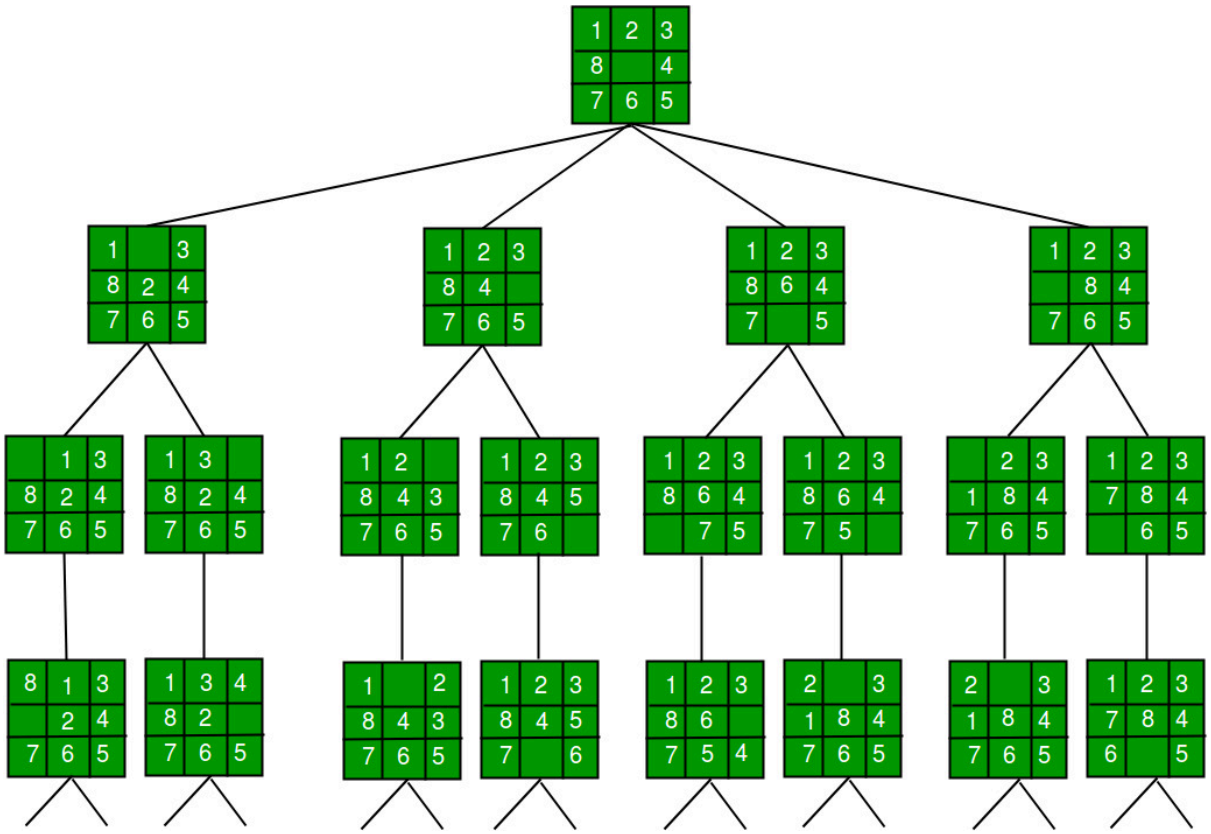
Q-LEARNING 8-PUZZLE



Adrián Yared Armas de la Nuez

Contenido

1. Objetivos de aprendizaje

- Comprender y aplicar los conceptos de aprendizaje por refuerzo y el algoritmo Q-learning.
- Implementar un agente de aprendizaje por refuerzo que resuelva el problema del 8-puzzle
- Evaluar el rendimiento del agente y ajustar los hiperparámetros para mejorar la eficacia de aprendizaje

2. Apartado

Se debe calcular la distancia a la solución desde cada estado examinando la tabla t.

2.1 Subapartado 1

Se inicializan todos los estados con distancia -1.

2.2 Código

```
def calculate_steps_to_end(transitions, goal_state):
    steps_to_end = {}
    queue = deque([(goal_state, 0)])  # (state, distance)
    visited = set()

    while queue:
        current_state, distance = queue.popleft()
        if current_state in visited:
            continue
        visited.add(current_state)
        steps_to_end[current_state] = distance - 1

        for prev_state in transitions.get(current_state, []):
            if prev_state not in visited:
                queue.append((prev_state, distance + 1))

    return steps_to_end
```

2.3 Explicación

Como podemos ver, se marca steps_to_end[current_state] = distance - 1 asigna a la clave correspondiente al estado actual (current_state) en el diccionario steps_to_end un valor que es igual a la distancia calculada desde el estado objetivo (goal_state) - 1.

2.4 Subapartado 2

Se marca el estado solución con distancia 0

2.5 Código

```
def calculate_steps_to_end(transitions, goal_state):
    steps_to_end = {}
    queue = deque([(goal_state, 0)])  # (state, distance)
    visited = set()

    while queue:
        current_state, distance = queue.popleft()
        if current_state in visited:
            continue
        visited.add(current_state)
        steps_to_end[current_state] = distance - 1

        for prev_state in transitions.get(current_state, []):
            if prev_state not in visited:
                queue.append((prev_state, distance + 1))

    return steps_to_end
```

2.6 Explicación

En queue = deque([(goal_state, 0)]) # (state, distance), se marca la distancia inicial del estado objetivo como 0. Este valor se propaga durante el proceso de cálculo de las distancias de los demás estados.

2.7 Subapartado 3

Se marcan con distancia 1 los estados contiguos a la solución

2.8 Código

```
def calculate_steps_to_end(transitions, goal_state):
    steps_to_end = {}
    queue = deque([(goal_state, 0)])  # (state, distance)
    visited = set()

    while queue:
        current_state, distance = queue.popleft()
```

```
        if current_state in visited:
            continue
        visited.add(current_state)
        steps_to_end[current_state] = distance - 1

        for prev_state in transitions.get(current_state, []):
            if prev_state not in visited:
                queue.append((prev_state, distance + 1))

    return steps_to_end
```

2.9 Explicación

Los estados contiguos al current_state se obtienen de la lista transitions.get(current_state, []), que contiene los estados conectados al estado actual. Para cada estado contiguo prev_state, se añade el valor incrementado +1, queue.append((prev_state, distance + 1)).

2.10 Subapartado 4

Se marcan con distancia k los estados (no marcados) contiguos a alguno de distancia k-1 se repite hasta que no se marque ninguno más.

2.11 Código

```
def calculate_steps_to_end(transitions, goal_state):
    steps_to_end = {}
    queue = deque([(goal_state, 0)])  # (state, distance)
    visited = set()

    while queue:
        current_state, distance = queue.popleft()
        if current_state in visited:
            continue
        visited.add(current_state)
        steps_to_end[current_state] = distance - 1

        for prev_state in transitions.get(current_state, []):
            if prev_state not in visited:
                queue.append((prev_state, distance + 1))

    return steps_to_end
```

2.12 Explicación

dentro del bucle while en la función calculate_steps_to_end. Este bucle utiliza una cola (queue) para procesar estados en orden de distancia, marcando los estados contiguos no visitados con la distancia correspondiente.
Se marcan con distancia K los estados (no marcados) contiguos a alguno de distancia K−1:

Esto se realiza iterativamente a medida que los estados se agregan a la cola con una distancia acumulada. El algoritmo BFS garantiza que todos los estados de distancia K se procesen antes que los de K+1.

Se repite hasta que no se marque ninguno más El bucle continúa mientras haya estados en la cola, lo que asegura que todos los estados alcanzables se procesen:

3. Apartado

Se usa esa información para compararla con el tamaño de path para cada estado de la tabla t.

3.1 Código

```
def compare_steps_and_paths(steps_to_end, path_lengths):
    discrepancies = []
    for state, steps in steps_to_end.items():
        path_length = path_lengths.get(state, None)
        if path_length is not None and steps != path_length:
            discrepancies.append((state, steps, path_length))
    return discrepancies
```

3.2 Explicación

Se usa la información de la distancia a la solución (steps_to_end) para compararla con el tamaño del camino (path_lengths) desde el estado inicial hasta cada estado en la tabla T. La comparación ocurre en la función compare_steps_and_paths.

Resultado si hay discrepancia:
State: , Steps to End: , Path Length:

```
State: 475639281, Steps to End: 26, Path Length: 22
State: 147938625, Steps to End: 26, Path Length: 24
State: 645981327, Steps to End: 26, Path Length: 22
State: 634581297, Steps to End: 26, Path Length: 22
State: 634951287, Steps to End: 26, Path Length: 22
State: 564931287, Steps to End: 26, Path Length: 22
State: 694231857, Steps to End: 26, Path Length: 20
State: 641237895, Steps to End: 26, Path Length: 20
State: 634915827, Steps to End: 26, Path Length: 22
```

Si no las hay:
No discrepancies found.

3. Cálculo del aprendizaje

3.1 Código

```
import numpy as np
path = []

# Mostrar los pasos para resolver el 8-puzzle desde un estado inicial
aleatorio
def resolver_8_puzzle(T, Q, steps_to_end, estado_inicial,
estado_objetivo="123456789"):
    estado_actual = estado_inicial
    pasos = 0

    while estado_actual != estado_objetivo:
        acciones = Q[estado_actual]
        # Filtrar acciones válidas
        acciones_validas = [a for a in Q[estado_actual] if
T[estado_actual][a] is not None]
        if not acciones_validas:
            break

        accion_mejor = max(acciones_validas, key=acciones.get)
        siguiente_estado = T[estado_actual][accion_mejor]

        if siguiente_estado is None:
            break

        estado_actual = siguiente_estado
        pasos += 1

        if pasos > 100:  # Limitar a 100 pasos para evitar bucles
infinitos
            break

    # Calcular porcentaje de aprendizaje
    if estado_actual == estado_objetivo:
        distancia_ideal = steps_to_end.get(estado_inicial, None)
```

```
        if distancia_ideal is not None and pasos > 0:
            return (distancia_ideal / pasos) * 100
    return 0

# Simulación de episodios
def entrenar_y_evaluar(T, Q, steps_to_end, num_episodios=1000000,
evaluacion_cada=100000):
    aprendizaje_porcentaje = []
    for episodio in range(1, num_episodios + 1):
        # Simulación de un episodio (aquí puedes entrenar `Q` según tu
lógica)

        # Evaluar cada `evaluacion_cada` episodios
        if episodio % evaluacion_cada == 0:
            print(f"Evaluando en el episodio {episodio}...")
            estados_prueba = np.random.choice(list(T.keys()), size=10,
replace=False)
            porcentajes = [
                resolver_8_puzzle(T, Q, steps_to_end,
estado_inicial=estado)
                for estado in estados_prueba
            ]
            promedio_aprendizaje = np.mean(porcentajes)
            aprendizaje_porcentaje.append((episodio,
promedio_aprendizaje))
            print(f"Porcentaje de aprendizaje promedio:
{promedio_aprendizaje:.2f}%")

    return aprendizaje_porcentaje

# Ejemplo de uso
estado_inicial = None  # Deja como None para seleccionar un estado
aleatorio
num_episodios = 1000000  # Total de episodios
evaluacion_cada = 100000  # Evaluar cada 100,000 episodios

# Asegúrate de tener `T`, `Q`, y `steps_to_end` definidos
resultados_aprendizaje = entrenar_y_evaluar(T, Q, steps_to_end,
num_episodios, evaluacion_cada)
```

3.2 Explicación

El código implementa un proceso para evaluar una solución al problema utilizando tablas de transición (T) y valores de recompensa/acción (Q). Comienza importando la librería numpy y definiendo variables iniciales como path, que probablemente se emplea para registrar los pasos del proceso. La función principal, resolver_8_puzzle, simula la resolución del rompecabezas desde un estado inicial hasta el objetivo (123456789). En cada iteración, filtra las acciones válidas basándose en las tablas Q y selecciona la mejor acción para avanzar al siguiente estado. Si no hay acciones válidas, si se alcanzan 100 pasos, o si se llega al estado objetivo, el bucle termina. Si se resuelve el puzzle, la función calcula un porcentaje de aprendizaje comparando los pasos ideales (steps_to_end) con los pasos realmente tomados.

La función entrenar_y_evaluar ejecuta simulaciones en múltiples episodios para entrenar y evaluar el modelo. Cada cierto número de episodios (evaluacion_cada), se evalúa el desempeño en estados iniciales aleatorios y se calcula el porcentaje promedio de aprendizaje, que se almacena e imprime para monitorear el progreso. Finalmente, un ejemplo de uso muestra cómo configurar parámetros como el número de episodios y realizar evaluaciones periódicas. Para ejecutarse correctamente, requiere que las tablas T, Q y steps_to_end estén definidas previamente. Este enfoque permite medir la eficiencia del aprendizaje y la solución al problema en un entorno controlado y reproducible.

3.2 Resultado

Según el resultado mostrado creo que está entrenando pero el estado inicial no se encuentra en el campo de entrenamiento, ya que tarda alrededor de 5 minutos entrenando, sin embargo muestra un 0 de aprendizaje, ya que si tardase ese tiempo pese a no tener bien ajustadas las w (los pesos), habría un aprendizaje mínimo reflejado en el resultado que no podemos ver en este caso.

```
Evaluando en el episodio 100000...
Porcentaje de aprendizaje promedio: 0.00%
Evaluando en el episodio 200000...
Porcentaje de aprendizaje promedio: 0.00%
Evaluando en el episodio 300000...
Porcentaje de aprendizaje promedio: 0.00%
Evaluando en el episodio 400000...
Porcentaje de aprendizaje promedio: 0.00%
Evaluando en el episodio 500000...
Porcentaje de aprendizaje promedio: 0.00%
Evaluando en el episodio 600000...
Porcentaje de aprendizaje promedio: 0.00%
```

3. Github y Colab