INTELIGENCIA ARTIFICIAL PARA JUEGOS

SESIÓN 8 (DEEP REINFORCEMENT LEARNING)

Dr. Edwin Villanueva Talavera

Contenido

- Generalización en Aprendizaje por Refuerzo
- Deep Q-Networks
- Doble Deep Q-netwotks

Generalización en Aprendizaje por Refuerzo

- Los agentes vistos hasta ahora guardan las utilidades y Q valores en tablas (un valor por estado o estado-acción)
- Funcionan bien para espacios de estados pequeños. Para espacios grandes no son adecuados (ej. Backgammon, Ajedrez)
- Una forma de escalar a problemas grandes es estimar las utilidades o Q-valores con funciones aproximadas. Por ejemplo, podríamos usar una aproximación lineal de la utilidad:

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \cdots + \theta_n f_n(s)$$

donde: $f_1, \ldots, f_n$ son atributos que describen el estado, y

$\theta_1, \ldots, \theta_n$ son parámetros ajustables para aproximar la utilidad

- Al usar una función aproximada se comprime la definición de utilidades (o q-valores) del número de estados del problema a n (# de parámetros) (ej. en ajedrez de 10^{40} a 20)

Generalización en Aprendizaje por Refuerzo

- Otra ventaja de funciones aproximadas es que ellas habilitan al agente a generalizar de estados visitados a estados no visitados
- La clave está en escoger la forma funcional adecuada (hipótesis). Entre más parámetros tenga más tiempo y datos serán necesarios
- Para el caso del laberinto 3x4 y una aproximación lineal podemos usar las coordenadas *x*, *y* como atributos:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

- Regresion lineal puede estimar los parámetros si hay datos.
- Es preferible usar aprendizaje online al fin de cada trial. Si $u_j(s)$ es la utilidad observada en trial j, el error de la función aproximada será:

$$E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$$

Generalización en Aprendizaje por Refuerzo

- La razón de cambio del error en relación a cada param $\partial E_j / \partial \theta_i$ sirve para actualizar los parámetros (regla Delta):

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

- Para el caso del laberinto 3x4 se tiene:

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) , \\
\theta_1 &\leftarrow \theta_1 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) x , \\
\theta_2 &\leftarrow \theta_2 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) y .
\end{aligned}$$

- Al actualizar los parámetros la función se actualiza para todas las futuras predicciones en todos los estados. Esto es, el agente generaliza de su experiencia

Generalización en Aprendizaje por Refuerzo

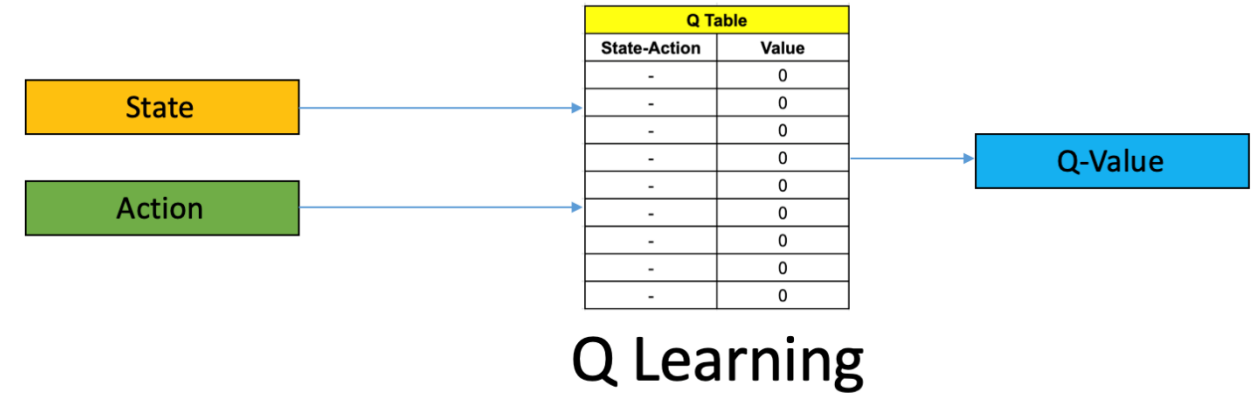
- Para el caso de Q-learning la regla de actualización online seria:

$$\theta_i \leftarrow \theta_i + \alpha\left[R(s) + \gamma \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)\right] \frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i}$$

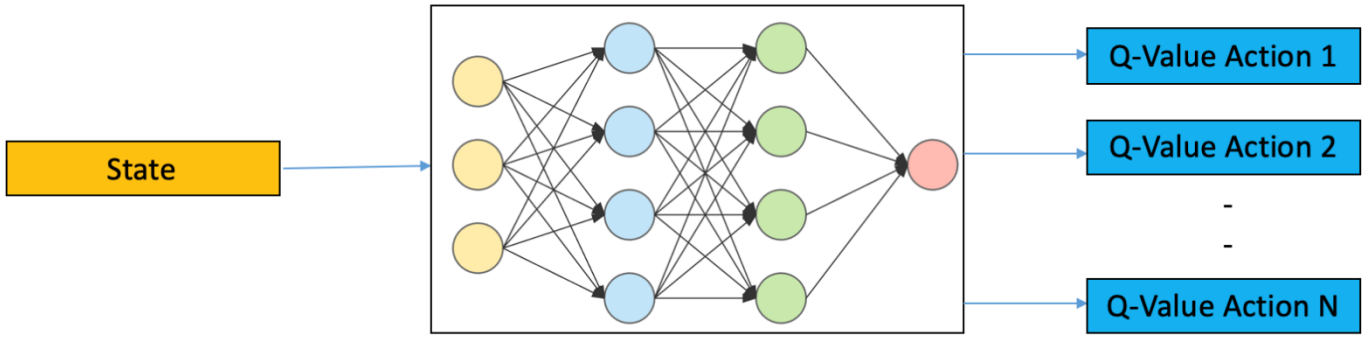
- Una buena alternativa como aproximador funcional es usar redes neuronales. Estos son modelos bastante flexibles. El cálculo de gradientes es simple si se usa funciones de activación diferenciables.
- De hecho, se ha usado redes neuronales profundas para ello, dando lugar al famoso Deep Q-learning (Mnih *et al.*, Nature 02/2015)
  - Testado en 49 juegos del Atari 2600. Entradas de 84x84 pixeles y score del juego. Nivel comparable al de un testador profesional de juegos (mismo algoritmo, arquitectura y híper-parámetros)

Deep Q-networks

- En Deep Q-network se usa una Red neuronal para aproximar los valores Q



Q Learning



Deep Q Learning

Deep Q-networks

- Cada experiencia simple es almacenada en memoria. Normalmente una experiencia simple es un cuarteto:

(state, action, next_state, reward)

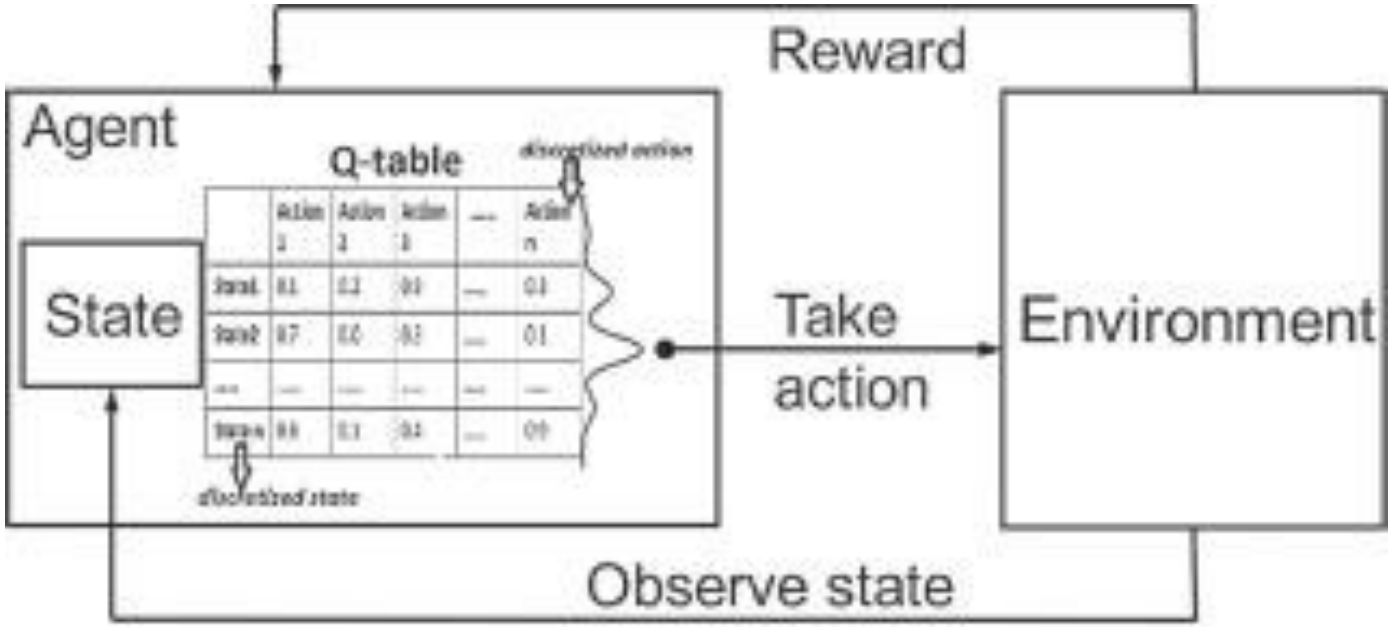
S_t A_t S_{t+1} R_{t+1}

- La acción tomada por el agente es la que tiene el máximo valor Q predicho por el modelo
- La función de perdida es el Error Cuadrático Medio (MSE) del valor Q predicho y el valor target Q* . Como no se conoce el target entonces se usa como target

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

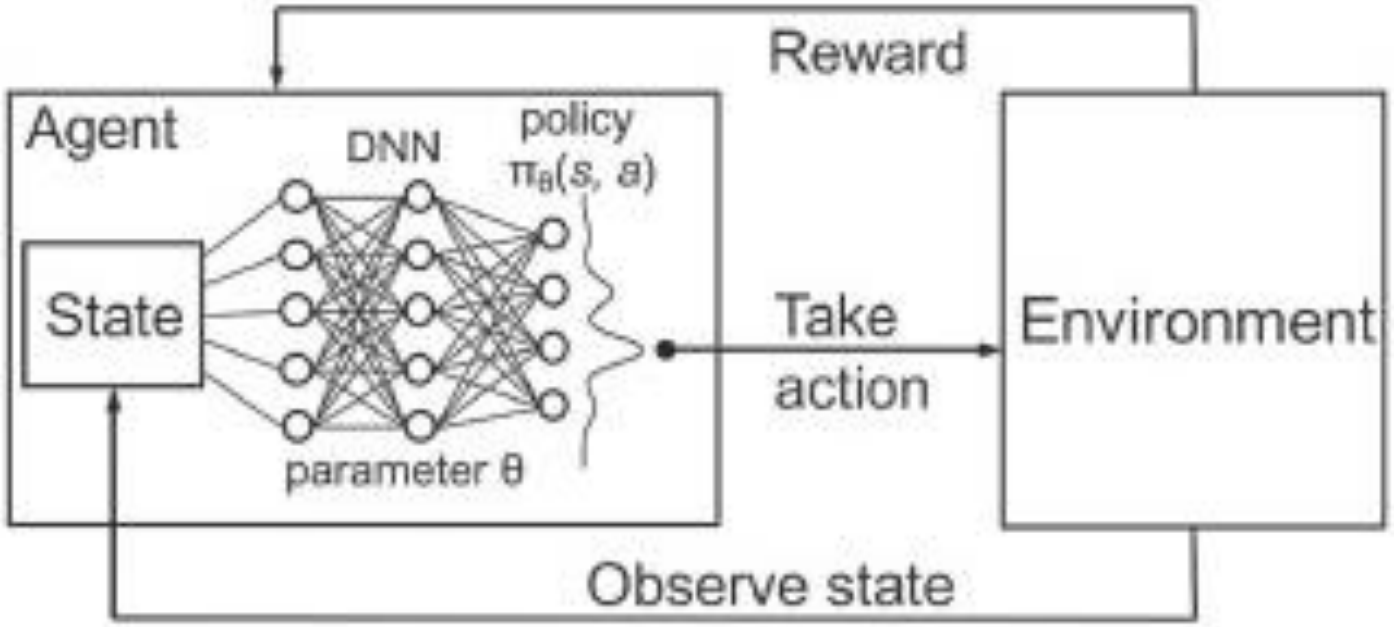
Deep Q-networks

- Operación del Agente Q-learning



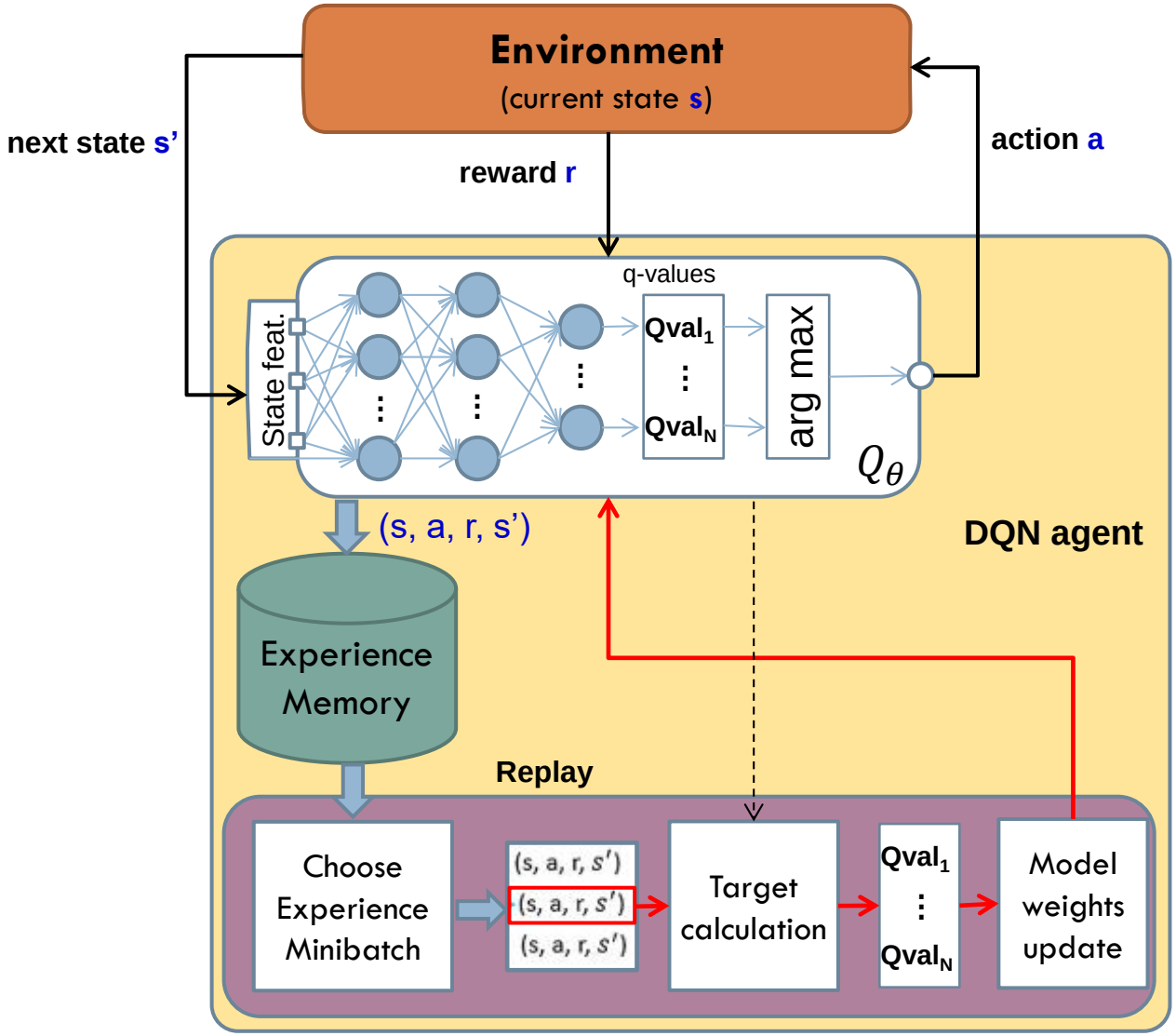
Deep Q-networks

- Operación del Agente Deep Q-network



Deep Q-networks

- Flujo de Entrenamiento de un DQN



Deep Q-networks

Initialize experience memory D
Initialize model Q_θ with random weights θ
for episode 1:n **do**
    # Make an episode experience with model Q_θ
    s = reset_environment()
    while $s != terminal$
        select an action a
            with probability ε: $a \leftarrow random(Actions(s))$
            otherwise: $a \leftarrow argmax_{a'} Q_\theta(s, a')$
        execute action a in environment and observe reward r and new state s'
        store transition $[s, a, r, s']$ in experience memory D
        $s \leftarrow s'$

    # Update the model Q_θ (replay)
    get a random sample of experiences: $Minibatch \leftarrow Sample(D, batchsize)$
    for each transition $[s, a, r, s'] \in Minibatch$
        $\boldsymbol{t} \leftarrow Q_\theta(s)$ *# vector of q-values predicted by the current model*
        if $s' = terminal$:
            $\boldsymbol{t}_a \leftarrow r$
        else:
            $\boldsymbol{t}_a \leftarrow r + \gamma\, max_{a'} Q_\theta(s', a')$ *# future discounted reward obtained with the model*
        update weights θ of model Q_θ with example $\langle s, \boldsymbol{t} \rangle$
    $\varepsilon \leftarrow \varepsilon * decay$ *# decay the probability of random actions (exploration)*

Double Deep Q-networks

- DQN ajusta el modelo Q_θ con cada experiencia (transición) revisada en el minibatch de replay. El problema es que la construcción del target de entrenamiento se realiza con el mismo modelo que esta siendo ajustado Q_θ : $\boldsymbol{t}_a \leftarrow r + \gamma \, max_{a'} \, Q_\theta(s', a')$

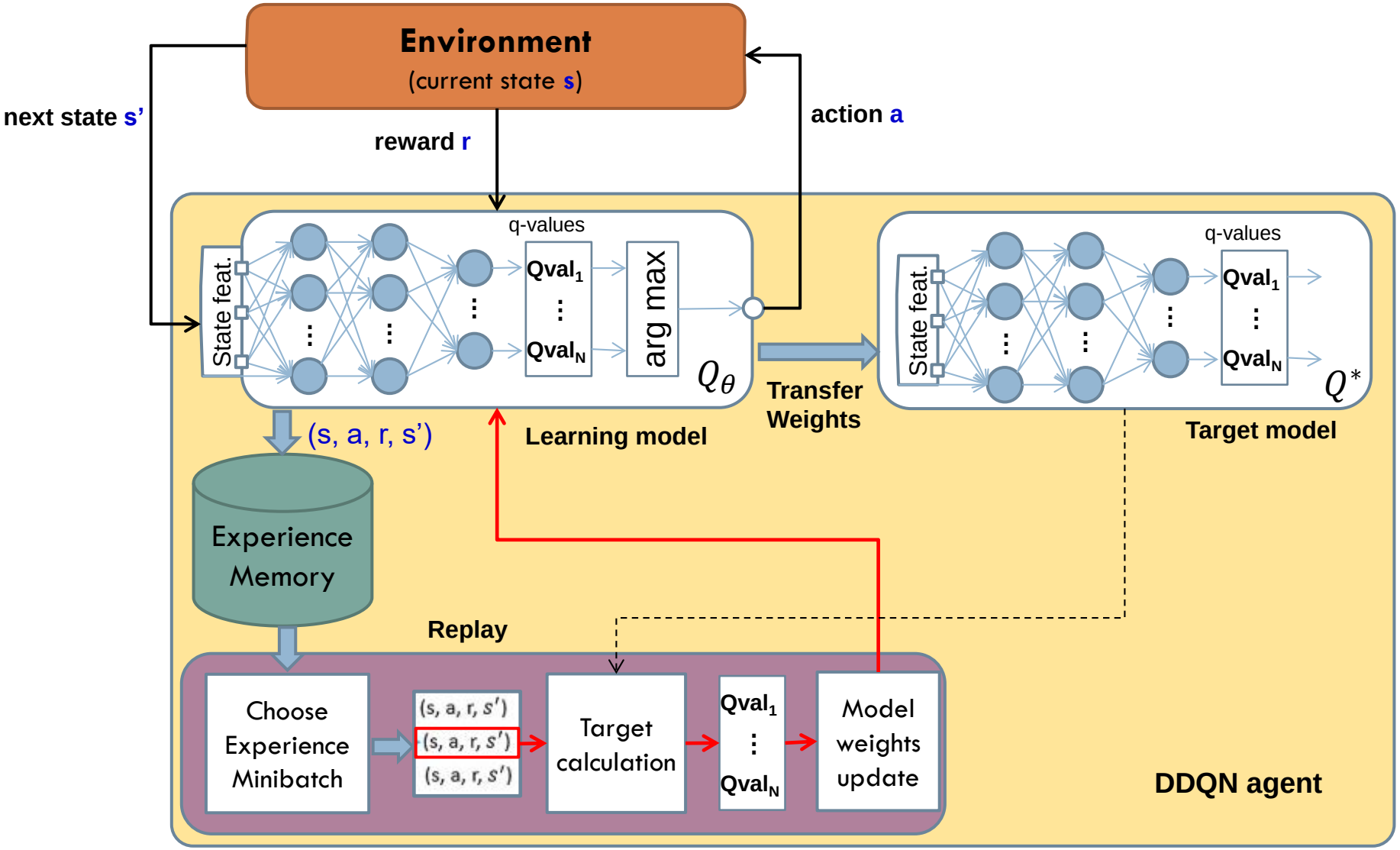
- Ello trae como consecuencia una inestabilidad de aprendizaje en DQN

- Una forma de aliviar ese problema es Double Deep Q-networks, el cual usa un modelo adicional Q^* (target model) para estimar el target durante la etapa de replay: $\boldsymbol{t}_a \leftarrow r + \gamma \, max_{a'} \, Q^*(s', a')$

- Después de cada etapa de replay se transfieren los pesos de Q_θ a Q^* , y este ultimo se usa como estimador de target en el siguiente replay

Double Deep Q-networks

- Flujo de Entrenamiento de un Double DQN (DDQN)



Double Deep Q-networks

Initialize experience memory D
Initialize model Q_θ with random weights θ
Initialize model Q^* with random weights θ^*
for episode 1:n **do**
    # Make a episode experience with model Q_θ
    s = reset_environment()
    while $s != terminal$
        select an action a
            with probability ε: $a \leftarrow random(Actions(s))$
            otherwise: $a \leftarrow argmax_{a'} Q_\theta(s, a')$
        execute action a in environment and observe reward r and new state s'
        store transition $[s, a, r, s']$ in experience memory D
        $s \leftarrow s'$

    # Update the model Q_θ (replay)
    get a random sample of experiences: $Minibatch \leftarrow Sample(D, batchsize)$
    for each transition $[s, a, r, s'] \in Minibatch$
        $\boldsymbol{t} \leftarrow Q_\theta(s)$ *# vector of q-values predicted by the model being learned*
        if $s' = terminal$:
            $\boldsymbol{t}_a \leftarrow r$
        else:
            $\boldsymbol{t}_a \leftarrow r + \gamma\, max_{a'} Q^*(s', a')$ *# future discounted reward obtained with the target model*
            update weights θ of model Q_θ with example $\langle s, \boldsymbol{t} \rangle$
    transfer weights to the target model : $\theta^* \leftarrow \theta$
    $\varepsilon \leftarrow \varepsilon * decay$ *# decay the probability of random actions (exploration)*

Referencias y Material complementar

- DQN: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf
- DDQN: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12389/11847
- DEMYSTIFYING DEEP REINFORCEMENT LEARNING: https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/
- Deep Reinforcement Learning: Pong from Pixels: https://karpathy.github.io/2016/05/31/rl/
- A Beginner's Guide to Deep Reinforcement Learning: https://skymind.ai/wiki/deep-reinforcement-learning

Preguntas?