Flujo de Información en la Red Neuronal DQN para Snake

Comprender cómo fluye la información a través del agente de aprendizaje por refuerzo (específicamente un DQN, DDQN o Dueling DDQN) es fundamental. Vamos a desglosarlo basándonos en la configuración actual.

1. Configuración del Estado (La Entrada de la Red Neuronal)

El agente percibe el entorno a través de un **vector de estado**. En este caso, se han definido 14 características booleanas (verdadero/falso, que se representan como 1/0) que describen la situación actual de la serpiente:

1. danger_straight: ¿Hay un peligro inmediato (pared o cuerpo) si la serpiente sigue recto?
2. danger_left_relative: ¿Hay un peligro inmediato si la serpiente gira a su izquierda?
3. danger_right_relative: ¿Hay un peligro inmediato si la serpiente gira a su derecha?
4. trap_straight: Si la serpiente sigue recto (y es seguro hacerlo), ¿caería en una trampa inmediata (sin salidas)?
5. trap_left: Si la serpiente gira a su izquierda (y es seguro hacerlo), ¿caería en una trampa inmediata?
6. trap_right: Si la serpiente gira a su derecha (y es seguro hacerlo), ¿caería en una trampa inmediata?
7. moving_up: ¿La serpiente se está moviendo actualmente hacia arriba?
8. moving_down: ¿La serpiente se está moviendo actualmente hacia abajo?
9. moving_left: ¿La serpiente se está moviendo actualmente hacia la izquierda?
10. moving_right: ¿La serpiente se está moviendo actualmente hacia la derecha?
11. food_left_of_snake: ¿La comida está a la izquierda de la cabeza de la serpiente?
12. food_right_of_snake: ¿La comida está a la derecha de la cabeza de la serpiente?
13. food_above_snake: ¿La comida está arriba de la cabeza de la serpiente?
14. food_below_snake: ¿La comida está abajo de la cabeza de la serpiente?

Estos 14 "pulsos lógicos" (1s o 0s) forman un vector que es la única entrada que la red neuronal del agente recibe en cada instante para tomar una decisión.
Además de esta información de estado, el agente aprende a través de las recompensas (positivas o negativas) que recibe después de cada acción, las cuales le indican si su acción fue beneficiosa o perjudicial en el contexto de ese estado.

2. Acciones Posibles (La Salida de la Red Neuronal)

La serpiente tiene un conjunto limitado de movimientos que puede realizar en relación con su dirección actual. En tu implementación, estas son **3 acciones relativas posibles**:

- **Acción 0:** Seguir recto.

- **Acción 1:** Girar a la izquierda (relativo a su dirección de movimiento actual).
- **Acción 2:** Girar a la derecha (relativo a su dirección de movimiento actual).

La red neuronal del agente está diseñada para, dado un estado de entrada (los 14 booleanos), producir una **estimación de la "calidad" o "valor Q" para cada una de estas 3 acciones**. Por lo tanto, la capa de salida de tu red neuronal tendrá 3 neuronas, cada una correspondiendo a una de estas acciones:

- Salida 1: Q(estado, recto)
- Salida 2: Q(estado, izquierda)
- Salida 3: Q(estado, derecha)

El valor Q(s,a) representa la recompensa total acumulada esperada si se toma la acción a en el estado s y se sigue la política óptima a partir de entonces.

3. Creación del Agente

Cuando inicializas el agente (por ejemplo, DQNAgent), le pasas el tamaño del espacio de estados y el tamaño del espacio de acciones:

agent = DQNAgent(state_size=14, action_size=3, ...)

Esto configura la arquitectura interna de la red neuronal del agente para que tenga 14 unidades de entrada y 3 unidades de salida (para los Q-values). Los demás hiperparámetros (tasa de aprendizaje, gamma, épsilon, etc.) definen cómo aprenderá el agente.

4. El Ciclo de Decisión y Aprendizaje en Cada Paso del Juego (step)

Este es el núcleo del proceso:

1. **Observar el Estado Actual (**St**):**
    - Al comienzo de cada paso en un episodio, el entorno del juego (tu clase SnakeLogic) calcula el vector de estado actual de 14 características utilizando su método get_state().
2. **Seleccionar una Acción (**At**):**
    - Este vector de estado St se pasa al método agent.get_action(S_t).
    - Dentro de get_action, ocurren dos cosas principales:
        - **Exploración vs. Explotación:** Se decide si explorar (tomar una acción aleatoria para descubrir nuevas posibilidades) o explotar (usar el conocimiento actual para tomar la mejor acción). Esto se controla mediante el parámetro épsilon (ϵ).
            - Con probabilidad ϵ, se elige una acción aleatoria (a menudo de un

conjunto de acciones "seguras" para evitar colisiones tontas al principio).
        - Con probabilidad $1-\epsilon$, se explota: la red neuronal principal (self.model) toma St como entrada y predice los 3 Q-values (Q(St,recto), Q(St,izquierda), Q(St,derecha)).
    - El agente selecciona la acción At que tiene el Q-value estimado más alto (esta es la acción "greedy" o codiciosa).
3. **Ejecutar la Acción y Observar el Resultado:**
    - La acción seleccionada At (un índice 0, 1 o 2) se pasa al entorno del juego mediante env.step(A_t).
    - SnakeLogic actualiza la posición de la serpiente según la acción, comprueba colisiones, si se comió la comida, etc.
    - El entorno devuelve:
        - La recompensa inmediata Rt+1 obtenida por realizar At en St.
        - El nuevo estado St+1 (el vector de 14 características después de que la acción se haya completado).
        - Un booleano done que indica si el episodio ha terminado (por ejemplo, por choque o estancamiento).
        - info adicional (como el tipo de colisión).
4. **Almacenar la Experiencia (Transición):**
    - La transición completa, que es la experiencia (St,At,Rt+1,St+1,done), se guarda en la memoria de repetición del agente (agent.remember(...)). Esta memoria almacena muchas de estas transiciones pasadas.
5. **Aprender de la Experiencia (Replay):**
    - Periódicamente (por ejemplo, después de cada paso, si la memoria tiene suficientes experiencias), se llama al método agent.replay().
    - **Muestreo:** Se toma un lote (minibatch) de transiciones (Sj,Aj,Rj+1,Sj+1,donej) aleatoriamente de la memoria de repetición.
    - **Cálculo del Target Q-value:** Para cada transición en el minibatch, el agente calcula un "valor Q objetivo". Este es el valor que *idealmente* debería tener Q(Sj,Aj).
        - Si donej es verdadero (el episodio terminó), el target es simplemente Rj+1.
        - Si no, el target se calcula usando la ecuación de Bellman (con las modificaciones de DDQN o Dueling DDQN si se usan):
            - **DQN Clásico:** $R_{j+1}+\gamma \max_{a'} Q_{target}(S_{j+1},a')$
            - DDQN: $R_{j+1}+\gamma Q_{target}(S_{j+1},\text{argmax}_{a'} Q_{online}(S_{j+1},a'))$
              (Donde Qtarget es la red objetivo y Qonline es la red principal).
    - **Entrenamiento de la Red:** La red neuronal principal (self.model) se entrena. Se le dan los Sj del minibatch como entrada y se ajustan sus pesos (mediante

descenso de gradiente) para que los Q-values que predice para las Aj se acerquen más a los "valores Q objetivo" calculados. El objetivo es minimizar una función de pérdida (generalmente el error cuadrático medio, mse) entre los Q-values predichos y los Q-values objetivo.

Conclusión del Flujo

Es importante entender que la red neuronal **no se reconfigura en cada paso**. Su arquitectura (14 entradas, capas ocultas, 3 salidas) se define una vez. Lo que cambia continuamente es el **valor de sus pesos internos** a medida que aprende de las experiencias a través del proceso de replay.

Este ciclo de **observar estado → seleccionar acción → ejecutar acción → observar resultado → almacenar experiencia → aprender del replay** se repite durante muchos episodios, permitiendo que la red neuronal mejore gradualmente su capacidad para estimar los Q-values y, por lo tanto, para tomar decisiones que maximicen la recompensa acumulada a largo plazo.