquierda).

Establece la semilla para NumPy:

Esto asegura la reproducibilidad de las operaciones aleatorias que se realizan con NumPy, como la generación del número aleatorio para la selección de acciones en la función pg policy.

Ejecuta un episodio en el entorno CartPole con la política definida:

La función show one episode toma la función de política pg policy y simula un episodio completo del entorno CartPole, generando una animación de este episodio. La política utiliza el modelo no entrenado para decidir las acciones, y la visualización permite ver cómo se comportaría esta política sin entrenamiento.

The Neural Network Policies

Podemos observar el entorno "CartPole-v1" el carro (la caja negra en la parte inferior) que se puede mover hacia la izquierda o hacia la derecha a lo largo de una pista, y un poste (la barra vertical marrón) que está equilibrado sobre el carro. El objetivo del juego es mover el carro de tal manera que mantenga el poste equilibrado y evite que se caiga.

La función de política pg policy utiliza un modelo de red neuronal simple para predecir la probabilidad de que la acción de moverse a la izquierda sea la mejor acción basada en la observación actual del entorno. Luego, utilizando un proceso aleatorio, decide si mover el carro hacia la izquierda o hacia la derecha.

Si se aplica la función de política pg policy al estado del entorno representado en el modelo haría una predicción basada en la posición actual y el ángulo del poste, así como la posición y la velocidad del carro. Luego tomaría una decisión basada en esa predicción y un poco de azar debido al número aleatorio generado, lo que afectaría el siguiente movimiento del carro en la simulación.

Ya que la red neuronal no se ha entrenado con datos de cómo mantener el poste en equilibrio, las acciones que toma probablemente no serán adecuadas para mantener el poste equilibrado por mucho tiempo.

Policy Gradients:

Función play one step:

Esta función realiza un paso de juego (acción) dentro del entorno de Gym utilizando el modelo de política para determinar la acción.

Grabación de gradiente:

Se usa tf.GradientTape() para registrar las operaciones para la diferenciación automática. Esto es necesario para calcular los gradientes de la función de pérdida con respecto a los parámetros del modelo más tarde.

Predicción de probabilidad:

El modelo hace una predicción (left_proba) para la observación actual (obs). Esto se interpreta como la probabilidad de que la acción de moverse hacia la izquierda sea la más beneficiosa.

Decisión de acción:

Se decide una acción (moverse a la izquierda o a la derecha) en base a la probabilidad predicha y un número aleatorio.

Cálculo de la función de pérdida:

Se calcula la función de pérdida usando la diferencia entre la acción tomada y la probabilidad predicha.

Cálculo de gradientes:

Se calculan los gradientes de la función de pérdida con respecto a las variables entrenables del modelo.

Ejecución de la acción en el entorno:

Se ejecuta la acción decidida en el entorno y se obtiene la nueva observación, recompensa y estado del juego (si el juego ha terminado o no). La función devuelve la nueva observación, la recompensa obtenida, si el juego ha terminado, los gradientes calculados y si el episodio fue truncado (corte anticipado por razones de estabilidad).

Función play multiple episodes:

Esta función juega múltiples episodios en el entorno utilizando la función play one step.

- **Inicialización de listas:** Se inicializan listas para almacenar las recompensas totales y los gradientes de cada episodio.
- **Loop de episodios:** Se juega un número determinado de episodios (`n_episodes`), y en cada episodio, se juega un número máximo de pasos (`n_max_steps`).
- **Recompensas y gradientes:** Para cada paso de un episodio, se recopilan las recompensas y los gradientes. La función devuelve las recompensas totales y los gradientes para cada episodio.

Función discount rewards:

Esta función toma una secuencia de recompensas y las descuenta hacia atrás para calcular la recompensa acumulada ajustada por un factor de descuento (discount factor).

Función discount and normalize rewards:

Esta función normaliza las recompensas descontadas calculadas por la función discount rewards.

- **Descuento de recompensas:** Calcula las recompensas descontadas para cada episodio.
- **Aplanamiento y normalización:** Aplana todas las recompensas descontadas en una sola lista, calcula la media y la desviación estándar, y normaliza las recompensas descontadas restando la media y dividiendo por la desviación estándar.

El resultado de esta función son las recompensas descontadas y normalizadas para cada paso de cada episodio, las cuales son utilizadas para ajustar los parámetros del modelo de forma que se maximicen las recompensas futuras.

Configuración Inicial

Se establece el número de iteraciones para el entrenamiento, el número de episodios para actualizar los pesos de la red neuronal, el número máximo de pasos por episodio y el factor de descuento para las recompensas futuras.

Creación del Modelo de Red Neuronal

Se crea un modelo secuencial de Keras con dos capas densas, que servirá para predecir la probabilidad de que la acción "izquierda" sea la correcta en cada paso del entorno.

Reset del Entorno

Se resetea el entorno de Gym para comenzar un nuevo episodio y se establece una semilla para la reproducibilidad.

Configuración del Optimizador y la Función de Pérdida

Se define un optimizador Nadam (una variante del optimizador Adam con Nesterov momentum) con una tasa de aprendizaje de 0.01 y se utiliza la entropía cruzada binaria como función de pérdida.

Ciclo de Entrenamiento

Para cada iteración del ciclo de entrenamiento:

1. **Jugar Múltiples Episodios:** Se juegan varios episodios utilizando la política actual del modelo, y se recopilan todas las recompensas y gradientes resultantes.
2. **Información de Depuración**: Se imprime información de depuración, como la iteración actual y la recompensa media obtenida.
3. **Normalización de Recompensas:** Se normalizan las recompensas con descuento para cada paso de cada episodio.
4. **Cálculo de Gradientes Promedio**: Se calcula el gradiente promedio para cada variable entrenable del modelo, ponderado por las recompensas normalizadas.

5. **Aplicación de Gradientes:** Se aplican estos gradientes promedio a las variables entrenables del modelo utilizando el optimizador. Esto actualiza los pesos de la red en la dirección que, en promedio, aumentará las recompensas futuras.

Sobre las Funciones discountrewards y discount and normalize rewards. Estas funciones se utilizan dentro del ciclo de entrenamiento para ajustar las recompensas que un agente recibe:

Discount rewards: toma una serie de recompensas y las descuenta para calcular la recompensa acumulada ajustada por el factor de descuento. Discount an normalize rewards normaliza las recompensas descontadas calculadas para todos los episodios

Ambas funciones son esenciales para el algoritmo REINFORCE, ya que ajustan las recompensas para tener en cuenta no solo las inmediatas sino también las futuras, promoviendo acciones que puedan no ser inmediatamente gratificantes pero que llevan a mayores recompensas a largo plazo. Además, normalizar estas recompensas ayuda a estabilizar el entrenamiento y acelerar la convergencia.

Policy Gradients

Q Learning:

Inicialización de los valores Q:

Los valores Q para todas las combinaciones de estados y acciones posibles se inicializan. Para los estados y acciones que no son posibles, los valores Q se establecen en menos infinito (np.inf) para representar que esas combinaciones de estado-acción no son viables. Para las combinaciones posibles, los valores Q se inicializan a cero.

Definición de hiperparámetros:

- Alpha0: Tasa de aprendizaje inicial. Controla cuánto se actualizan los valores Q en cada iteración.
- Decay: Tasa de decaimiento que reduce la tasa de aprendizaje alpha con el tiempo para permitir una exploración inicial más agresiva y una explotación más refinada más adelante en el entrenamiento.

- Gamma: Factor de descuento. Determina cuánta importancia se le da a las recompensas futuras en comparación con las inmediatas.

Bucle de entrenamiento:

El algoritmo itera a través de una cantidad definida de pasos (10,000 en este caso). En cada iteración:

Se registra el estado actual de los valores Q (`history2`), se selecciona una acción basada en una política de exploración, que elige aleatoriamente una acción de las posibles para el estado actual.

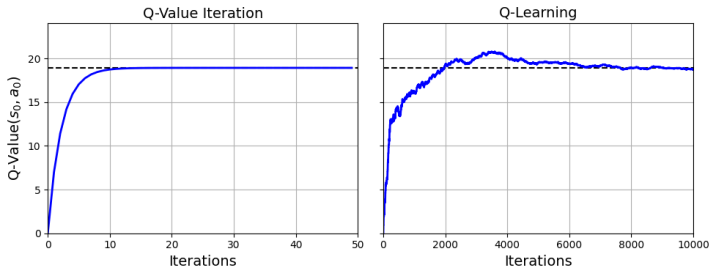
Se ejecuta la acción seleccionada en la función `step`, obteniendo el siguiente estado y la recompensa correspondiente. Y se obtiene el máximo valor Q del próximo estado, que representa la mejor recompensa esperada en el siguiente paso si se sigue una política codiciosa. Por lo que se actualiza el valor Q para el par estado-acción actual utilizando la fórmula de actualización de Q-Learning, que incluye la recompensa obtenida y el valor máximo del siguiente estado, ajustado por la tasa de aprendizaje y el factor de descuento.

La tasa de aprendizaje se ajusta en función del número de iteración y la tasa de decaimiento. El estado se actualiza al siguiente estado, y el proceso se repite.

Convergencia de los valores Q:

Con el tiempo, los valores Q convergen a estimaciones estables que reflejan el valor máximo esperado de las recompensas futuras para cada par estado-acción, dado un comportamiento óptimo.

El propósito del entrenamiento es encontrar una política óptima que le diga al agente qué acción tomar en cada estado para maximizar la suma de recompensas futuras descontadas. La política final sería una política codiciosa con respecto a los valores Q finales, seleccionando en cada estado la acción con el mayor valor Q.



Q-Learning and Q-Value Iteration

Dueling Double DQN:

Este código muestra cómo configurar y entrenar un modelo de red neuronal para un agente que actúa en un entorno de aprendizaje por refuerzo, utilizando una técnica conocida como Q-learning con función de ventaja (Advantage Function).

Definición de la Capa Personalizada

CustomAdvantageLayer: Se define una capa personalizada en Keras que calcula los valores Q de un agente a partir de dos entradas: los valores de estado y las ventajas brutas para cada acción. La ventaja se calcula restando el máximo de las ventajas brutas para cada acción, lo que ayuda a estabilizar el entrenamiento al centrar las ventajas. Los valores Q se calculan sumando los valores de estado a las ventajas centradas.

Construcción del Modelo

Se construye una red neuronal con dos capas ocultas y dos salidas separadas para los valores de estado y las ventajas brutas. La red recibe como entrada el estado del entorno (input_states).

Las dos salidas se pasan a la CustomAdvantageLayer para obtener los valores Q finales para cada acción.

Entrenamiento del Modelo

Se inicializa un buffer de repetición (replay_buffer) para implementar el aprendizaje por refuerzo basado en la experiencia (Experience Replay), que mejora la estabilidad y la eficiencia del aprendizaje.

Se establecen las variables para llevar un registro de las recompensas y el mejor puntaje obtenido (best_score).

Bucle de Entrenamiento

Se juegan múltiples episodios (600 en este caso), y en cada episodio:

- Se reinicia el entorno para obtener el estado inicial.
- Se juega el episodio, donde en cada paso se toma una acción basada en la política actual y se añade la experiencia al buffer de repetición. La variable epsilon se utiliza para explorar aleatoriamente con menor frecuencia a medida que avanzan los episodios (exploración decreciente).
- Se actualizan las recompensas y se comprueba si el puntaje del paso actual es el mejor, en cuyo caso se guardan los pesos actuales del modelo como los mejores.
- Después de los primeros 50 episodios, se llevan a cabo pasos de entrenamiento training step no está definido en este fragmento, pero implicaría tomar un batch del buffer de repetición para entrenar el modelo).
- Cada 50 episodios, los pesos del modelo objetivo se actualizan con los pesos del modelo principal para estabilizar el aprendizaje.

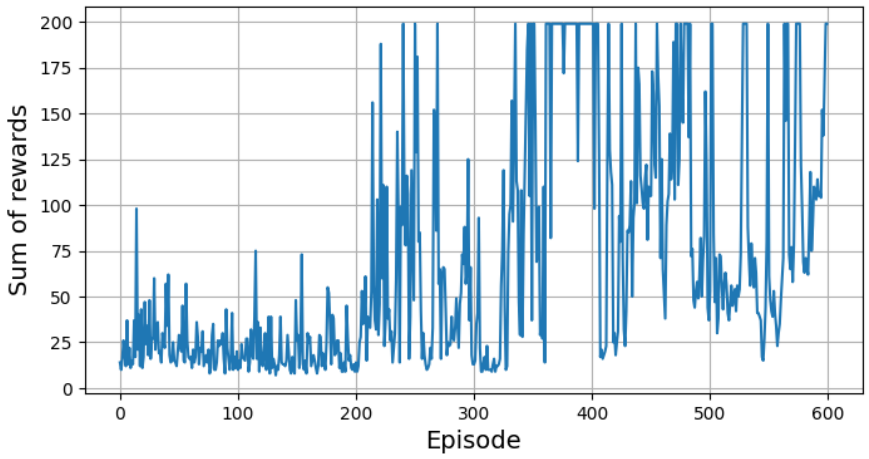
- Finalmente, se actualizan los pesos del modelo con los mejores pesos encontrados durante el entrenamiento.

Visualización

Se muestra una gráfica de la suma de recompensas obtenidas en cada episodio, lo cual proporciona una visión del desempeño y la mejora del agente a lo largo del tiempo.

Este proceso de entrenamiento es una forma de Q-learning con Separación de Ventajas, que intenta separar la estimación de los valores Q en dos partes: el valor de estar en un estado determinado (independientemente de la acción) y la ventaja (o desventaja) de tomar una acción específica en ese estado. Este enfoque a menudo mejora el rendimiento sobre los métodos de Q-learning tradicionales al reducir la varianza de la estimación de los valores Q y acelerar la convergencia hacia una política óptima.

Policy Gradients



Episode

El código ilustra cómo implementar y entrenar un modelo de red neuronal para un agente de aprendizaje por refuerzo utilizando Q-Learning con una capa de ventaja personalizada. A través de la interacción con un entorno simulado y un enfoque de exploración decreciente, el agente mejora gradualmente su política de acción para maximizar las recompensas acumuladas. El entrenamiento involucra actualizaciones periódicas de un modelo objetivo y la utilización de un buffer de repetición de experiencias para estabilizar y eficientizar el aprendizaje. La gráfica final muestra la progresión de la habilidad del agente para acumular recompensas a lo largo de los episodios, evidenciando su aprendizaje y mejora a lo largo del tiempo.