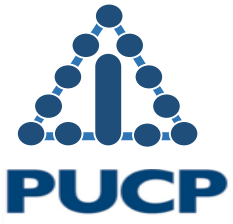


SUMMER CAMP EN IA - 2025

INTRODUCCION AL APRENDIZAJE POR REFUERZO

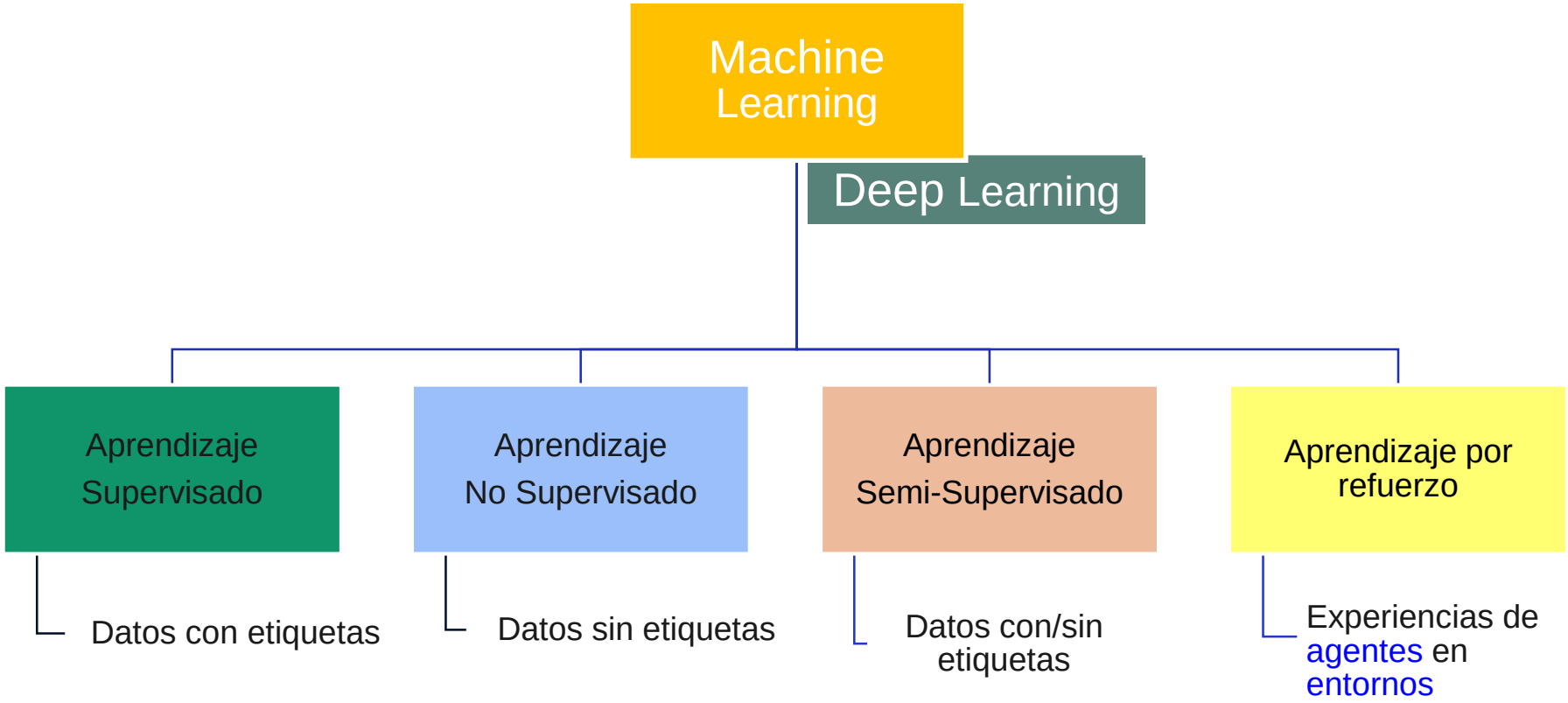
Dr. Edwin Villanueva Talavera

ervillanueva@pucp.edu.pe

Contenido

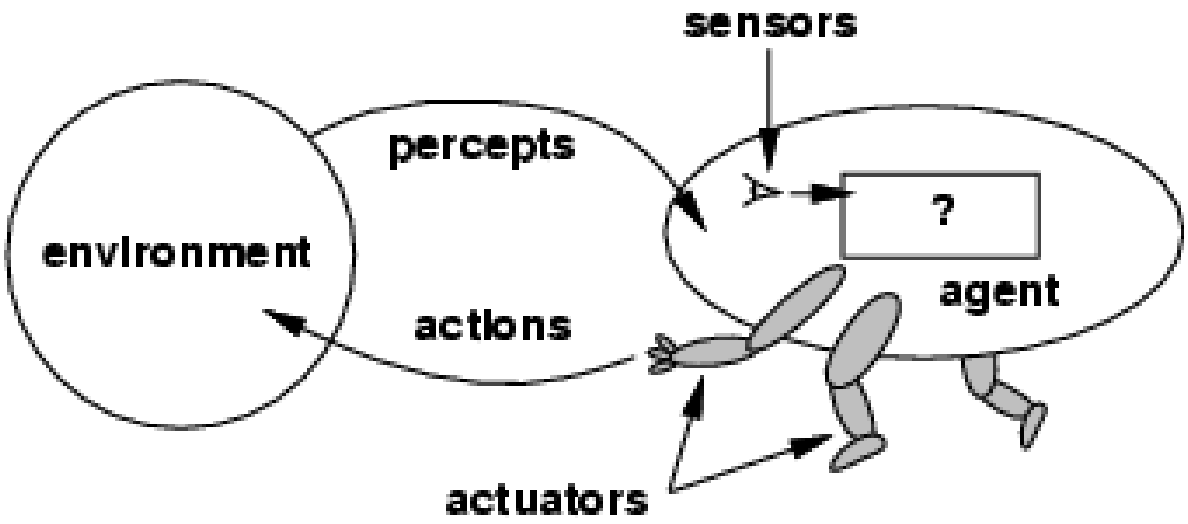
- Procesos de decisiones de Markov (PDM)
- Iteración de Valor
- Aprendizaje por Refuerzo
- Q-learning
- Deep Q-networks

Tipos de Aprendizaje



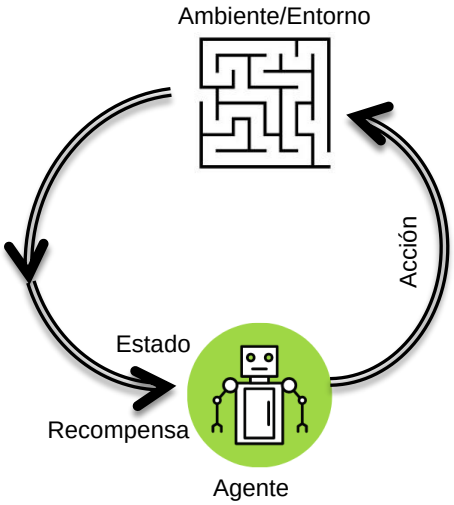
El agente inteligente

- Un agente es algo capaz de percibir su ambiente por medio de sensores y de actuar sobre ese ambiente por medio de actuadores.

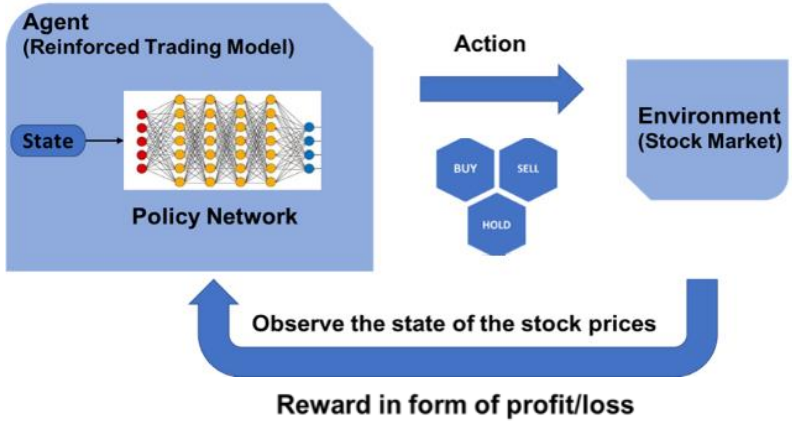


Agentes de Aprendizaje por Refuerzo

Agente de RL que aprende a jugar



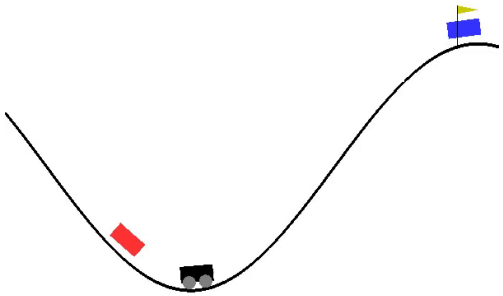
Agente de RL que aprende a invertir

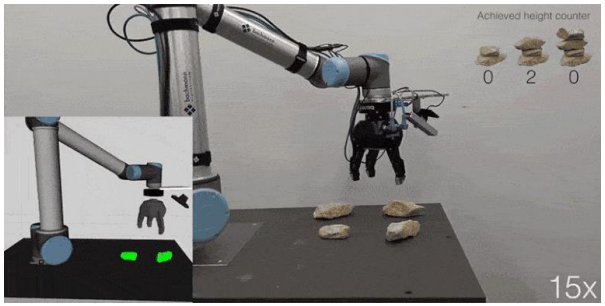


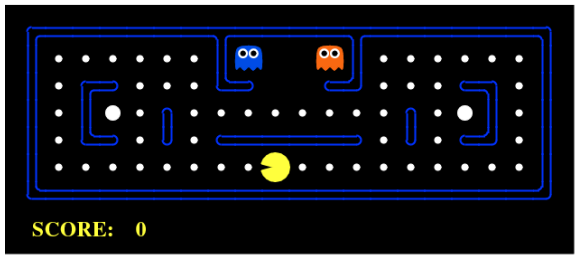
Agentes de Aprendizaje por Refuerzo

En aprendizaje por refuerzo, un agente actúa dentro de un entorno tratando de maximizar las recompensas acumuladas

- El agente necesita ejecutar acciones y recibir recompensas para aprender.
- Objetivo: Encontrar una política de acciones optima.







Procesos de decisiones de Markov

PUCP

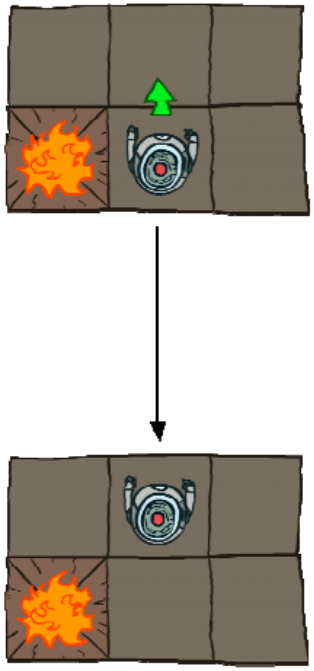
Características:

- Son entornos de problemas de toma de decisiones secuenciales
- El ambiente es No Determinístico: el resultado de la acción del agente es incierta
- Modelo de transición probabilístico: El resultado de cada acción resulta en otro estado con cierta probabilidad
- Transiciones Markovianas: probabilidades de alcanzar un nuevo estado solo dependen del estado actual, no de estados previos.
- Existe una función de recompensa en cada estado s, R(s)

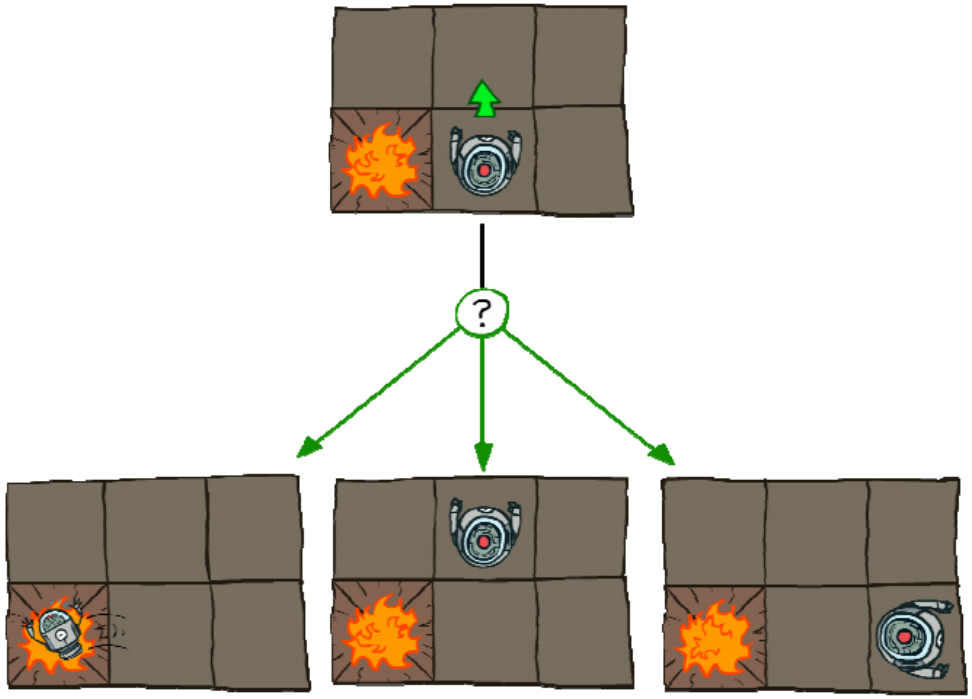
Procesos de decisiones de Markov

Diferencia entre ambiente determinístico y un PDM:

Deterministic Grid World



Stochastic Grid World



Procesos de decisiones de Markov

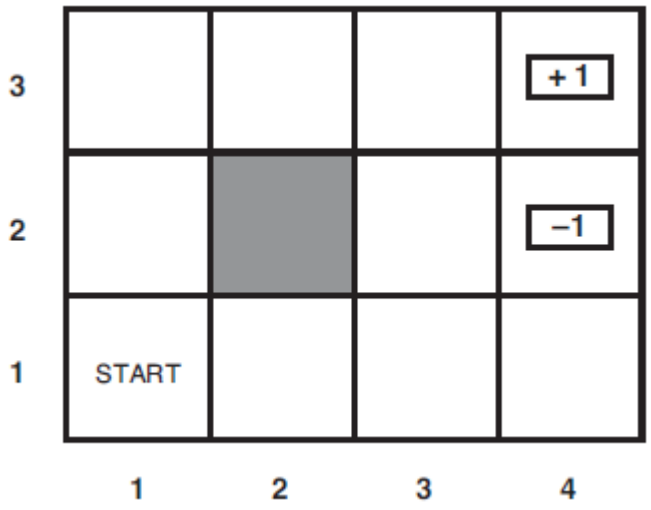
PUCP

Definición de un PDM:

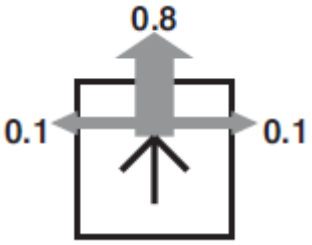
- un estado inicial s_o
- un conjunto de estados $s \in S$
- un conjunto de acciones en cada estado: **Actions(s)**
- un modelo de transición (markoviano) de estados: P(s' | s, a)
  - probabilidad de alcanzarse s' desde s ejecutando acción a.
  - Propiedad de Markov: esa probabilidad depende apenas de s y a y no del histórico de estados y acciones
- Una función de recompensa, puede ser:
  - recompensa de estado $R(s)$, o
  - recompensa de q-estado (estado-accion) $R(s,a,s')$
- (Tal vez) uno o mas estados terminales

Procesos de decisiones de Markov

Ejemplo de PDM:



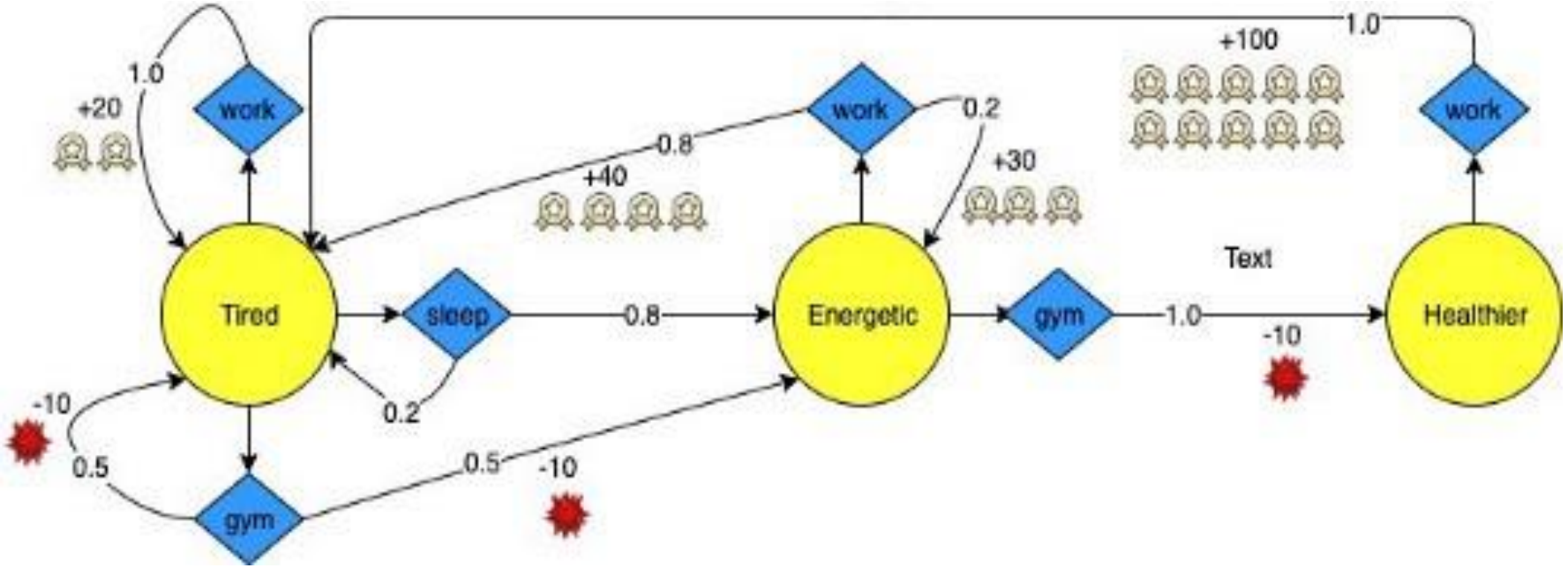
- Estado inicial: START
- Acciones: Ir Arriba, Abajo, Izquierda, Derecha
- Modelo de transición: P(s' | s, a)
  - El agente tiene probabilidad de 0.8 de moverse en la dirección pretendida y 0.2 de moverse a los estados laterales.
- Función de recompensa: R(s)
  - Estados terminales tienen recompensa: +1 y -1
  - Todos los otros estados tienen recompensa: -.04



Si no hubiese incerteza, podríamos usar búsqueda para encontrar la solución óptima.

Procesos de decisiones de Markov

Ejemplo de PDM:

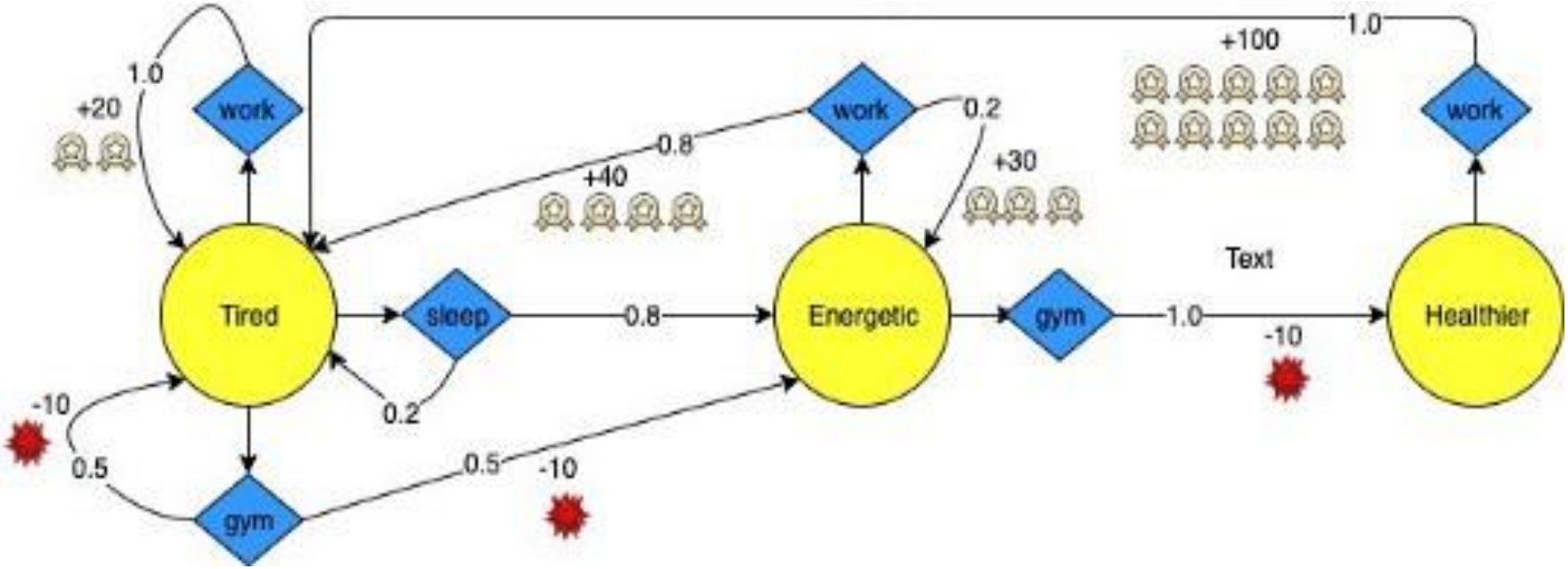


- Estados: Tired, Energetic, Healthier
- Acciones: work, sleep, gym
- Modelo de transición: números en las aristas (probabilidades)
- Función de recompensa: Dinero ganado/perdido en cada estado-acccion

https://medium.com/ai%C2%B3-theory-practice-business/reinforcement-learning-part-3-the-markov-decision-process-9f5066e073a2

Procesos de decisiones de Markov

Cuál es la secuencia de acciones optima para acumular dinero?



política

Procesos de decisiones de Markov

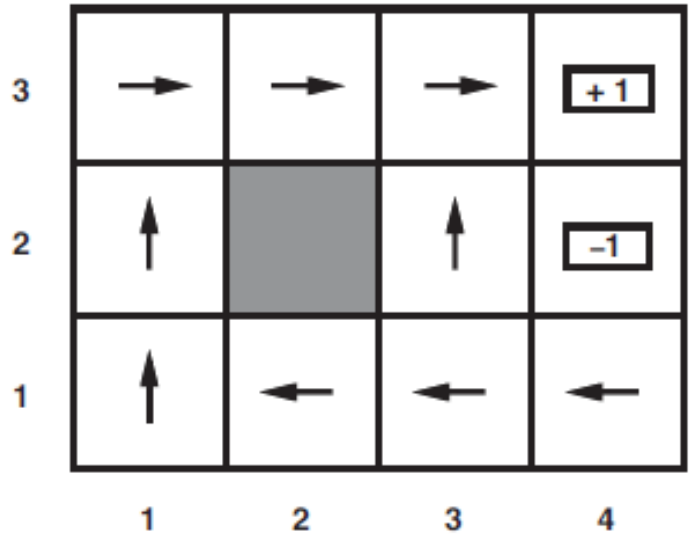
Soluciones de PDMs:

- En un ambiente determinístico con un único agente, la solución es un **plan** = secuencia de acciones óptima.
- En un PDM, la solución es una **política** (denotada por $\pi(s)$) = especifica una acción para cada estado.
  - La **política óptima** es la que produce una Utilidad esperada mas alta posible.
- Si el agente tuviese una política completa, independiente del resultado de sus acciones, el sabría que hacer enseguida.
- Cada vez que una política es ejecutada a partir del estado inicial, la naturaleza estocástica del ambiente llevará a un histórico diferente.

Procesos de decisiones de Markov

Ejemplo:

- Política óptima para recompensa en estados no-terminales $R(s) = -.04$



Procesos de decisiones de Markov

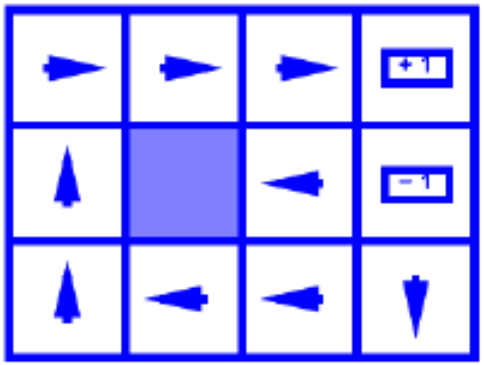
Equilibrio Riesgo-Recompensa:

- El equilibrio entre riesgo y recompensa cambia dependiendo del valor R(s) para los estados no terminales.
- El mantenimiento de un equilibrio cuidadoso entre riesgo y recompensa es una característica de los PDMs que no surge en problemas de búsqueda determinística.

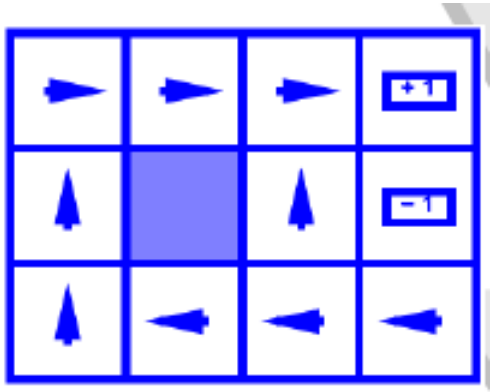
Procesos de decisiones de Markov

Ejemplo:

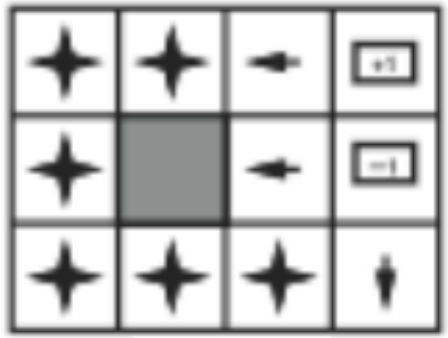
- Políticas óptima para diferentes recompensas de estados no terminales



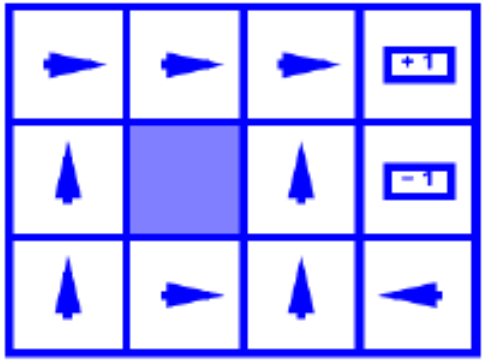
R(s) = -0.01



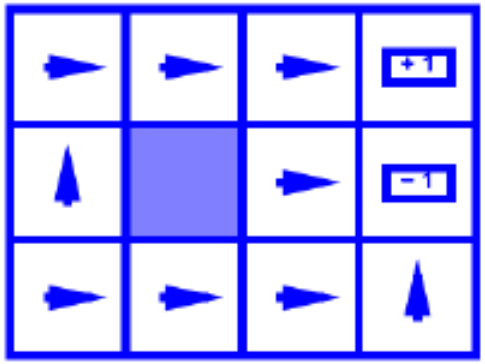
R(s) = -0.03



R(s)>0



R(s) = -0.4



R(s) = -2.0

Procesos de decisiones de Markov

Utilidades de estados

- En un PDM se puede generar múltiples secuencias de estados a partir del estado s
- Para evaluar la utilidad de un estado s en un PDM con política π podemos usar el valor esperado las recompensas descontadas obtenidas con dicha politica :

$$U^{\pi}(s) = E\left[\mathrm{R}(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \right]$$

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right]$$

Procesos de decisiones de Markov

Política Optima

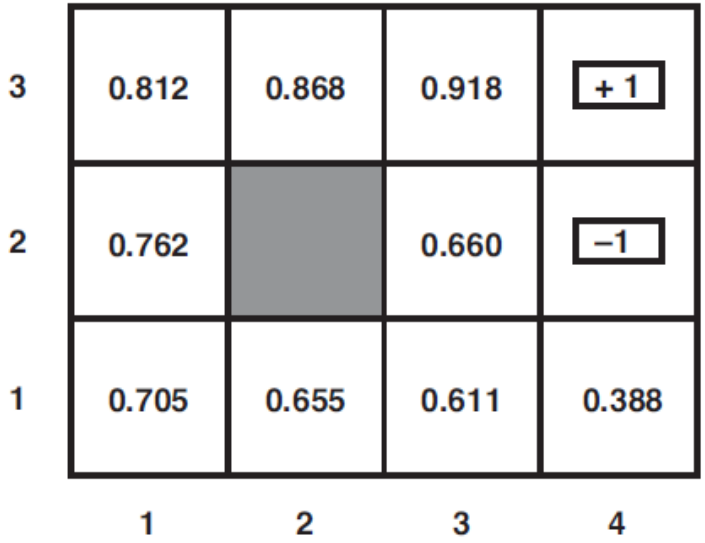
- Una política óptima, $\pi^*(s)$, es aquella que sugiere la acción a ejecutarse en cualquier estado s que genera el mayor valor $U^{\pi}(s)$ de todas las políticas posibles en s:

$$\pi_s^* = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(s)$$

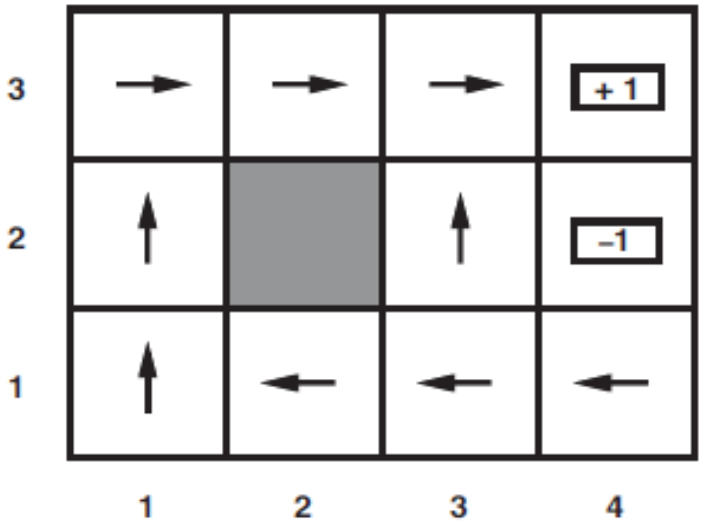
- Definimos la utilidad de un estado *s* como el valor $U(s) = U^{\pi^*}(s)$ esto es, el valor esperado de la suma de recompensas descontadas a partir de **s** dado que el agente ejecuta una política optima

Procesos de decisiones de Markov

Ejemplo de Utilidades



Utilidades (con politica óptima)



Politica óptima

No es lo mismo Utilidad de um estado (U(s)) que Recompensa del estado (R(s) !

En la práctica, Cómo encontramos la política optima?

Recuerde, la política debe especificar una acción para cada estado

Procesos de decisiones de Markov

Ecuación de Bellman:

- Ecuación recursiva definiendo la utilidad de un estado:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

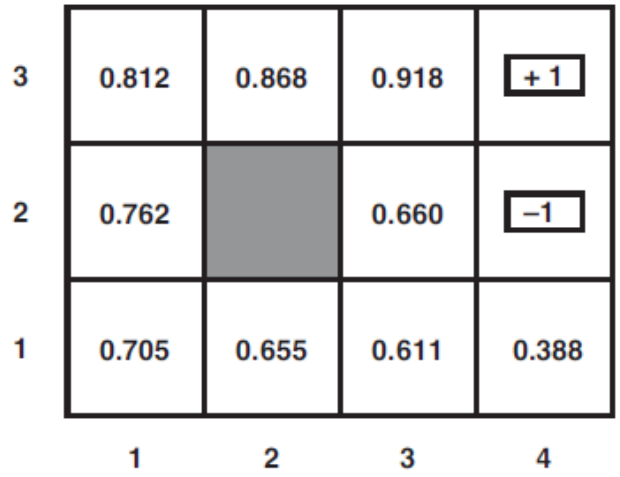
es la recompensa inmediata correspondiente a ese estado + la utilidad descontada esperada del próximo estado, suponiendo que el agente escoja la acción óptima

- Al resolver la ecuación de Bellman y obtener las utilidades *U(s)* para cada estado **s** se puede encontrar la política optima:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

Procesos de decisiones de Markov

Ejercicio Ecuación de Bellman:



Utilidades encontradas al resolver la ecuación de Bellman con R(s) = -0.04, γ=1

¿Que acción ejecutar en (1,1)?

- $U(1,1) = -0.04 + \gamma \max\{\ 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1),$ ↑
  $0.9U(1,1) + 0.1U(1,2),$ ←
  $0.9U(1,1) + 0.1U(2,1),$ ↓
  $0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1),$ →
  $\}$

la acción recomendada por la política optima seria ↑

Algoritmo de Iteración de Valor

Idea: Calcular valores de utilidad óptimos para todos los estados simultáneamente, usando aproximaciones sucesivas.

- Valores iniciales de las utilidades de estados no terminales $U_0(s) = 0$
- Repetir hasta llegar al equilibrio
  - Calcular el lado derecho de la ecuación de Bellman
  - Actualizar la utilidad de cada estado a partir de la utilidad de sus vecinos

Lo que se hace en cada iteración es la actualización de Bellman:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

- Los valores finales serán soluciones para las ecuaciones de Bellman

Iteración de Valor

Evolución de utilidades en el ejemplo de PDM con el algoritmo de Iteración de Valor:

V_2

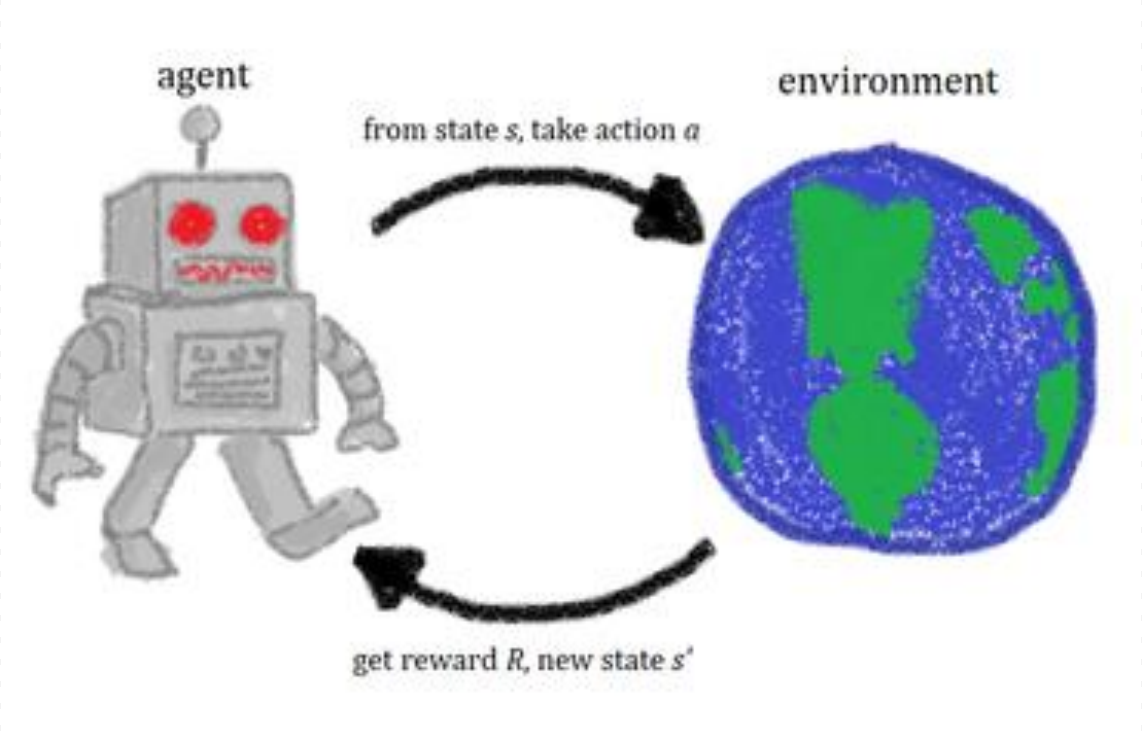
3	0	0	0.72	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

V_3

3	0	0.52	0.78	+1
2	0		0.43	-1
1	0	0	0	0
	1	2	3	4

La información se propaga para adentro a partir de los estados terminales.

APRENDIZAJE POR REFUERZO

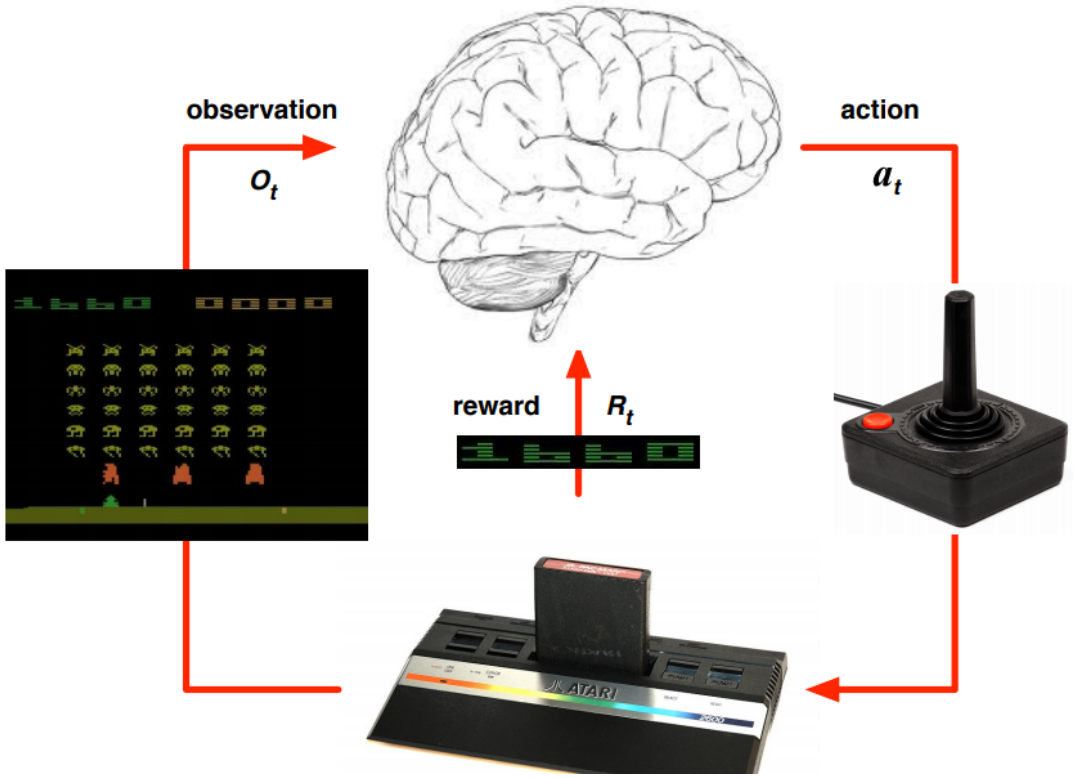


Aprendizaje por Refuerzo

- El agente actúa en un entorno PDM, especificado por:
  - un conjunto de estados $s \in S$
  - un conjunto de acciones $a \in A$
  - un estado inicial s_o
  - (Tal vez) uno o mas estados terminales
- Novedad: El agente no conoce el modelo de transición $P(s' \mid s, a)$ ni la función de recompensa R(s) (recibe ella cuando visita s)
  - El agente necesita ejecutar acciones y recibir recompensas para aprender
- Objetivo: Encontrar uma politica opima $\pi^*(s)$

Aprendizaje por Refuerzo

Ejemplo: Juegos de Atari

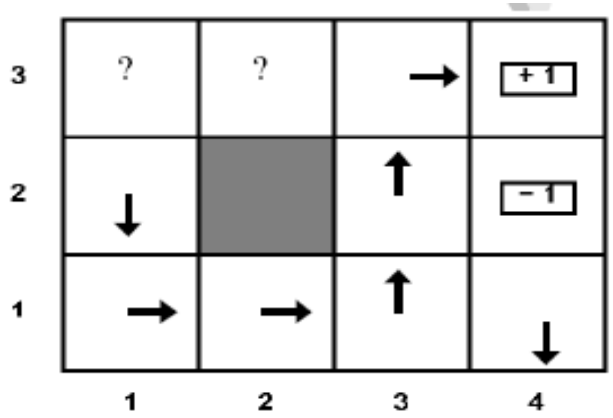


- Entorno desconocido. Se necesita aprender de la experimentación: mover el joystick y ver la pantalla y el score resultante

Aprendizaje por Refuerzo

La exploración es crucial!

- Imagine que el agente encuentra la siguiente politica:



- Sin exploración el agente continuará usando esa política, nunca visitará los otros estados
  - El agente no aprende las utilidades de las mejores regiones ya que con la política actual nunca se visita esas regiones.
  - La política es optima para el modelo aprendido, pero este no refleja el verdadero entorno

Aprendizaje por Refuerzo

Exploración versus Explotación

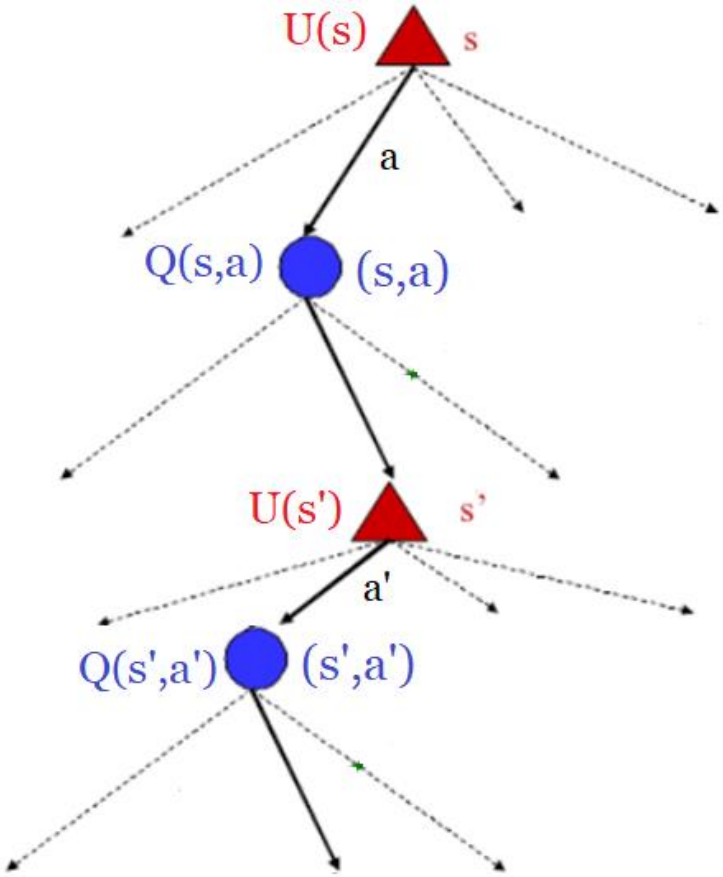
- Exploración: escoger acciones sub-óptimas de acuerdo con el modelo actual para poder mejorar el modelo
- Explotación: escoger las mejores acciones para el modelo actual
- Pura explotación o pura exploración no sirve!. Los agentes deben explorar más al inicio y explotar más al final.

Posibles Soluciones

- Acciones aleatorias una fracción de tiempo (e-*greedy*)
- El agente escoge acciones con probabilidad inversamente proporcional al número de veces que la acción fue ejecutada en el estado. Se implementa a través de una función de exploración f

Aprendizaje por Refuerzo

Algoritmo Q-learning: opera en el espacio de Q-estados



Aprendizaje por Refuerzo

Algoritmo Q-learning

- Denotaremos **$Q(s, a)$** la utilidad de ejecutarse la acción ***a*** en el estado ***s***. La relación con la utilidad del estado es:

$$U(s) = \max_a Q(s, a)$$

- La función $Q(s,a)$ es útil porque a partir de ella podemos calcular directamente la política, sin necesitar de modelo de transición.
- **Ecuación de actualización**: con cada transición observada ***s*** -> ***s'*** con acción ***a*** ejecutada se actualiza $Q(s,a)$:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

- A medida que se realiza iteraciones, los valores $Q(s,a)$ se acercan a los valores de equilibrio:

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q(s', a')$$

Aprendizaje por Refuerzo

Algoritmo Q-learning:

function Q-LEARNING-AGENT($percept$) **returns** an action

inputs: $percept$, a percept indicating the current state s' and reward signal r'

persistent: Q, a table of action values indexed by state and action, initially 0

N_{sa}, a table of frequencies for state–action pairs, initially zero

s, a, r, the previous state, action, and reward, initially null

if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$

if s is not null **then**

increment $N_{sa}[s, a]$

$Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

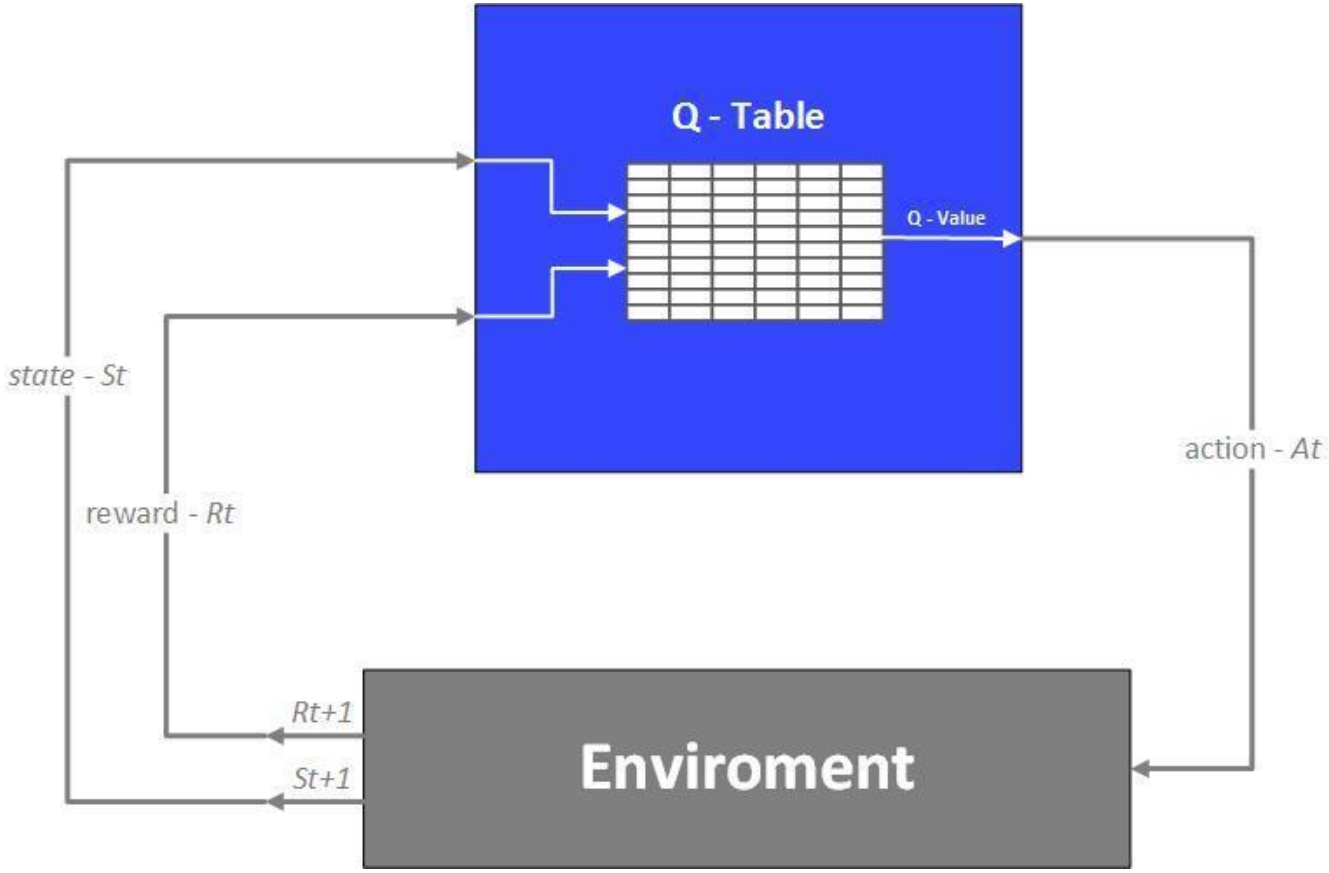
$s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$

return a

f es una función de exploración. Por Ejemplo: $f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$

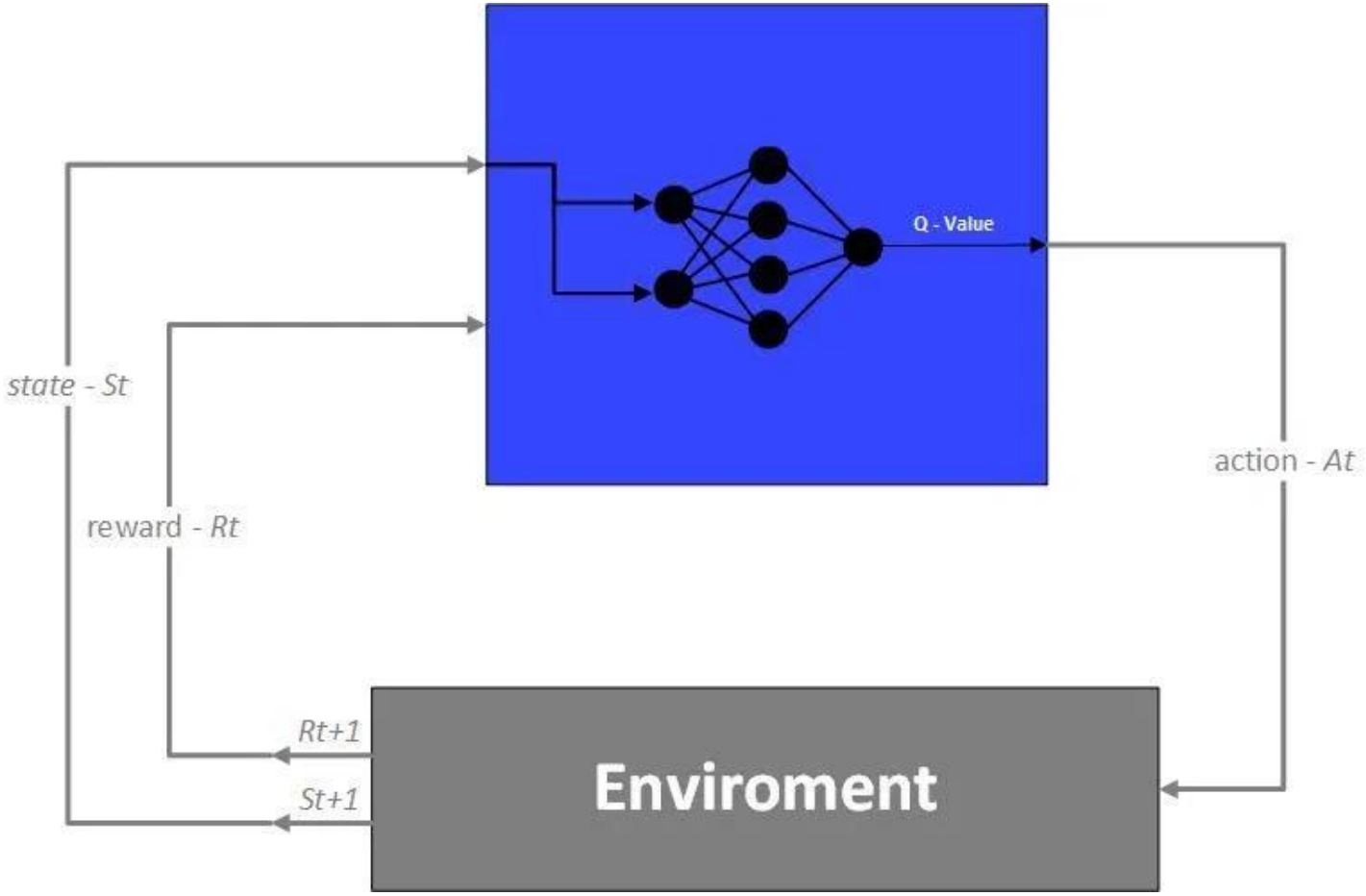
Aprendizaje por Refuerzo

- Un Agente Q-learnig entrenado sigue el siguiente esquema:



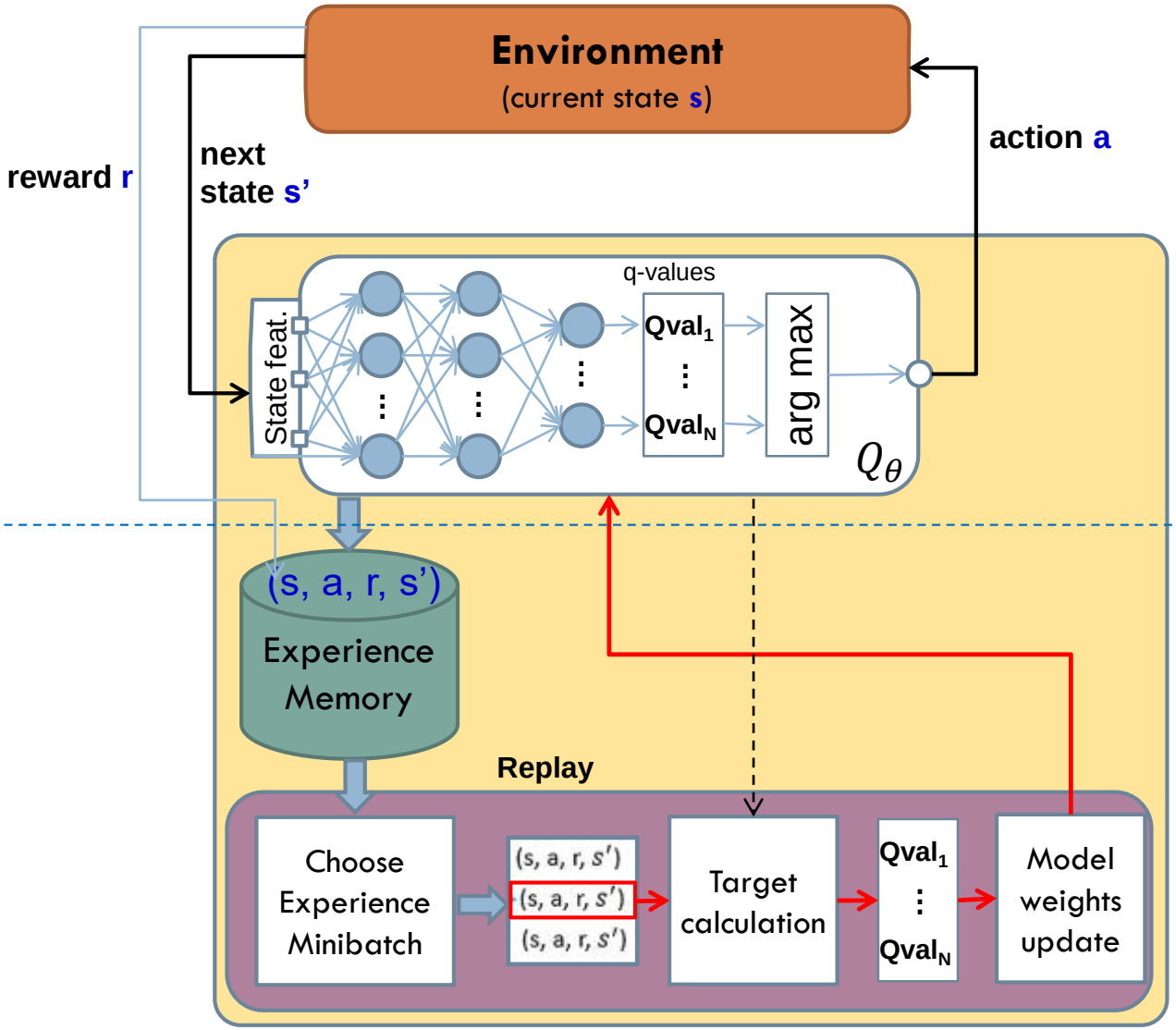
Deep Q-networks (DQN)

- En Deep Q-network se reemplaza la Q-tabla por una Red neuronal para aproximar los valores Q



Deep Q-networks

- Estructura de un agente DQN

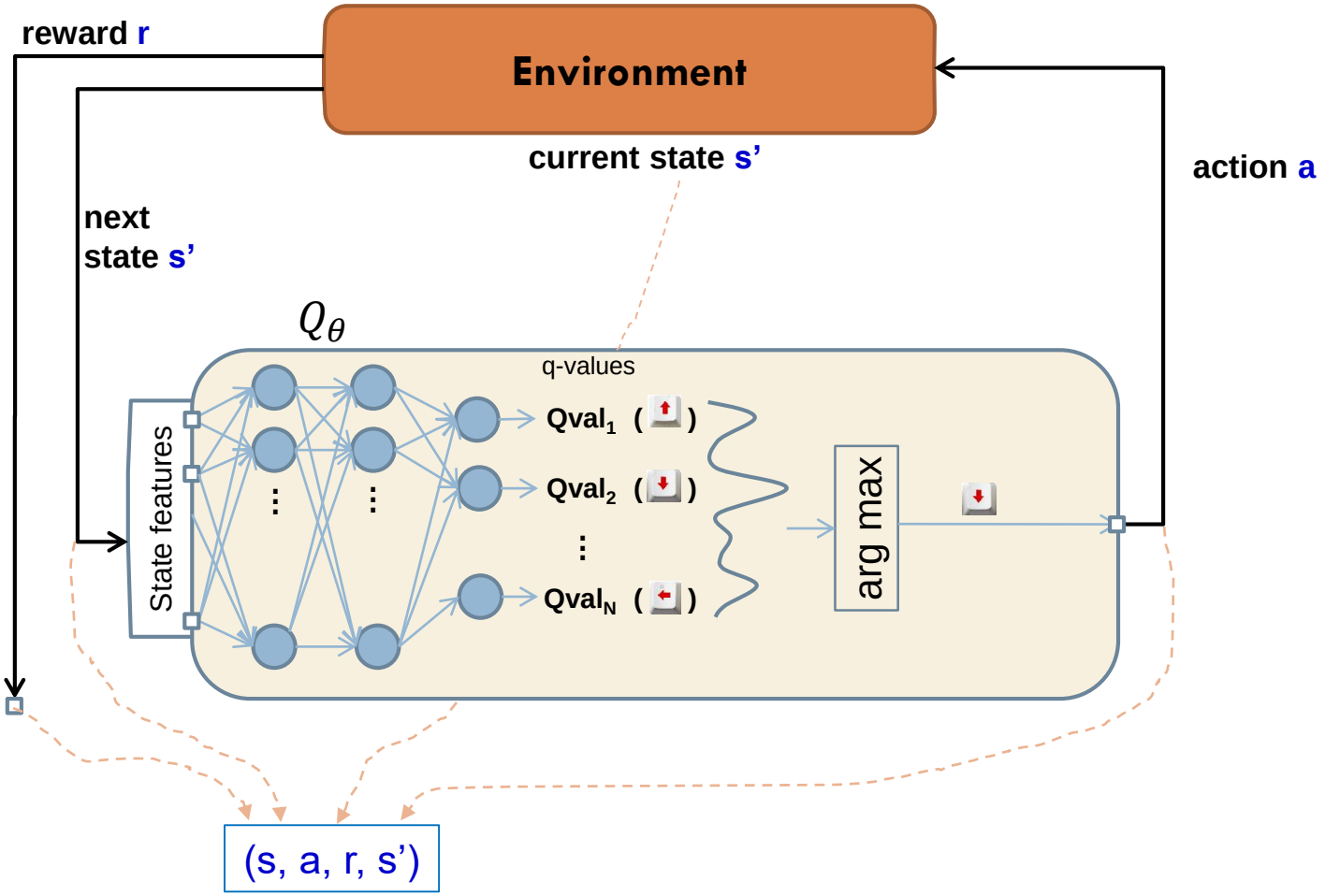


Generación de experiencias

Ajustar modelo (replay)

Deep Q-networks

- Generación de experiencias en DQN



Con cada step del entorno se genera una experiencia que se guarda en la memoria de experiencias

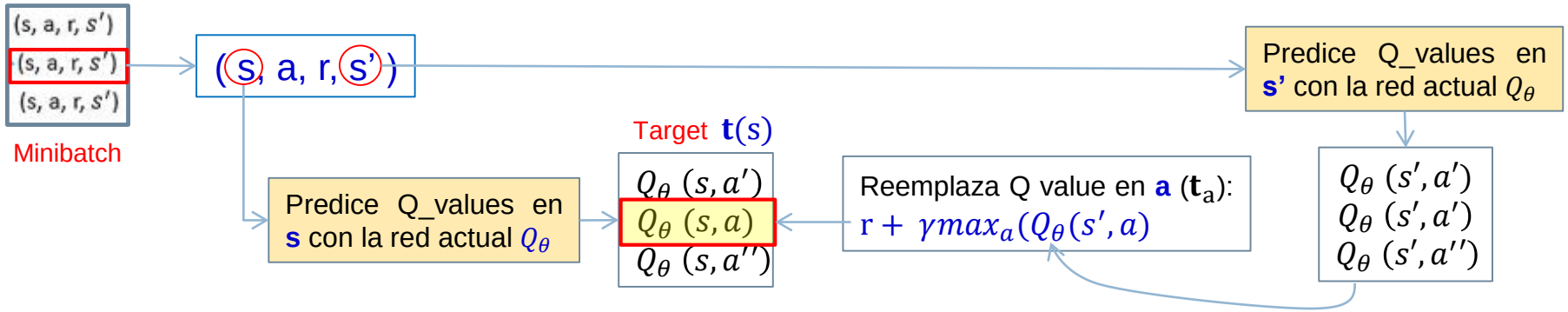
Deep Q-networks

Replay

- Cada step del entorno genera un registro experiencia en la memoria de experiencias:

(state, action, next_state, reward)

- Cada cantidad de steps del entorno se realiza el replay, donde se reajusta el modelo neuronal. En un replay se realiza lo siguiente:
  - Minibatchselection: Se selecciona una pequeña muestra de la memoria de experiencias (minibatch)
  - Target calculation: Se calcula los valores Q deseados (**targets**) para cada **state** en minibatch. Estos son los valores predichos por la red actual (Q_θ) en **state**, salvo el valor Q de la acción ejecutada **a**, el cual debe obedecer Bellman (reward + máximo Q predicho en **next_state**):

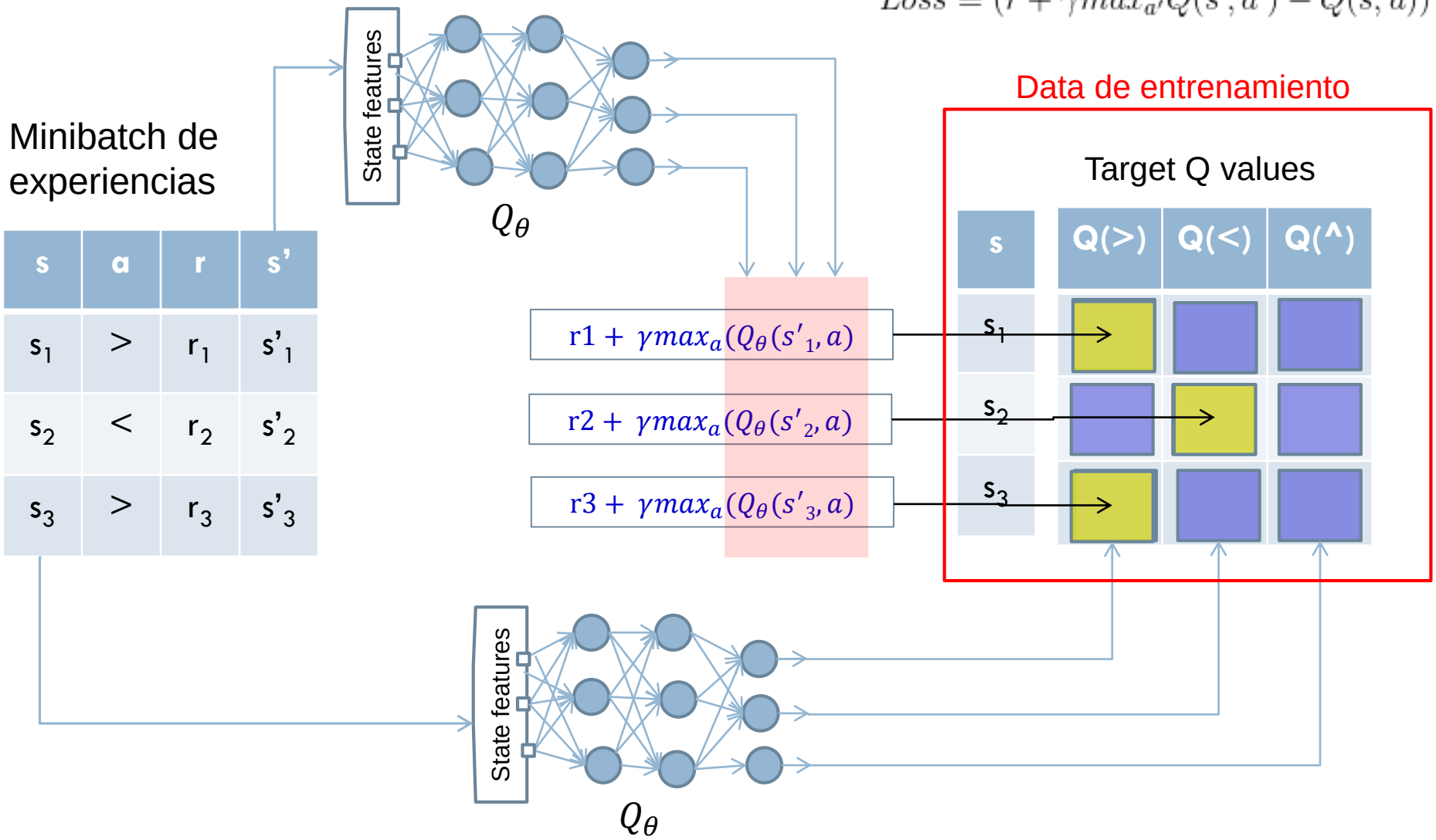


  - Reajuste del modelo: Se reajusta el modelo actual Q_θ con las tuplas de entrenamiento del minibatch [s, $\mathbf{t}(s)$]

Deep Q-networks

Ejemplo de Target Calculation

$$Loss = (r + \gamma max_{a'} Q(s', a') - Q(s, a))^2$$



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 examples $\langle s, \boldsymbol{t} \rangle$
  $\varepsilon \leftarrow \varepsilon * decay$ *# decay the probability of random actions (exploration)*

Referencias y Material complementar

- DQN: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.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?