

Aprendizaje por refuerzo profundo

Fernando Lozano

Universidad de los Andes

2 de mayo de 2023



Deep Reinforcement Learning

Deep Reinforcement Learning

- RL con aproximación de funciones, donde $q_* \approx \hat{q}(s, a, \mathbf{w})$ es una red profunda.

Deep Reinforcement Learning

- RL con aproximación de funciones, donde $q_* \approx \hat{q}(s, a, \mathbf{w})$ es una red profunda.
- Red profunda:

Deep Reinforcement Learning

- RL con aproximación de funciones, donde $q_* \approx \hat{q}(s, a, \mathbf{w})$ es una red profunda.
- Red profunda:
 - ▶ Aprende representación de la entrada.

Deep Reinforcement Learning

- RL con aproximación de funciones, donde $q_* \approx \hat{q}(s, a, \mathbf{w})$ es una red profunda.
- Red profunda:
 - ▶ Aprende representación de la entrada.
 - ▶ No requiere diseño de descriptores **a mano**.

Deep Reinforcement Learning

- RL con aproximación de funciones, donde $q_* \approx \hat{q}(s, a, \mathbf{w})$ es una red profunda.
- Red profunda:
 - ▶ Aprende representación de la entrada.
 - ▶ No requiere diseño de descriptores **a mano**.
 - ▶ Requiere \gg datos.

Deep Reinforcement Learning

- RL con aproximación de funciones, donde $q_* \approx \hat{q}(s, a, \mathbf{w})$ es una red profunda.
- Red profunda:
 - ▶ Aprende representación de la entrada.
 - ▶ No requiere diseño de descriptores **a mano**.
 - ▶ Requiere \gg datos.
 - ▶ Transfer learning.

Deep Reinforcement Learning

- RL con aproximación de funciones, donde $q_* \approx \hat{q}(s, a, \mathbf{w})$ es una red profunda.
- Red profunda:
 - ▶ Aprende representación de la entrada.
 - ▶ No requiere diseño de descriptores **a mano**.
 - ▶ Requiere \gg datos.
 - ▶ Transfer learning.
- Redes convolucionales → imágenes.

Deep Reinforcement Learning

- RL con aproximación de funciones, donde $q_* \approx \hat{q}(s, a, \mathbf{w})$ es una red profunda.
- Red profunda:
 - ▶ Aprende representación de la entrada.
 - ▶ No requiere diseño de descriptores **a mano**.
 - ▶ Requiere \gg datos.
 - ▶ Transfer learning.
- Redes convolucionales → imágenes.
- Misma arquitectura aplicable a diferentes problemas.

Deep Q-Networks (DQN) (Mnih et.al, 2013)

Deep Q-Networks (DQN) (Mnih et.al, 2013)

- Aprendizaje de 49 juegos de Atari 2600.

Deep Q-Networks (DQN) (Mnih et.al, 2013)

- Aprendizaje de 49 juegos de Atari 2600.



Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Deep Q-Networks (DQN) (Mnih et.al, 2013)

- Aprendizaje de 49 juegos de Atari 2600.



Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

- ▶ Misma entrada: 3 cuadros 84×84 pixeles, blanco y negro

Deep Q-Networks (DQN) (Mnih et.al, 2013)

- Aprendizaje de 49 juegos de Atari 2600.



Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

- ▶ Misma entrada: 3 cuadros 84×84 pixeles, blanco y negro
- ▶ Misma arquitectura de la red: 2 capas convolucionales, 1 capa totalmente conectada, 1 salida por acción.

Deep Q-Networks (DQN) (Mnih et.al, 2013)

- Aprendizaje de 49 juegos de Atari 2600.



Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

- ▶ Misma entrada: 3 cuadros 84×84 pixeles, blanco y negro
- ▶ Misma arquitectura de la red: 2 capas convolucionales, 1 capa totalmente conectada, 1 salida por acción.
- ▶ Igual valor de hiperparámetros.

- Q -Learning Tabular:

$$Q(S_t, A_t) \leftarrow Q(\textcolor{red}{S}_t, A_t) + \alpha \left[\textcolor{red}{R}_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

- Q -Learning Tabular:

$$Q(S_t, A_t) \leftarrow Q(\textcolor{red}{S}_t, \textcolor{red}{A}_t) + \alpha \left[\textcolor{red}{R}_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

- Con aproximación de funciones:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[\textcolor{red}{R}_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Modificaciones

Modificaciones

- Experience Replay

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .
 - ▶ Selecciona minibatch aleatoriamente de \mathcal{D}

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .
 - ▶ Selecciona minibatch aleatoriamente de \mathcal{D}
 - ★ Una experiencia se puede usar en diferentes actualizaciones.

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .
 - ▶ Selecciona minibatch aleatoriamente de \mathcal{D}
 - ★ Una experiencia se puede usar en diferentes actualizaciones.
 - ★ Rompe correlaciones entre experiencias sucesivas, reduce varianza.

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .
 - ▶ Selecciona minibatch aleatoriamente de \mathcal{D}
 - ★ Una experiencia se puede usar en diferentes actualizaciones.
 - ★ Rompe correlaciones entre experiencias sucesivas, reduce varianza.
 - ★ Evita oscilaciones o divergencia en los parámetros.

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .
 - ▶ Selecciona minibatch aleatoriamente de \mathcal{D}
 - ★ Una experiencia se puede usar en diferentes actualizaciones.
 - ★ Rompe correlaciones entre experiencias sucesivas, reduce varianza.
 - ★ Evita oscilaciones o divergencia en los parámetros.
 - ▶ Experiencia es **off-policy**

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .
 - ▶ Selecciona minibatch aleatoriamente de \mathcal{D}
 - ★ Una experiencia se puede usar en diferentes actualizaciones.
 - ★ Rompe correlaciones entre experiencias sucesivas, reduce varianza.
 - ★ Evita oscilaciones o divergencia en los parámetros.
 - ▶ Experiencia es **off-policy** → Q-Learning.

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .
 - ▶ Selecciona minibatch aleatoriamente de \mathcal{D}
 - ★ Una experiencia se puede usar en diferentes actualizaciones.
 - ★ Rompe correlaciones entre experiencias sucesivas, reduce varianza.
 - ★ Evita oscilaciones o divergencia en los parámetros.
 - ▶ Experiencia es **off-policy** → Q-Learning.
- Target Network se actualiza más lentamente:

Modificaciones

- Experience Replay
 - ▶ Almacena **experiencias** $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ en memoria de replay \mathcal{D} .
 - ▶ Selecciona minibatch aleatoriamente de \mathcal{D}
 - ★ Una experiencia se puede usar en diferentes actualizaciones.
 - ★ Rompe correlaciones entre experiencias sucesivas, reduce varianza.
 - ★ Evita oscilaciones o divergencia en los parámetros.
 - ▶ Experiencia es **off-policy** → Q-Learning.
- Target Network se actualiza más lentamente:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[R_t + \gamma \max_a \hat{q}(S_{t+1}, a, \bar{\mathbf{w}}) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)$$

- ▶ $\bar{\mathbf{w}}$ se actualiza cada C iteraciones.

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, $\mathcal{D}, k = 0$

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, $\mathcal{D}, k = 0$

repeat

▷ para cada episodio

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

$k \leftarrow k + 1$

▷ para cada paso del episodio

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, $\mathcal{D}, k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \max_a \hat{q}(s_{i+1}, a_i, \bar{\mathbf{w}}) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \max_a \hat{q}(s_{i+1}, a_i, \bar{\mathbf{w}}) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^B [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \max_a \hat{q}(s_{i+1}, a_i, \bar{\mathbf{w}}) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^B [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

if $(k \bmod C = 0)$ **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \max_a \hat{q}(s_{i+1}, a_i, \bar{\mathbf{w}}) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^B [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

if $(k \bmod C = 0)$ **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

until S' es terminal

DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \max_a \hat{q}(s_{i+1}, a_i, \bar{\mathbf{w}}) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^B [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

if $(k \bmod C = 0)$ **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

until S' es terminal

until ∞

ALE: Atari 2600

ALE: Atari 2600

- Red convolucional:

ALE: Atari 2600

- Red convolucional:
 - ▶ 3 capas convolucionales.

ALE: Atari 2600

- Red convolucional:
 - ▶ 3 capas convolucionales.
 - ▶ 1 capa totalmente conectada.

ALE: Atari 2600

- Red convolucional:
 - ▶ 3 capas convolucionales.
 - ▶ 1 capa totalmente conectada.
 - ▶ Entrada es estado (3 frames), 1 salida para cada acción (18).

ALE: Atari 2600

- Red convolucional:
 - ▶ 3 capas convolucionales.
 - ▶ 1 capa totalmente conectada.
 - ▶ Entrada es estado (3 frames), 1 salida para cada acción (18).
- Recompensa: $\pm 1, 0$: score aumenta, disminuye o no cambia.

ALE: Atari 2600

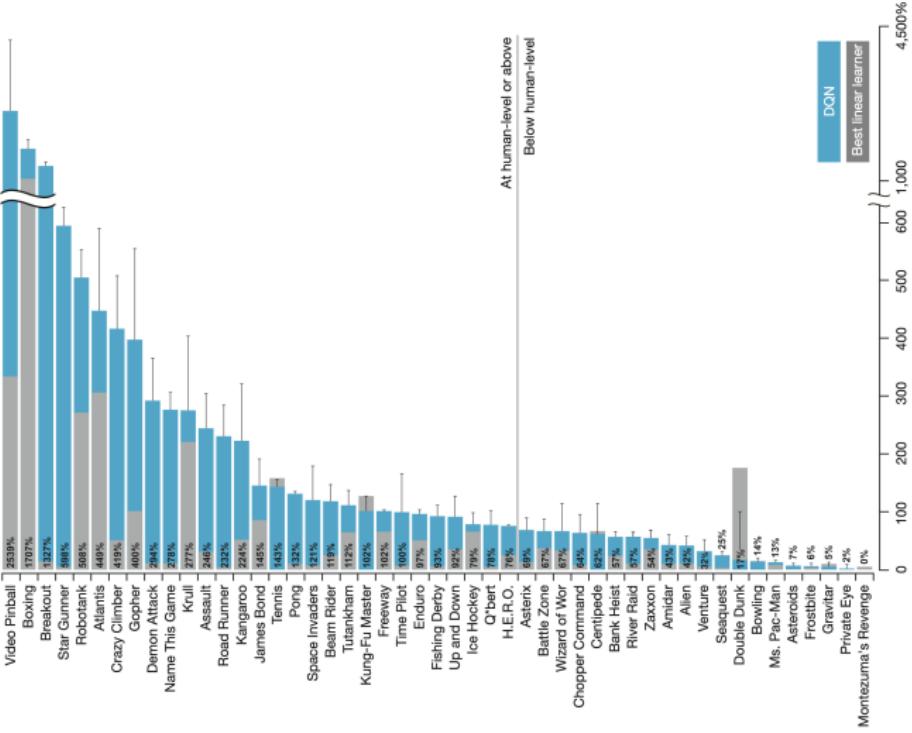
- Red convolucional:
 - ▶ 3 capas convolucionales.
 - ▶ 1 capa totalmente conectada.
 - ▶ Entrada es estado (3 frames), 1 salida para cada acción (18).
- Recompensa: $\pm 1, 0$: score aumenta, disminuye o no cambia.
- Entrenamiento con RMSProp con $B = 32$.

ALE: Atari 2600

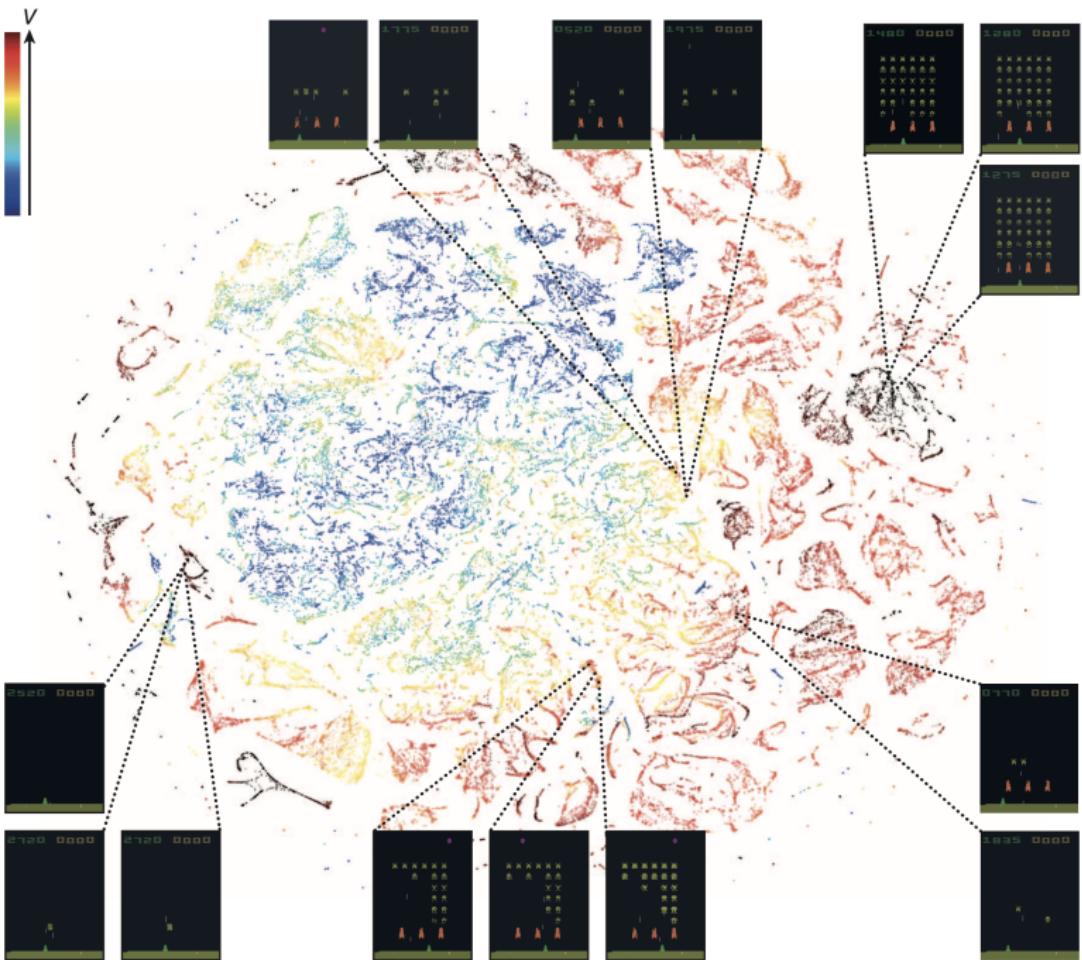
- Red convolucional:
 - ▶ 3 capas convolucionales.
 - ▶ 1 capa totalmente conectada.
 - ▶ Entrada es estado (3 frames), 1 salida para cada acción (18).
- Recompensa: $\pm 1, 0$: score aumenta, disminuye o no cambia.
- Entrenamiento con RMSProp con $B = 32$.
- 5×10^7 iteraciones, \mathcal{D} con 10^6 experiencias más recientes.

ALE: Atari 2600

- Red convolucional:
 - ▶ 3 capas convolucionales.
 - ▶ 1 capa totalmente conectada.
 - ▶ Entrada es estado (3 frames), 1 salida para cada acción (18).
- Recompensa: $\pm 1, 0$: score aumenta, disminuye o no cambia.
- Entrenamiento con RMSProp con $B = 32$.
- 5×10^7 iteraciones, \mathcal{D} con 10^6 experiencias más recientes.
- Agente supera jugador humano en 22 de los juegos.



$$100 \times \frac{\text{DQN} - \text{random}}{\text{humano} - \text{random}}$$



Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat

▷ para cada episodio

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat

▷ para cada episodio

Incialice S

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Inicialice S

repeat ▷ para cada paso del episodio

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

 Tome acción A , observe R, S' .

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

 Tome acción A , observe R, S' .

 Lanzar moneda

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

 Tome acción A , observe R, S' .

 Lanzar moneda

if Cara **then**

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

 Tome acción A , observe R, S' .

 Lanzar moneda

if Cara **then**

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha [R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A)]$$

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

 Tome acción A , observe R, S' .

 Lanzar moneda

if Cara **then**

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha [R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A)]$$

else

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha [R + \gamma Q_1(S', \arg \max_a Q_2(S', a)) - Q_2(S, A)]$$

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

 Tome acción A , observe R, S' .

 Lanzar moneda

if Cara **then**

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha [R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A)]$$

else

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha [R + \gamma Q_1(S', \arg \max_a Q_2(S', a)) - Q_2(S, A)]$$

end if

$S \leftarrow S'$

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

 Tome acción A , observe R, S' .

 Lanzar moneda

if Cara **then**

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha [R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A)]$$

else

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha [R + \gamma Q_1(S', \arg \max_a Q_2(S', a)) - Q_2(S, A)]$$

end if

$S \leftarrow S'$

until S es terminal

Double Q-Learning (Tabular)

Require: Tamaño de paso $\alpha \in (0, 1]$, $\epsilon > 0$

Incialice $Q_1(s, a), Q_2(s, a) \forall s \in \mathcal{S}^+$, con $Q_{1,2}(s_{\text{terminal}}, \cdot) = 0$

repeat ▷ para cada episodio

 Incialice S

repeat ▷ para cada paso del episodio

 Escoja A de $\mathcal{A}(S)$, de acuerdo a $Q_1 + Q_2$ (ϵ – greedy)

 Tome acción A , observe R, S' .

 Lanzar moneda

if Cara **then**

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha [R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A)]$$

else

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha [R + \gamma Q_1(S', \arg \max_a Q_2(S', a)) - Q_2(S, A)]$$

end if

$S \leftarrow S'$

until S es terminal

until ∞

Errores de estimación \Rightarrow sobreoptimismo

Errores de estimación \Rightarrow sobreoptimismo

$$q_*(s, a) = v_*(s)$$

Errores de estimación \Rightarrow sobreoptimismo

$$\begin{aligned} q_*(s, a) &= v_*(s) \\ \sum_a (Q_t(s, a) - v_*(s)) &= 0 \end{aligned}$$

Errores de estimación \Rightarrow sobreoptimismo

$$\begin{aligned} q_*(s, a) &= v_*(s) \\ \sum_a (Q_t(s, a) - v_*(s)) &= 0 \\ \frac{1}{m} \sum_a (Q_t(s, a) - v_*(s))^2 &= C \end{aligned}$$

Errores de estimación \Rightarrow sobreoptimismo

$$\left. \begin{array}{l} q_*(s, a) = v_*(s) \\ \sum_a (Q_t(s, a) - v_*(s)) = 0 \\ \frac{1}{m} \sum_a (Q_t(s, a) - v_*(s))^2 = C \end{array} \right\} \Rightarrow \max_a Q_t(s, a) \geq v_*(s) + \sqrt{\frac{C}{m-1}}$$

Errores de estimación \Rightarrow sobreoptimismo

$$\left. \begin{array}{l} q_*(s, a) = v_*(s) \\ \sum_a (Q_t(s, a) - v_*(s)) = 0 \\ \frac{1}{m} \sum_a (Q_t(s, a) - v_*(s))^2 = C \end{array} \right\} \Rightarrow \max_a Q_t(s, a) \geq v_*(s) + \sqrt{\frac{C}{m-1}}$$

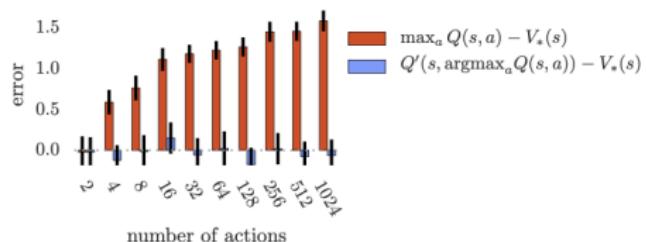


Figure 1: The orange bars show the bias in a single Q-learning update when the action values are $Q(s, a) = V_*(s) + \epsilon_a$ and the errors $\{\epsilon_a\}_{a=1}^m$ are independent standard normal random variables. The second set of action values Q' , used for the blue bars, was generated identically and independently. All bars are the average of 100 repetitions.

Sesgo de Maximización con aproximación de funciones

Sesgo de Maximización con aproximación de funciones

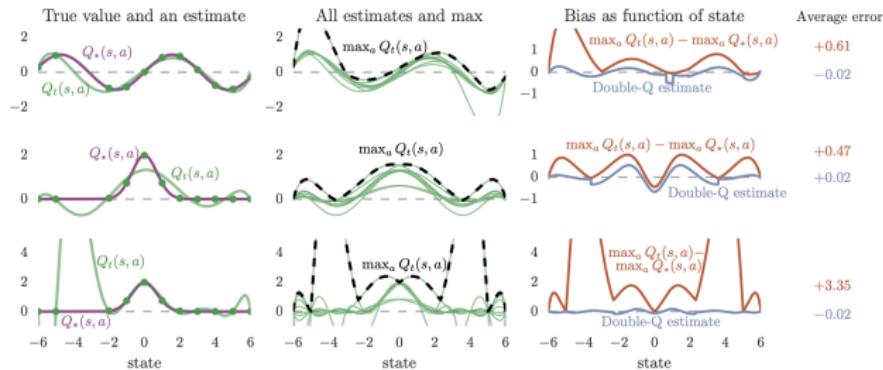


Figure 2: Illustration of overestimations during learning. In each state (x -axis), there are 10 actions. The **left column** shows the true values $V_*(s)$ (purple line). All true action values are defined by $Q_*(s, a) = V_*(s)$. The green line shows estimated values $Q(s, a)$ for one action as a function of state, fitted to the true value at several sampled states (green dots). The **middle column** plots show all the estimated values (green), and the maximum of these values (dashed black). The maximum is higher than the true value (purple, left plot) almost everywhere. The **right column** plots show the difference in orange. The blue line in the right plots is the estimate used by Double Q-learning with a second set of samples for each state. The blue line is much closer to zero, indicating less bias. The three **rows** correspond to different true functions (left, purple) or capacities of the fitted function (left, green). (Details in the text)

Doble DQN(Hasselt et.al. 2016)

Doble DQN(Hasselt et.al. 2016)

- Q Learning con aproximación de funciones:

Doble DQN(Hasselt et.al. 2016)

- Q Learning con aproximación de funciones:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Doble DQN(Hasselt et.al. 2016)

- Q Learning con aproximación de funciones:

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t) \\ &= \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)\end{aligned}$$

Doble DQN(Hasselt et.al. 2016)

- Q Learning con aproximación de funciones:

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t) \\ &= \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)\end{aligned}$$

- Actualización separando evaluación y estimación:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \hat{\mathbf{w}}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Doble DQN(Hasselt et.al. 2016)

- Q Learning con aproximación de funciones:

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t) \\ &= \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)\end{aligned}$$

- Actualización separando evaluación y estimación:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \hat{\mathbf{w}}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)$$

- Segunda red es la red target:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, $\mathcal{D}, k = 0$

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

$k \leftarrow k + 1$

▷ para cada paso del episodio

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, $\mathcal{D}, k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

▷ para cada episodio

repeat

▷ para cada paso del episodio

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^B [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

repeat

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^B [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

if $(k \bmod C = 0)$ **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

repeat

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^B [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

if $(k \bmod C = 0)$ **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

until S' es terminal

Double DQN

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

repeat

$k \leftarrow k + 1$

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D}

Muestree minibatch aleatorio $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

end for

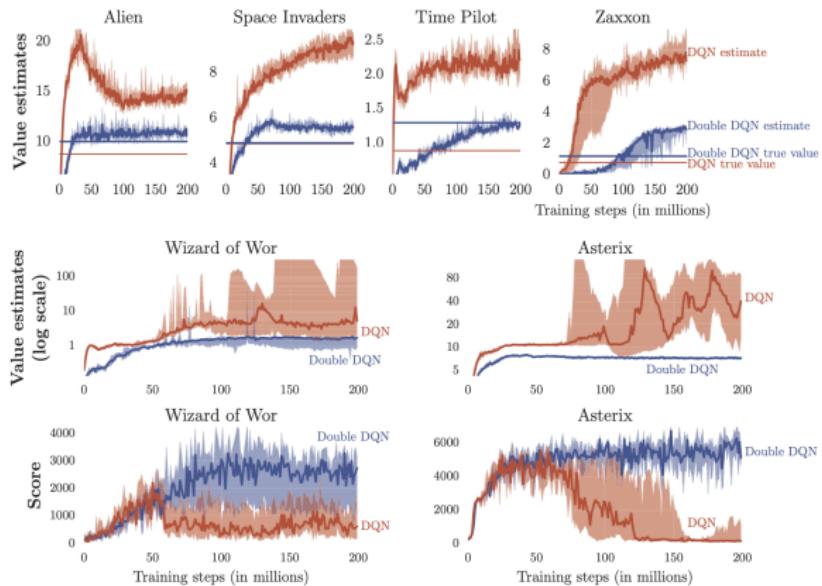
$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^B [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

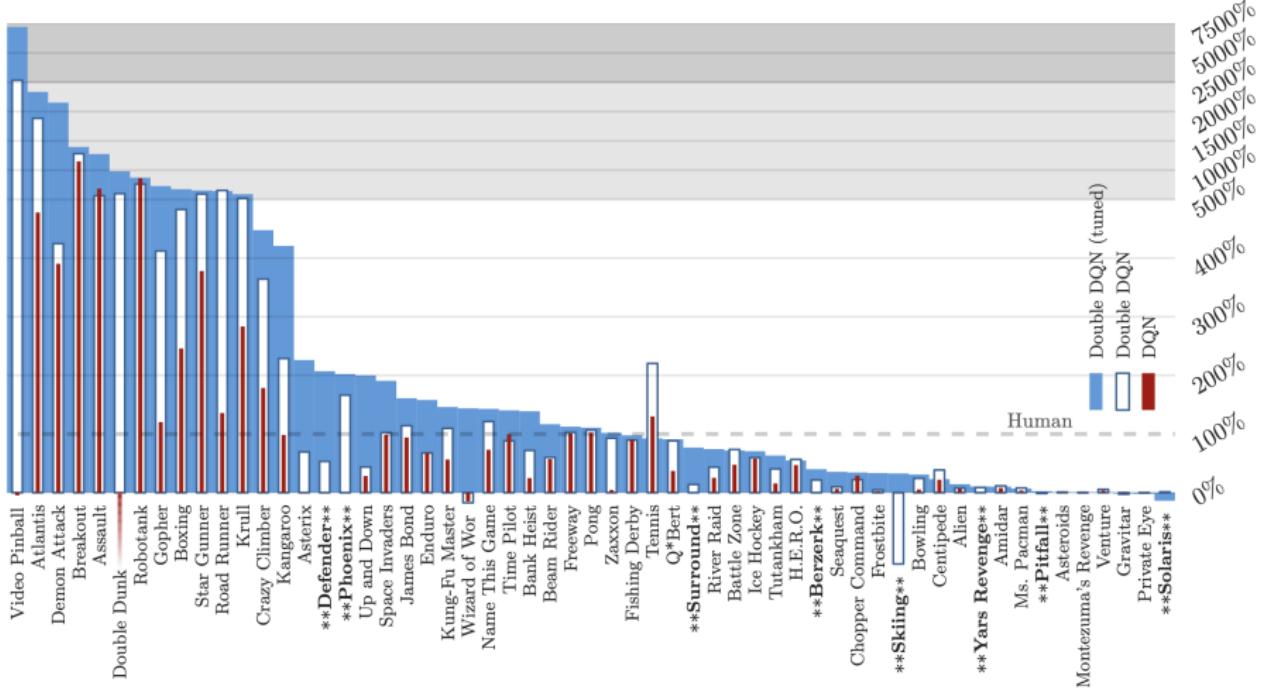
if $(k \bmod C = 0)$ **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

until S' es terminal

until ∞





Prioritized experience replay (Schaul et.al. , 2016)

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:
 - ▶ Usa experiencias en orden diferente al que el agente las encuentra.

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:

- ▶ Usa experiencias en orden diferente al que el agente las encuentra.
- ▶ Probabilidad de usar una experiencia \sim uniforme.

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:
 - ▶ Usa experiencias en orden diferente al que el agente las encuentra.
 - ▶ Probabilidad de usar una experiencia \sim uniforme.
- Agente puede aprender más de algunas experiencias que de otras:

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:
 - ▶ Usa experiencias en orden diferente al que el agente las encuentra.
 - ▶ Probabilidad de usar una experiencia \sim uniforme.
- Agente puede aprender más de algunas experiencias que de otras:
 - ▶ Redundantes..

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:
 - ▶ Usa experiencias en orden diferente al que el agente las encuentra.
 - ▶ Probabilidad de usar una experiencia \sim uniforme.
- Agente puede aprender más de algunas experiencias que de otras:
 - ▶ Redundantes..
 - ▶ Sorpresivas.

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:
 - ▶ Usa experiencias en orden diferente al que el agente las encuentra.
 - ▶ Probabilidad de usar una experiencia \sim uniforme.
- Agente puede aprender más de algunas experiencias que de otras:
 - ▶ Redundantes..
 - ▶ Sorpresivas.
 - ▶ Relevante.

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:
 - ▶ Usa experiencias en orden diferente al que el agente las encuentra.
 - ▶ Probabilidad de usar una experiencia \sim uniforme.
- Agente puede aprender más de algunas experiencias que de otras:
 - ▶ Redundantes..
 - ▶ Sorpresivas.
 - ▶ Relevante.
 - ▶ Utiles dependiendo de cuánto ha aprendido.

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:
 - ▶ Usa experiencias en orden diferente al que el agente las encuentra.
 - ▶ Probabilidad de usar una experiencia \sim uniforme.
- Agente puede aprender más de algunas experiencias que de otras:
 - ▶ Redundantes..
 - ▶ Sorpresivas.
 - ▶ Relevante.
 - ▶ Utiles dependiendo de cuánto ha aprendido.
- Usar experiencias con diferentes frecuencias.

Prioritized experience replay (Schaul et.al. , 2016)

- Experience Replay:
 - ▶ Usa experiencias en orden diferente al que el agente las encuentra.
 - ▶ Probabilidad de usar una experiencia \sim uniforme.
- Agente puede aprender más de algunas experiencias que de otras:
 - ▶ Redundantes..
 - ▶ Sorpresivas.
 - ▶ Relevante.
 - ▶ Utiles dependiendo de cuánto ha aprendido.
- Usar experiencias con diferentes frecuencias.

- Preferir experiencias que causan mayor progreso del aprendizaje.

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- ★ Error entre estimativo actual

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- ★ Error entre estimativo actual y mejor estimativo.

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- ★ Error entre estimativo actual y mejor estimativo.
- ★ Estimativo de $Q(S_t, A_t)$ disponible en $t + 1$.

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- ★ Error entre estimativo actual y mejor estimativo.
- ★ Estimativo de $Q(S_t, A_t)$ disponible en $t + 1$.
- ★ Con aproximación de funciones: actualizar con cambio en \hat{q} .

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- Poca diversidad
 - ★ Error entre estimativo actual y mejor estimativo.
 - ★ Estimativo de $Q(S_t, A_t)$ disponible en $t + 1$.
 - ★ Con aproximación de funciones: actualizar con cambio en \hat{q} .

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- Error entre estimativo actual y mejor estimativo.
 - Estimativo de $Q(S_t, A_t)$ disponible en $t + 1$.
 - Con aproximación de funciones: actualizar con cambio en \hat{q} .
- Poca diversidad → Priorización aleatoria.

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- Error entre estimativo actual y mejor estimativo.
 - Estimativo de $Q(S_t, A_t)$ disponible en $t + 1$.
 - Con aproximación de funciones: actualizar con cambio en \hat{q} .
- Poca diversidad → Priorización aleatoria.
 - Sesgo

- Preferir experiencias que causan mayor progreso del aprendizaje.
 - ▶ Criterio: Magnitud del error TD.

$$|\delta_t| = |R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)|$$

- Error entre estimativo actual y mejor estimativo.
 - Estimativo de $Q(S_t, A_t)$ disponible en $t + 1$.
 - Con aproximación de funciones: actualizar con cambio en \hat{q} .
- Poca diversidad → Priorización aleatoria.
 - Sesgo → Muestreo por importancia

- Neurociencia:

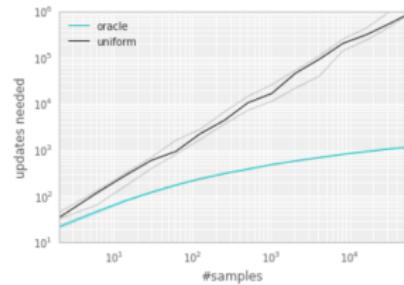
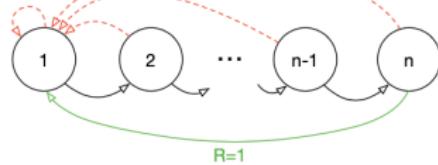
- Neurociencia:
 - ▶ Evidencia de experience replay en el hipocampo en roedores.

- Neurociencia:
 - ▶ Evidencia de experience replay en el hipocampo en roedores.
 - ▶ Hipocampo: aprendizaje, memoria espacio-temporal, navegación.

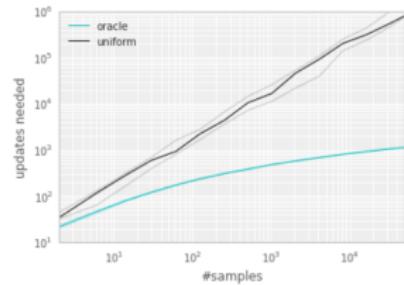
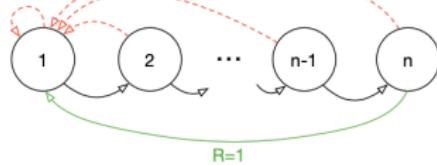
- Neurociencia:

- ▶ Evidencia de experience replay en el hipocampo en roedores.
- ▶ Hipocampo: aprendizaje, memoria espacio-temporal, navegación.
- ▶ Secuencias asociadas con recompensas se repiten más frecuentemente.

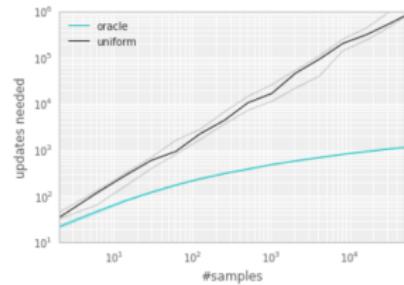
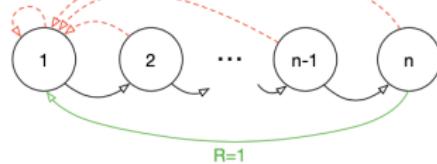
Blind Cliffwalk



Blind Cliffwalk

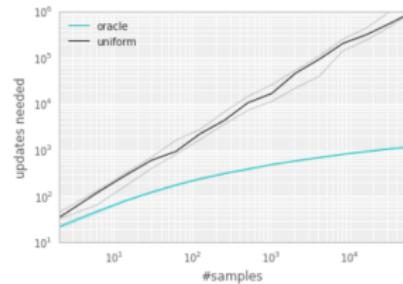
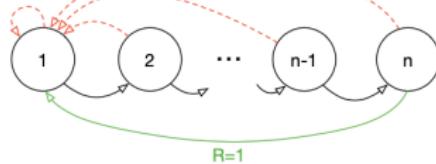


Blind Cliffwalk



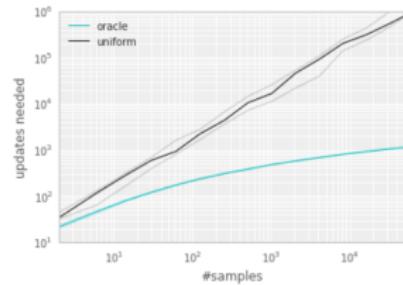
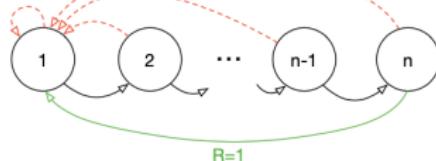
- Q-learning con aproximación lineal.

Blind Cliffwalk



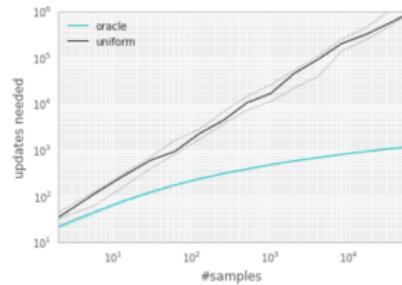
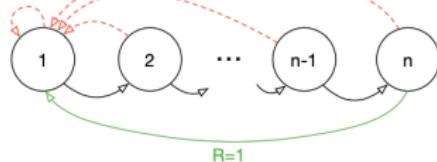
- Q-learning con aproximación lineal.
- Acción correcta/incorrecta alternadas.

Blind Cliffwalk



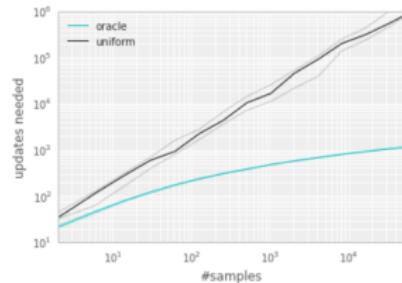
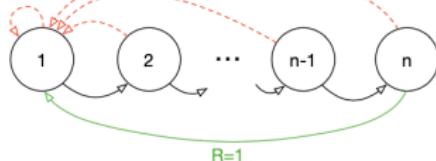
- Q-learning con aproximación lineal.
- Acción correcta/incorrecta alternadas.
- Memoria de replay: todas las $2^{n+1} - 1$ posibles transiciones con frecuencias \approx política aleatoria.

Blind Cliffwalk



- Q-learning con aproximación lineal.
- Acción correcta/incorrecta alternadas.
- Memoria de replay: todas las $2^{n+1} - 1$ posibles transiciones con frecuencias \approx política aleatoria.
- Convergencia: $MSE < 10^{-3}$.

Blind Cliffwalk



- Q-learning con aproximación lineal.
- Acción correcta/incorrecta alternadas.
- Memoria de replay: todas las $2^{n+1} - 1$ posibles transiciones con frecuencias \approx política aleatoria.
- Convergencia: $MSE < 10^{-3}$.
- Oráculo: selecciona experiencia que reduce más error en retrospecto (después de actualizar).

Greedy TD-error prioritization

Greedy TD-error prioritization

- Seleccionar experiencia con mayor magnitud del error TD

Greedy TD-error prioritization

- Seleccionar experiencia con mayor magnitud del error TD.
- Se aprende más de transiciones más sorpresivas.

Greedy TD-error prioritization

- Seleccionar experiencia con mayor magnitud del error TD.
- Se aprende más de transiciones más sorpresivas.
- Actualizar error TD para experiencia seleccionada ($\downarrow\downarrow$ tiempo).

Greedy TD-error prioritization

- Seleccionar experiencia con mayor magnitud del error TD.
- Se aprende más de transiciones más sorpresivas.
- Actualizar error TD para experiencia seleccionada ($\downarrow\downarrow$ tiempo).
- Error TD:
 - ▶ Tabular:

$$\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

Greedy TD-error prioritization

- Seleccionar experiencia con mayor magnitud del error TD.
- Se aprende más de transiciones más sorpresivas.
- Actualizar error TD para experiencia seleccionada ($\downarrow\downarrow$ tiempo).
- Error TD:

- ▶ Tabular:

$$\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

- ▶ DQN:

$$\delta_t = R_t + \gamma \max_a \hat{q}(S_{t+1}, a, \bar{\mathbf{w}}) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Greedy TD-error prioritization

- Seleccionar experiencia con mayor magnitud del error TD.
- Se aprende más de transiciones más sorpresivas.
- Actualizar error TD para experiencia seleccionada ($\downarrow\downarrow$ tiempo).
- Error TD:

- ▶ Tabular:

$$\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

- ▶ DQN:

$$\delta_t = R_t + \gamma \max_a \hat{q}(S_{t+1}, a, \bar{\mathbf{w}}) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

- ▶ DQN doble:

$$\delta_t = R_t + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

Greedy TD-error prioritization

- Seleccionar experiencia con mayor magnitud del error TD.
- Se aprende más de transiciones más sorpresivas.
- Actualizar error TD para experiencia seleccionada ($\downarrow\downarrow$ tiempo).
- Error TD:

- ▶ Tabular:

$$\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

- ▶ DQN:

$$\delta_t = R_t + \gamma \max_a \hat{q}(S_{t+1}, a, \bar{\mathbf{w}}) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

- ▶ DQN doble:

$$\delta_t = R_t + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

- Heap: Retornar máximo toma $O(1)$, actualizar en $O(\log N)$.

Blind Cliffwalk 2

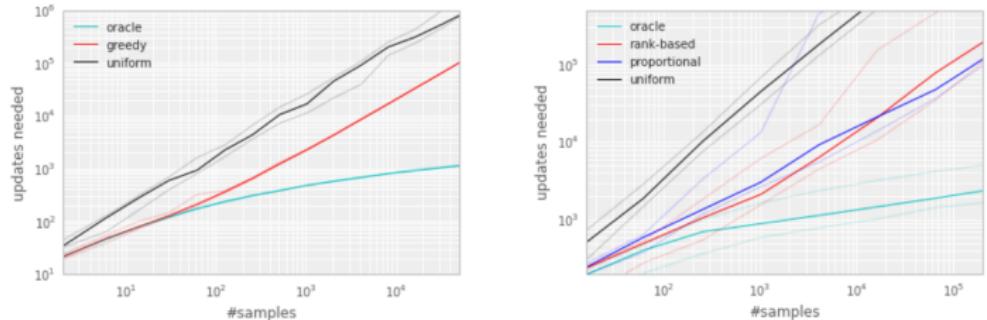


Figure 2: Median number of updates required for Q-learning to learn the value function on the Blind Cliffwalk example, as a function of the total number of transitions (only a single one of which was successful and saw the non-zero reward). Faint lines are min/max values from 10 random initializations. Black is uniform random replay, cyan uses the hindsight-oracle to select transitions, red and blue use prioritized replay (rank-based and proportional respectively). The results differ by multiple orders of magnitude, thus the need for a log-log plot. In both subplots it is evident that replaying experience in the right order makes an enormous difference to the number of updates required. See Appendix B.1 for details. **Left:** Tabular representation, greedy prioritization. **Right:** Linear function approximation, both variants of stochastic prioritization.

Priorización estocástica

- Desventajas de priorización greedy:

Priorización estocástica

- Desventajas de priorización greedy:
 - ▶ Experiencias con error TD inicial \ll no se vuelven a seleccionar.

Priorización estocástica

- Desventajas de priorización greedy:
 - ▶ Experiencias con error TD inicial \ll no se vuelven a seleccionar.
 - ▶ Sensitivo a ruido.

Priorización estocástica

- Desventajas de priorización greedy:
 - ▶ Experiencias con error TD inicial \ll no se vuelven a seleccionar.
 - ▶ Sensitivo a ruido.
 - ▶ Tiende a usar subconjunto pequeño de experiencias repetidamente:

Priorización estocástica

- Desventajas de priorización greedy:
 - ▶ Experiencias con error TD inicial \ll no se vuelven a seleccionar.
 - ▶ Sensitivo a ruido.
 - ▶ Tiende a usar subconjunto pequeño de experiencias repetidamente: **overfitting**.

Priorización estocástica

- Desventajas de priorización greedy:
 - ▶ Experiencias con error TD inicial \ll no se vuelven a seleccionar.
 - ▶ Sensitivo a ruido.
 - ▶ Tiende a usar subconjunto pequeño de experiencias repetidamente: **overfitting**.
- Selección estocástica de acuerdo a:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Priorización estocástica

- Desventajas de priorización greedy:
 - ▶ Experiencias con error TD inicial \ll no se vuelven a seleccionar.
 - ▶ Sensitivo a ruido.
 - ▶ Tiende a usar subconjunto pequeño de experiencias repetidamente: **overfitting**.
- Selección estocástica de acuerdo a:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

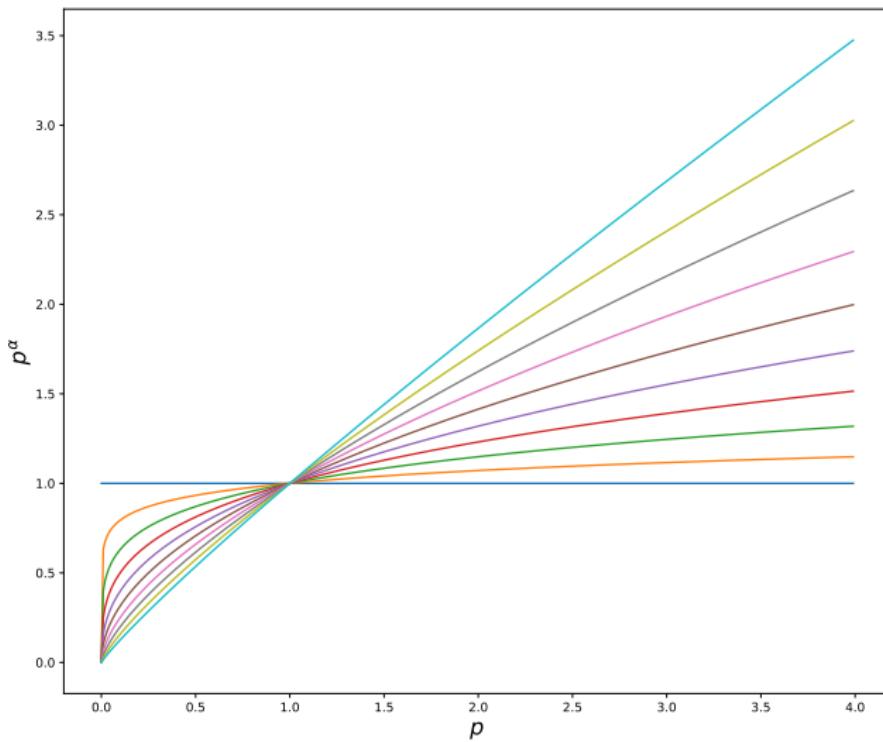
- ▶ $p_i > 0$ es prioridad de transición i .

Priorización estocástica

- Desventajas de priorización greedy:
 - ▶ Experiencias con error TD inicial \ll no se vuelven a seleccionar.
 - ▶ Sensitivo a ruido.
 - ▶ Tiende a usar subconjunto pequeño de experiencias repetidamente: **overfitting**.
- Selección estocástica de acuerdo a:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- ▶ $p_i > 0$ es prioridad de transición i .
- ▶ α : hiperparámetro que indica cuánta priorización.



- Dos variantes:

- Dos variantes:
 - ▶ Proporcional:

$$p_i = |\delta_i| + \epsilon$$

- Dos variantes:

- ▶ Proporcional:

$$p_i = |\delta_i| + \epsilon$$

- ★ $\epsilon \ll$: Evita descartar experiencias con error cero.

- Dos variantes:

- ▶ Proporcional:

$$p_i = |\delta_i| + \epsilon$$

- ★ $\epsilon \ll$: Evita descartar experiencias con error cero.

- ▶ Basada en ranking:

$$p_i = \frac{1}{\text{rank}(i)}$$

- Dos variantes:

- ▶ Proporcional:

$$p_i = |\delta_i| + \epsilon$$

- ★ $\epsilon \ll$: Evita descartar experiencias con error cero.

- ▶ Basada en ranking:

$$p_i = \frac{1}{\text{rank}(i)}$$

- ★ $\text{rank}(i)$ es el ranking de la experiencia i en lista ordenada por $|\delta_i|$.

- Dos variantes:

- ▶ Proporcional:

$$p_i = |\delta_i| + \epsilon$$

- ★ $\epsilon \ll$: Evita descartar experiencias con error cero.

- ▶ Basada en ranking:

$$p_i = \frac{1}{\text{rank}(i)}$$

- ★ $\text{rank}(i)$ es el ranking de la experiencia i en lista ordenada por $|\delta_i|$.
 - ★ Insensitiva a outliers.

Corrección de sesgo

Corrección de sesgo

- Priorización cambia probabilidades de muestras usadas en actualización

Corrección de sesgo

- Priorización cambia probabilidades de muestras usadas en actualización → **sesgo**.

Corrección de sesgo

- Priorización cambia probabilidades de muestras usadas en actualización → **sesgo**.
- Corrección: actualizar con $w_i \delta_i$ donde

$$w_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$$

Corrección de sesgo

- Priorización cambia probabilidades de muestras usadas en actualización → **sesgo**.
- Corrección: actualizar con $w_i \delta_i$ donde

$$w_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$$

- ▶ w_i normalizados por $1/\max w_i$.

Corrección de sesgo

- Priorización cambia probabilidades de muestras usadas en actualización → **sesgo**.
- Corrección: actualizar con $w_i \delta_i$ donde

$$w_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$$

- ▶ w_i normalizados por $1/\max w_i$.
- ▶ β es hiperparámetro, annealing $\beta_0 \rightarrow 1$.

Corrección de sesgo

- Priorización cambia probabilidades de muestras usadas en actualización → **sesgo**.
- Corrección: actualizar con $w_i \delta_i$ donde

$$w_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$$

- ▶ w_i normalizados por $1/\max w_i$.
- ▶ β es hiperparámetro, annealing $\beta_0 \rightarrow 1$.

Corrección de sesgo

- Priorización cambia probabilidades de muestras usadas en actualización → **sesgo**.
- Corrección: actualizar con $w_i \delta_i$ donde

$$w_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$$

- ▶ w_i normalizados por $1/\max w_i$.
- ▶ β es hiperparámetro, annealing $\beta_0 \rightarrow 1$.
- Actualiza **w** con errores más grandes, pero pasos más pequeños.

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Incialice S

 ▷ para cada episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

▷ para cada episodio

Incialice S

repeat

▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Incialice S

repeat

$k \leftarrow k + 1$

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

 Tome acción A , observe R, S' .

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad **máxima**

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad **máxima**

end for

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ - greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad **máxima**

end for

 Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad **máxima**

end for

 Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

 Calcule pesos $z_i = \frac{1}{\max z_i} \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad máxima

end for

 Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

 Calcule pesos $z_i = \frac{1}{\max z_i} \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ - greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad máxima

end for

 Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

 Calcule pesos $z_i = \frac{1}{\max z_i} \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

 Actualice $p_i = |y_i - \hat{q}(s_i, a_i, \mathbf{w})|$

▷ para cada episodio

▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ – greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad máxima

end for

 Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

 Calcule pesos $z_i = \frac{1}{\max z_i} \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

 Actualice $p_i = |y_i - \hat{q}(s_i, a_i, \mathbf{w})|$

end for

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ - greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad máxima

end for

 Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

 Calcule pesos $z_i = \frac{1}{\max z_i} \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

 Actualice $p_i = |y_i - \hat{q}(s_i, a_i, \mathbf{w})|$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{i=1}^B z_i [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ - greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad máxima

end for

 Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

 Calcule pesos $z_i = \frac{1}{\max z_i} \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

 Actualice $p_i = |y_i - \hat{q}(s_i, a_i, \mathbf{w})|$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{i=1}^B z_i [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

if (k mód $C = 0$) **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

 ▷ para cada episodio

 ▷ para cada paso del episodio

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

 Inicialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

 Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ - greedy)

 Tome acción A , observe R, S' .

 Almacene (S, A, R, S') en \mathcal{D} con prioridad máxima

end for

 Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

 Calcule pesos $z_i = \frac{1}{\max z_i} \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

 Actualice $p_i = |y_i - \hat{q}(s_i, a_i, \mathbf{w})|$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{i=1}^B z_i [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

if (k mód $C = 0$) **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

until S' es terminal

Double DQN con priorización proporcional

Require: Función $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$, Memoria \mathcal{D} con capacidad N

Require: $\alpha, \eta, \beta \in (0, 1]$, $\epsilon > 0$, B, C

Incialice $\mathbf{w} = \bar{\mathbf{w}}$, \mathcal{D} , $k = 0$

repeat

Incialice S

repeat

$k \leftarrow k + 1$

for K iteraciones **do**

Escoja A de $\mathcal{A}(S)$, de acuerdo a \hat{q} (ϵ - greedy)

Tome acción A , observe R, S' .

Almacene (S, A, R, S') en \mathcal{D} con prioridad máxima

end for

Muestree minibatch $\mathcal{B} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^B \sim \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

Calcule pesos $z_i = \frac{1}{\max z_i} \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta$

for $(s_i, a_i, r_i, s_{i+1}) \in \mathcal{B}$ **do**

$$y_i = \begin{cases} r_i & \text{si } s_{i+1} \text{ es terminal} \\ r_i + \gamma \hat{q}(S_{t+1}, \arg \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)), \bar{\mathbf{w}}_t) & \text{si } s_{i+1} \text{ no es terminal} \end{cases}$$

Actualice $p_i = |y_i - \hat{q}(s_i, a_i, \mathbf{w})|$

end for

$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{i=1}^B z_i [y_i - \hat{q}(s_i, a_i, \mathbf{w})] \nabla_i \hat{q}(s_i, a_i, \mathbf{w})$

if (k mód $C = 0$) **then** $\bar{\mathbf{w}} = \mathbf{w}$

end if

until S' es terminal

until ∞

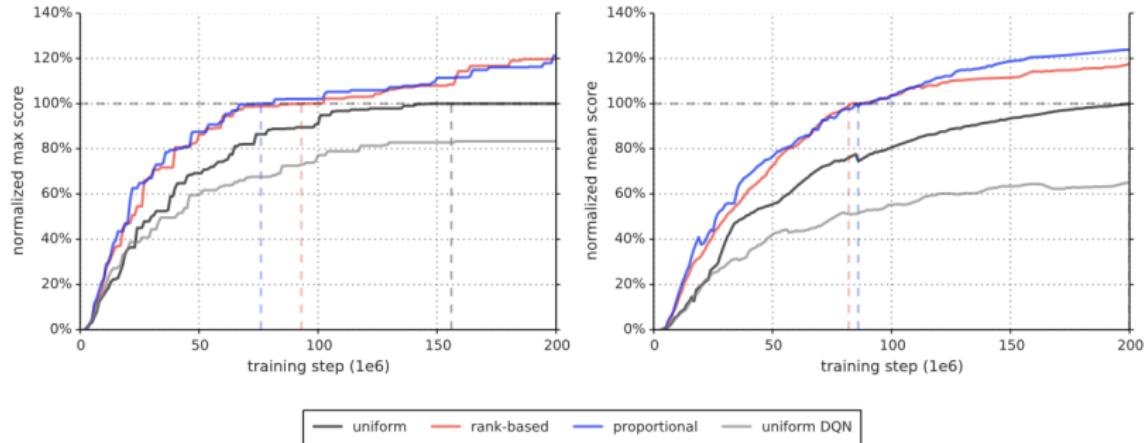
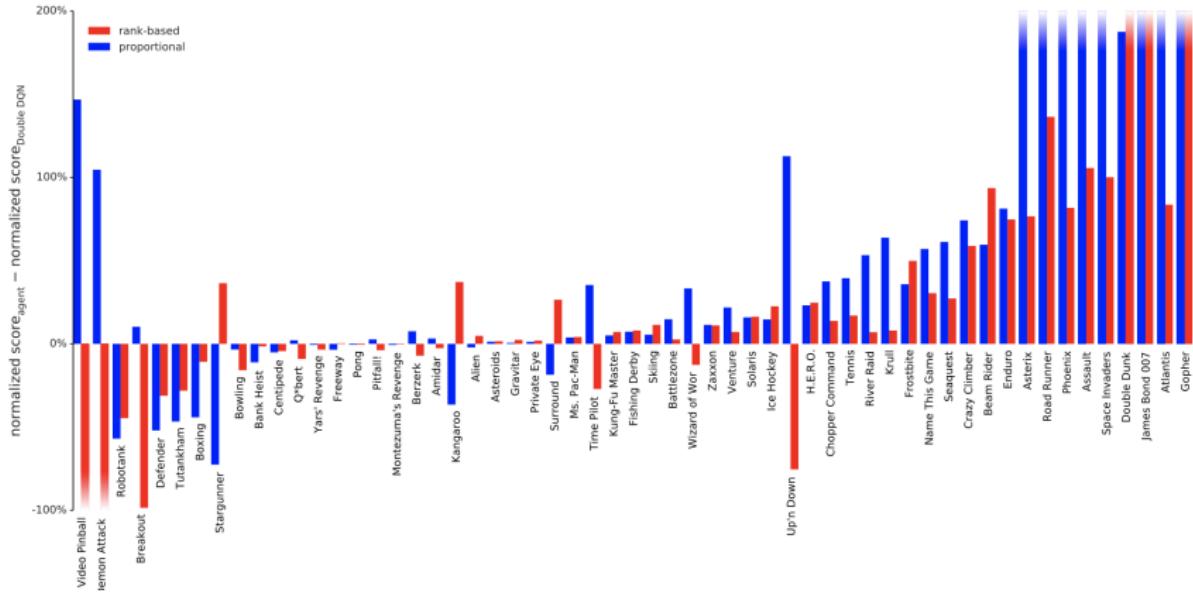


Figure 4: Summary plots of learning speed. **Left:** median over 57 games of the maximum baseline-normalized score achieved so far. The baseline-normalized score is calculated as in Equation 4 but using the maximum Double DQN score seen across training is used instead of the human score. The equivalence points are highlighted with dashed lines; those are the steps at which the curves reach 100%, (i.e., when the algorithm performs equivalently to Double DQN in terms of median over games). For rank-based and proportional prioritization these are at 47% and 38% of total training time. **Right:** Similar to the left, but using the mean instead of maximum, which captures cumulative performance rather than peak performance. Here rank-based and proportional prioritization reach the equivalence points at 41% and 43% of total training time, respectively. For the detailed learning curves that these plots summarize, see Figure 7.



Dueling Networks (Wang et. al. 2016)

Dueling Networks (Wang et. al. 2016)

- Nueva arquitectura de la red que estima $q_* \approx \hat{q}(s, a, \mathbf{w})$.

Dueling Networks (Wang et. al. 2016)

- Nueva arquitectura de la red que estima $q_* \approx \hat{q}(s, a, \mathbf{w})$.
 - ▶ Algoritmo de entrenamiento: DQN, DDQN, ...

Dueling Networks (Wang et. al. 2016)

- Nueva arquitectura de la red que estima $q_* \approx \hat{q}(s, a, \mathbf{w})$.
 - ▶ Algoritmo de entrenamiento: DQN, DDQN, ...
- Ventaja (advantage):

$$a_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$$

Dueling Networks (Wang et. al. 2016)

- Nueva arquitectura de la red que estima $q_* \approx \hat{q}(s, a, \mathbf{w})$.
 - ▶ Algoritmo de entrenamiento: DQN, DDQN, ...
- Ventaja (advantage):

$$\begin{aligned} a_\pi(s, a) &= q_\pi(s, a) - v_\pi(s) \\ &= q_\pi(s, a) - \mathbb{E}_\pi [q_\pi(s, a)] \end{aligned}$$

Dueling Networks (Wang et. al. 2016)

- Nueva arquitectura de la red que estima $q_* \approx \hat{q}(s, a, \mathbf{w})$.
 - ▶ Algoritmo de entrenamiento: DQN, DDQN, ...
- Ventaja (advantage):

$$\begin{aligned} a_\pi(s, a) &= q_\pi(s, a) - v_\pi(s) \\ &= q_\pi(s, a) - \mathbb{E}_\pi [q_\pi(s, a)] \end{aligned}$$

- ▶ Medida relativa de la importancia de cada acción.

Dueling Networks (Wang et. al. 2016)

- Nueva arquitectura de la red que estima $q_* \approx \hat{q}(s, a, \mathbf{w})$.
 - ▶ Algoritmo de entrenamiento: DQN, DDQN, ...
- Ventaja (advantage):

$$\begin{aligned} a_\pi(s, a) &= q_\pi(s, a) - v_\pi(s) \\ &= q_\pi(s, a) - \mathbb{E}_\pi [q_\pi(s, a)] \end{aligned}$$

- ▶ Medida relativa de la importancia de cada acción.
- ▶ $\mathbb{E}_\pi [a_\pi] = 0$.

Dueling Networks (Wang et. al. 2016)

- Nueva arquitectura de la red que estima $q_* \approx \hat{q}(s, a, \mathbf{w})$.
 - ▶ Algoritmo de entrenamiento: DQN, DDQN, ...
- Ventaja (advantage):

$$\begin{aligned} a_\pi(s, a) &= q_\pi(s, a) - v_\pi(s) \\ &= q_\pi(s, a) - \mathbb{E}_\pi [q_\pi(s, a)] \end{aligned}$$

- ▶ Medida relativa de la importancia de cada acción.
- ▶ $\mathbb{E}_\pi [a_\pi] = 0$.
- ▶ Estimar $q_\pi(s, a) = v_\pi(s) + a_\pi(s, a)$

Dueling Networks (Wang et. al. 2016)

- Nueva arquitectura de la red que estima $q_* \approx \hat{q}(s, a, \mathbf{w})$.
 - ▶ Algoritmo de entrenamiento: DQN, DDQN, ...
- Ventaja (advantage):

$$\begin{aligned} a_\pi(s, a) &= q_\pi(s, a) - v_\pi(s) \\ &= q_\pi(s, a) - \mathbb{E}_\pi [q_\pi(s, a)] \end{aligned}$$

- ▶ Medida relativa de la importancia de cada acción.
- ▶ $\mathbb{E}_\pi [a_\pi] = 0$.
- ▶ Estimar $q_\pi(s, a) = v_\pi(s) + a_\pi(s, a)$
- ▶ No en todos los estados la acción modifica significativamente el estado.

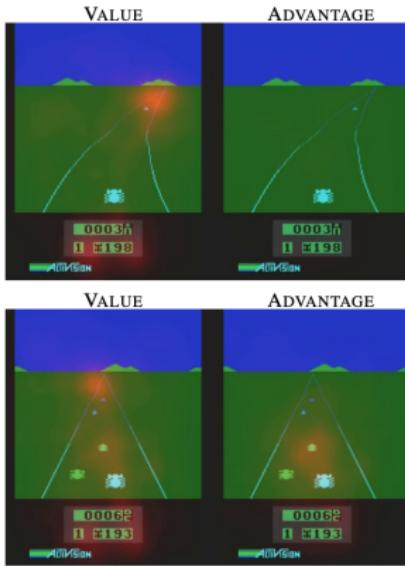
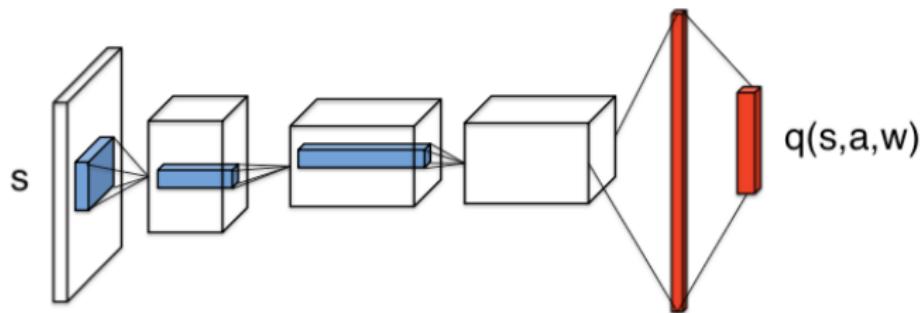
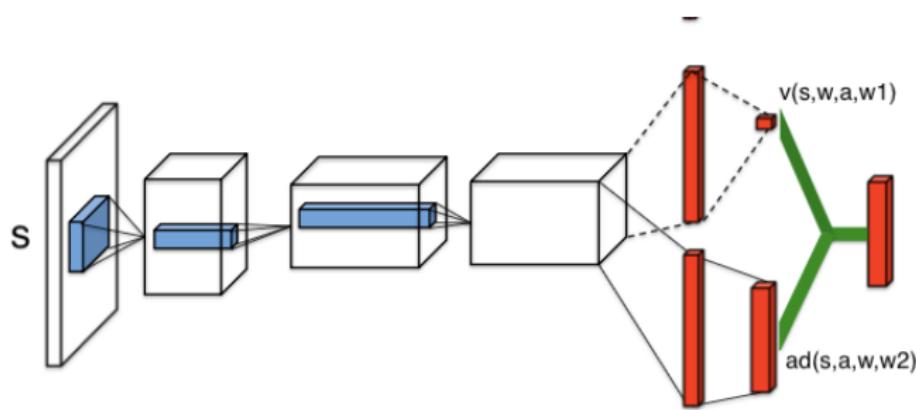
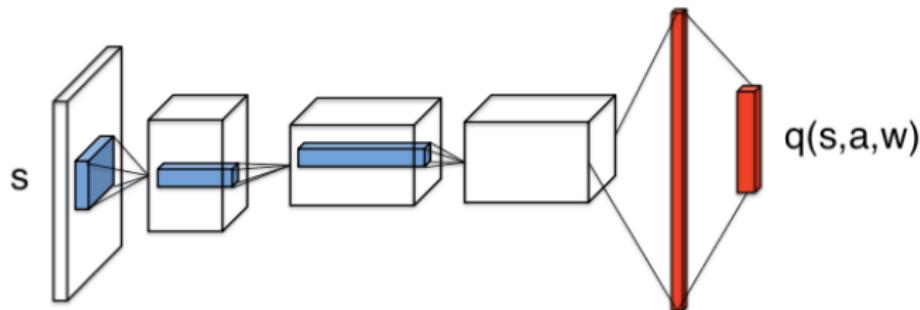


Figure 2. See, attend and drive: Value and advantage saliency maps (red-tinted overlay) on the Atari game Enduro, for a trained dueling architecture. The value stream learns to pay attention to the road. The advantage stream learns to pay attention only when there are cars immediately in front, so as to avoid collisions.

Arquitectura



Arquitectura



- Implementación directa:

$$\hat{q}(s, a_i, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v) + \hat{a}(s, a_i, \mathbf{w}, \mathbf{w}_a), \quad a_i \in \mathcal{A}$$

- Implementación directa:

$$\hat{q}(s, a_i, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v) + \hat{a}(s, a_i, \mathbf{w}, \mathbf{w}_a), \quad a_i \in \mathcal{A}$$

- ▶ \hat{v}, \hat{a} son estimativos.

- Implementación directa:

$$\hat{q}(s, a_i, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v) + \hat{a}(s, a_i, \mathbf{w}, \mathbf{w}_a), \quad a_i \in \mathcal{A}$$

- ▶ \hat{v}, \hat{a} son estimativos.
- ▶ No identifiable \rightarrow no se puede recuperar v_π, a_π a partir de q_π

- Implementación directa:

$$\hat{q}(s, a_i, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v) + \hat{a}(s, a_i, \mathbf{w}, \mathbf{w}_a), \quad a_i \in \mathcal{A}$$

- ▶ \hat{v}, \hat{a} son estimativos.
- ▶ No identifiable \rightarrow no se puede recuperar v_π, a_π a partir de q_π
- ▶ No funciona bien en la práctica.

- Implementación directa:

$$\hat{q}(s, a_i, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v) + \hat{a}(s, a_i, \mathbf{w}, \mathbf{w}_a), \quad a_i \in \mathcal{A}$$

- ▶ \hat{v}, \hat{a} son estimativos.
 - ▶ No identifiable \rightarrow no se puede recuperar v_π, a_π a partir de q_π
 - ▶ No funciona bien en la práctica.
- Para la acción a^* greedy con respecto a q_π :

$$a_\pi(s, a) = q_\pi(s, a^*) - \mathbb{E}_\pi [q_\pi(s, a^*)]$$

- Implementación directa:

$$\hat{q}(s, a_i, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v) + \hat{a}(s, a_i, \mathbf{w}, \mathbf{w}_a), \quad a_i \in \mathcal{A}$$

- ▶ \hat{v}, \hat{a} son estimativos.
 - ▶ No identifiable \rightarrow no se puede recuperar v_π, a_π a partir de q_π
 - ▶ No funciona bien en la práctica.
- Para la acción a^* greedy con respecto a q_π :

$$a_\pi(s, a) = q_\pi(s, a^*) - \mathbb{E}_\pi [q_\pi(s, a^*)] = 0$$

$$\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v)$$

$$+ \left(\hat{a}(s, a, \mathbf{w}, \mathbf{w}_a) - \max_{a' \in \mathcal{A}} \hat{a}(s, a', \mathbf{w}, \mathbf{w}_a) \right)$$



$$\begin{aligned}\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) &= \hat{v}(s, \mathbf{w}, \mathbf{w}_v) \\ &\quad + \left(\hat{a}(s, a, \mathbf{w}, \mathbf{w}_a) - \max_{a' \in \mathcal{A}} \hat{a}(s, a', \mathbf{w}, \mathbf{w}_a) \right)\end{aligned}$$

- Para $a^* = \arg \max_{a' \in \mathcal{A}} \hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v)$:

$$\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v)$$



$$\begin{aligned}\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) &= \hat{v}(s, \mathbf{w}, \mathbf{w}_v) \\ &+ \left(\hat{a}(s, a, \mathbf{w}, \mathbf{w}_a) - \max_{a' \in \mathcal{A}} \hat{a}(s, a', \mathbf{w}, \mathbf{w}_a) \right)\end{aligned}$$

- Para $a^* = \arg \max_{a' \in \mathcal{A}} \hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v)$:

$$\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v)$$

- ▶ Caminos separados para valor y ventaja.

- La red implementa:

$$\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v)$$

$$+ \left(\hat{a}(s, a, \mathbf{w}, \mathbf{w}_a) - \frac{1}{|\mathcal{A}|} \sum_{a'} \hat{a}(s, a', \mathbf{w}, \mathbf{w}_a) \right)$$

- La red implementa:

$$\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v)$$

$$+ \left(\hat{a}(s, a, \mathbf{w}, \mathbf{w}_a) - \frac{1}{|\mathcal{A}|} \sum_{a'} \hat{a}(s, a', \mathbf{w}, \mathbf{w}_a) \right)$$

- ▶ \mathbf{w} : pesos de red convolucional.

- La red implementa:

$$\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v)$$

$$+ \left(\hat{a}(s, a, \mathbf{w}, \mathbf{w}_a) - \frac{1}{|\mathcal{A}|} \sum_{a'} \hat{a}(s, a', \mathbf{w}, \mathbf{w}_a) \right)$$

- ▶ \mathbf{w} : pesos de red convolucional.
- ▶ \mathbf{w}_a : pesos de red de ventaja (totalmente conectada).

- La red implementa:

$$\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v)$$

$$+ \left(\hat{a}(s, a, \mathbf{w}, \mathbf{w}_a) - \frac{1}{|\mathcal{A}|} \sum_{a'} \hat{a}(s, a', \mathbf{w}, \mathbf{w}_a) \right)$$

- ▶ \mathbf{w} : pesos de red convolucional.
- ▶ \mathbf{w}_a : pesos de red de ventaja (totalmente conectada).
- ▶ \mathbf{w}_v : pesos de red de valor (totalmente conectada).

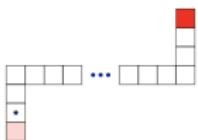
- La red implementa:

$$\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v) = \hat{v}(s, \mathbf{w}, \mathbf{w}_v)$$

$$+ \left(\hat{a}(s, a, \mathbf{w}, \mathbf{w}_a) - \frac{1}{|\mathcal{A}|} \sum_{a'} \hat{a}(s, a', \mathbf{w}, \mathbf{w}_a) \right)$$

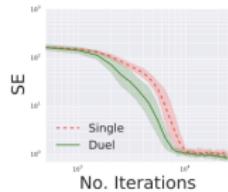
- ▶ \mathbf{w} : pesos de red convolucional.
 - ▶ \mathbf{w}_a : pesos de red de ventaja (totalmente conectada).
 - ▶ \mathbf{w}_v : pesos de red de valor (totalmente conectada).
- Con $\hat{q}(s, a, \mathbf{w}, \mathbf{w}_a, \mathbf{w}_v)$, cualquier algoritmo (DQN, doble DQN, etc).

CORRIDOR ENVIRONMENT



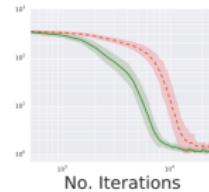
(a)

5 ACTIONS



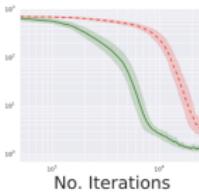
(b)

10 ACTIONS



(c)

20 ACTIONS



(d)

Figure 3. (a) The corridor environment. The star marks the starting state. The redness of a state signifies the reward the agent receives upon arrival. The game terminates upon reaching either reward state. The agent's actions are going up, down, left, right and no action. Plots (b), (c) and (d) shows squared error for policy evaluation with 5, 10, and 20 actions on a log-log scale. The dueling network (Duel) consistently outperforms a conventional single-stream network (Single), with the performance gap increasing with the number of actions.

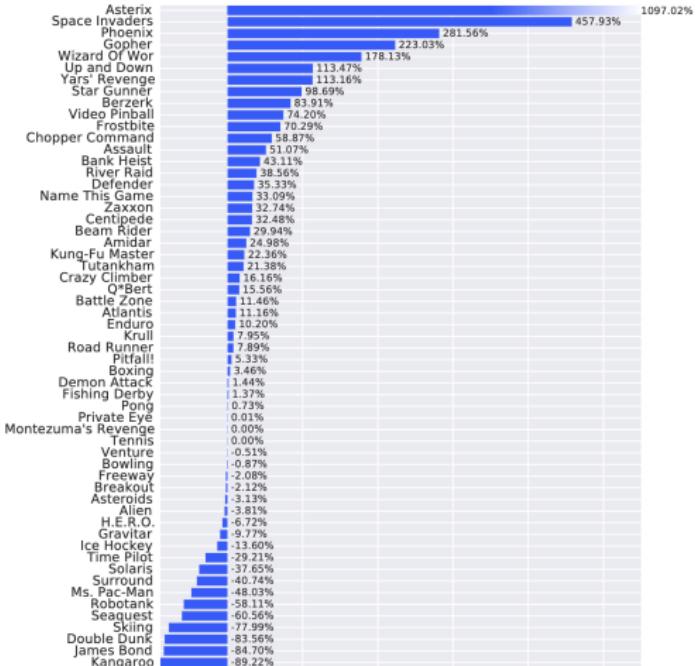


Figure 5. Improvements of dueling architecture over Prioritized DDQN baseline, using the same metric as Figure 4. Again, the dueling architecture leads to significant improvements over the single-stream baseline on the majority of games.