Universidad de La Coruña

Facultad de Informática

Aprendizaje por Refuerzo: Q-Learning

Xian Priego Martín
Samuel Otero Agraso

16 de Mayo de 2024

1 Introducción

El *Aprendizaje por Refuerzo* (Reinforcement Learning, RL) es una técnica de aprendizaje automático en la que un agente aprende a tomar decisiones mediante la interacción con un entorno. El objetivo del agente es maximizar una recompensa acumulada a lo largo del tiempo. El agente observa el estado actual del entorno, toma una acción y recibe una recompensa basada en esa acción. Con el tiempo, el agente aprende una política de comportamiento óptima que le permite maximizar su recompensa.

Una de las metodologías más conocidas dentro del Aprendizaje por Refuerzo es el *Q-learning*. El Q-learning es un algoritmo de aprendizaje fuera de línea que busca aprender una función de valor Q, que representa la utilidad esperada de tomar una acción específica en un estado dado y seguir la política óptima en adelante. Esta función se actualiza iterativamente utilizando la siguiente fórmula de actualización:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$$

donde:

- $Q(s,a)$ es el valor Q para el estado s y la acción a,
- α es la tasa de aprendizaje,
- r es la recompensa recibida,
- γ es el factor de descuento,
- s' es el nuevo estado después de tomar la acción a,
- $\max_{a'} Q(s',a')$ es el valor Q máximo posible en el nuevo estado s' para cualquier acción a'.

La práctica consiste en implementar estas técnicas utilizando el robot *Khepera IV* en el simulador *Webots*. El robot Khepera IV está equipado con varios sensores infrarrojos que le permiten detectar obstáculos y seguir líneas en el suelo. La tarea principal del robot en esta práctica es aprender a seguir líneas negras en el entorno simulado, maximizando así la recompensa acumulada. Por otra parte, de forma completamente ajena al proceso de aprendizaje, ejecutará un comportamiento básico definido para evitar las paredes únicamente cuando se encuentre muy cercano a ellas. Cabe destacar que este comportamiento se encuentra completamente aislado del proceso de aprendizaje y no influye en él.

2 Implementación

En esta sección se describe detalladamente la implementación del código para aplicar el algoritmo de Q-learning al robot Khepera IV en el simulador Webots. La implementación abarca desde la inicialización de las estructuras de datos hasta la gestión del proceso de aprendizaje.

2.1 Estructuras Usadas para las Matrices

Para implementar el algoritmo de Q-learning, se utilizan dos matrices principales:

- **Matriz Q:** Esta matriz tiene dimensiones 3×3, donde el número de filas representa los posibles estados y el número de columnas representa las posibles acciones. La matriz se inicializa con ceros y se actualiza a medida que el robot aprende.
- **Matriz de Visitas:** Esta matriz también tiene dimensiones 3×3 y se utiliza para contar cuántas veces se ha visitado cada par estado-acción. Ayuda a ajustar la tasa de aprendizaje (α) en la actualización de la matriz Q.

2.2 Determinación del Estado Actual

El robot puede encontrarse en uno de tres posibles estados, que se determinan mediante las lecturas de los sensores infrarrojos del suelo:

- **Estado 0:** Khepera abandona línea por la izquierda.
- **Estado 1:** Khepera abandona línea por la derecha.
- **Estado 2:** Resto de casos.

La función `get_state()` evalúa las lecturas de los sensores y asigna el estado correspondiente.

2.3 Definición de las Acciones

Se definen tres posibles acciones que el robot puede realizar:

- **Seguir Recto:** La función `a3()` hace que el robot avance en línea recta durante 0.1 segundos.
- **Girar a la Derecha:** La función `a1()` ajusta las velocidades de los motores para girar el robot a la derecha durante 0.1 segundos.
- **Girar a la Izquierda:** La función `a2()` ajusta las velocidades de los motores para girar el robot a la izquierda durante 0.1 segundos.

Estas acciones son fundamentales para que el robot pueda navegar y aprender a evitar obstáculos en su entorno.

2.4 Gestión del Proceso de Aprendizaje

El proceso de aprendizaje se gestiona mediante la probabilidad de elegir una acción óptima o aleatoria, la cual varía en función del número de iteraciones:

- **Función de Probabilidad:** La función `probability_function(i)` define una probabilidad lineal que disminuye a medida que aumentan las iteraciones. Al inicio, es más probable que el robot elija una acción aleatoria (exploración) y, con el tiempo, se favorece la elección de la acción óptima (explotación).
- **Selección de Acción:** La función `choose_action(i, state)` utiliza la probabilidad calculada para decidir si elegir una acción aleatoria o la acción con el valor Q más alto para el estado actual.

2.5 Cálculo del Refuerzo

El cálculo del refuerzo inicialmente se intentó realizar mediante una estrategia más compleja que se basaba en la siguiente fórmula:

$$reinforcement(previous\_values, current\_values) = \sum_{i=1}^{n} (p_i - c_i) \cdot w_i$$

donde:

- $previous\_values$ son los valores anteriores de los sensores,
- $current\_values$ son los valores actuales de los sensores,
- w_i son los pesos asociados a cada sensor,
- n es el número total de sensores.

En esta fórmula, se resta el valor actual del sensor del valor anterior para calcular la diferencia. Luego, esta diferencia se multiplica por el peso asociado al sensor correspondiente y se suma para obtener el refuerzo total.

Esta estrategia se diseñó para considerar de manera ponderada las diferencias entre las lecturas de los sensores infrarrojos del suelo en el cálculo del refuerzo. Sin embargo, se observó que el comportamiento del robot era más errático y, en el mejor de los casos, daba iguales resultados que la estrategia más simple. Además, el robot podía quedarse dando vueltas en las curvas cerradas, lo que indicaba un aprendizaje menos efectivo.

Debido a estos resultados, se descartó esta estrategia más compleja y se optó por utilizar la estrategia más simple, que consiste en contar el número de sensores que detectan el color negro una vez ejecutada una acción determinada, a mayor número de sensores que detectan negro mayor será el refuerzo. Esta estrategia demostró ser más efectiva para el propósito de seguir líneas negras en el entorno simulado en Webots.

2.6 Actualización de la Matriz Q

La actualización de la matriz Q se realiza utilizando la fórmula del Q-learning:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') \right)$$

donde:

- α es la tasa de aprendizaje, calculada como $\frac{1}{1+visitas[s][a]}$.
- r es la recompensa obtenida.
- γ es el factor de descuento, que pondera la importancia de las recompensas futuras.
- $\max_{a'} Q(s', a')$ es el valor Q máximo para el estado futuro s'.

La función `update_Q(s, s_prima, a, r, alpha)` se encarga de realizar esta actualización.

2.7 Separación del Comportamiento de Evitar Paredes del Proceso de Aprendizaje

Se ha implementado una separación completa entre el comportamiento de evitar paredes y el proceso de aprendizaje del algoritmo Q-learning. Esta separación se controla de dos maneras principales:

1. **Control desde el Bucle Principal:** En el bucle principal del programa, se realiza una verificación constante para detectar la presencia de paredes utilizando la función `there_is_wall()`. Si se detecta una pared, se activa el comportamiento de evitar paredes mediante la función `avoid_walls()`. Esto garantiza que el robot pueda evitar colisiones mientras continúa ejecutando el algoritmo de aprendizaje.
2. **Control durante la Ejecución de Acciones:** Durante la ejecución de cada acción (avanzar recto, girar a la derecha o girar a la izquierda), se realiza una comprobación continua para detectar si se encuentra una pared. Si se detecta una pared durante la ejecución de una acción, se sale de esa acción con un indicador de bandera (*flag*). Esta bandera le avisa al algoritmo de Q-learning que no debe aprender nada de esa acción y que la iteración actual no debe tenerse en cuenta. Posteriormente, se repite el proceso de selección y ejecución de acciones en la siguiente iteración del bucle principal.

Esta separación cuidadosa garantiza que el comportamiento de evitar paredes se maneje de manera independiente y no interfiera con el proceso de aprendizaje del algoritmo Q-learning. Además, permite que el robot evite obstáculos de manera efectiva mientras sigue aprendiendo a navegar en su entorno.

3 Problemas y Conclusiones

Durante la realización de la práctica, se enfrentaron varios desafíos que influyeron en la implementación y los resultados obtenidos. A continuación, se presentan los principales problemas encontrados y las conclusiones derivadas de ellos:

3.1 Estrategia de Cálculo del Refuerzo

Uno de los desafíos principales fue encontrar una estrategia adecuada para el cálculo del refuerzo. Se experimentó con diferentes enfoques, incluida una estrategia más compleja que consideraba ponderaciones para las diferencias entre las lecturas de los sensores infrarrojos. Sin embargo, se determinó que la estrategia más simple, que contaba el número de sensores que detectaban el color negro, ofrecía los mejores resultados en términos de eficacia y estabilidad del comportamiento del robot.

3.2 Definición de la Duración y las Acciones

Definir la duración óptima de las acciones fue un desafío significativo así como definir correctamente estas acciones. Por ejemplo, al principio los giros se realizaban de forma estática, posteriormente se modificó para que el robot girase en movimiento. De eesta forma el comportamiento era más suave y funcionaba mejor el aprendizaje, se evitaba que el robot aprendiese a quedarse quieto oscilando de izauierda a derecha.

3.3 Naturaleza Estocástica del Algoritmo

Otro aspecto importante a considerar es la naturaleza estocástica del algoritmo de Q-learning. Aunque en la gran mayoría de los casos el comportamiento aprendido por el robot es correcto y efectivo para seguir el circuito negro, existen algunos casos, aunque en minoría, en los que el comportamiento resultante no es el más deseado. Esto se debe a la aleatoriedad inherente al proceso de aprendizaje.

3.4 Sesgo en el Aprendizaje

Se observó un sesgo en el aprendizaje del robot, ya que tendía a seguir el circuito en sentido horario. Esto se debe a la forma en que se diseñó el entorno y cómo se realizaron las acciones durante el proceso de aprendizaje. Una posible solución para evitar este sesgo sería introducir cambios de sentido forzados durante el aprendizaje para que el robot también aprenda a seguir el circuito en sentido contrario.

3.5 Atascos en Esquinas

En algunas situaciones extremas, el robot puede quedar atascado en una esquina debido a la forma en que se implementó la función `avoid_walls()`. Una posible solución para este problema sería modificar la función `avoid_walls()` para que el robot pueda liberarse de las esquinas de manera más efectiva, por ejemplo, mediante el uso de movimientos alternativos o estrategias de evasión específicas.

En conclusión, a pesar de los desafíos encontrados, la práctica permitió comprender en profundidad los conceptos de aprendizaje por refuerzo y Q-learning.