Aprendizaje por Refuerzo

Índice

- La mayoría de las técnicas de aprendizaje automático desarrollan un aprendizaje de tipo supervisado. Este tipo de aprendizaje se basa en un conjunto de ejemplos (positivos o negativos) que reflejan el comportamiento deseado del sistema.
  - La adquisición de conceptos se basa en un conjunto de ejemplos positivos y negativos del concepto a aprender.
  - Los árboles de decisión y las reglas de clasificación se basan en un conjunto de patrones clasificados que dirigen la creación de estos sistemas.
  - La programación lógica inductiva está dirigida por un conjunto de ejemplos positivos y negativos del predicado que se pretende inducir.
  - El aprendizaje por optimización paramétrica se basa en un conjunto de entrenamiento con valores entrada-salida que reflejan el comportamiento deseado del sistema

- El *aprendizaje por refuerzo* parte de un enfoque totalmente distinto. Se trata de aprender de la experiencia y no de un conjunto de ejemplos.
- Características del aprendizaje por refuerzo:
  - Está dirigido por objetivos. Este objetivo se expresa por una recompensa que devuelve el entorno al realizar una acción sobre él. No se conoce cual es la salida adecuada para el sistema. Tan solo que el efecto que debe producir esta salida sobre el entorno sea tal que se maximice la recompensa recibida a largo plazo.
  - El comportamiento del entorno es, en general, desconocido y puede ser estocástico, es decir, que la evolución del entorno y la recompensa generada pueden obedecer a una cierta función de probabilidad.

- Características del aprendizaje por refuerzo:
  - La recompensa puede tener un cierto retardo. Es decir, la bondad de una acción tomada por el sistema puede que no se refleje hasta un cierto número de evaluaciones posteriores.
  - Dado que el comportamiento del entorno es desconocido, el aprendizaje por refuerzo conlleva una fuerte carga de ensayo y error.
  - Uno de los problemas asociados es el balance exploración-explotación. Se trata de evaluar si es mejor explorar el entorno para mejorar el conocimiento del problema (a costa de empeorar a corto plazo la recompensa obtenida) o explotar el conocimiento acumulado (intentando maximizar la recompensa).
  - Es importante señalar las diferencias respecto al aprendizaje por analogía. El aprendizaje por analogía también está basado en la experiencia, pero se limita a almacenar experiencias pasadas y buscar correspondencias.

- **Agente**: Es el sujeto del aprendizaje por refuerzo. Su funcionamiento consiste en leer el estado del entorno, realizar acciones sobre el entorno y leer las recompensas que producen estas acciones.
- **Entorno**: Es el objeto sobre el que opera el agente. El entorno recibe las acciones del agente y evoluciona. Su comportamiento suele ser desconocido y estocástico. Es el responsable de generar las recompensas asociadas a las acciones y cambios de estado.
- **Política**: Define el comportamiento del agente. Puede verse como un mapeo de estado a acción, es decir, establece las reglas de asociación entre el estado del entorno y la acción a tomar. Puede ser estocástica.

- **Función de refuerzo**: Establece la recompensa a generar en función del estado del entorno y la acción realizada sobre el. Puede ser estocástica. El objetivo del aprendizaje por refuerzo es maximizar la recompensa total obtenida a largo plazo.
- **Función de evaluación** (**función de valor**): refleja una estimación de la recompensa que se va a recibir a partiendo de un cierto estado y siguiendo una cierta política. Esta función sirve de base para escoger la acción a realizar (aquella que conduzca al estado con mayor valor). El objetivo de los algoritmos de aprendizaje por refuerzo es construir esta función.
- **Modelo del entorno**: permite predecir el comportamiento del entorno y aprovechar esta información para resolver el problema.

- **Aprendizaje supervisado**:
  - Se basa en conocer información de tipo "*para el estado X la acción a tomar es A*".
- **Aprendizaje por refuerzo**:
  - Se basa en experiencias del tipo "*para el estado X, la acción A ha producido la recompensa R*".
  - No se sabe si la acción tomada es la mejor posible.
  - Requiere explorar las diferentes acciones para aprender cual es la mejor
- **Retroalimentación evaluativa (evaluative feedback)**:
  - Es una simplificación del problema de aprendizaje por refuerzo.
  - El estado del entorno no tiene influencia. La recompensa depende solo de la acción tomada.

- **El problema de los bandidos de n brazos (n-*armed bandits*)**:
  - En Las Vegas, un bandido con un arma (*bandit with an arm*) se refiere a una máquina tragaperras. Se trata de un juego de palabras (*arm* significa arma y brazo y hace referencia a la palanca de estas máquinas).
  - Un bandido con un brazo es un sistema sobre el que se realiza una acción (tirar de la palanca) y responde con una recompensa (ganar o perder dinero).
  - Un bandido con N brazos es un sistema sobre el que se pueden realizar N acciones y que responde con una recompensa.
  - El problema del bandido con los N brazos consiste en escoger la acción a tomar de manera que la recompensa obtenida sea máxima.
  - A largo plazo, la solución consiste en escoger la acción con mayor recompensa promedio. El problema es saber cual es esa acción.

- **Estimación de la recompensa promedio:**
  - Sea Q*(a) la recompensa promedio de una acción.
  - La estimación de la recompensa promedio es
    - $Q_t(a) = (r_1 + r_2 + \ldots + r_k) / k$
  - Donde *k* es el número de veces que se ha probado la acción *a* y r_i es la recompensa obtenida en cada una de estas pruebas.
  - Según la *ley de los grandes números:*
    - si $k \to \infty$, entonces $Q_t(a) \to Q^*(a)$.
  - Una forma de actualizar la estimación es:
    - $Q_{t+1}(a) = Q_t(a) + (1/k+1) \cdot [r_{k+1} - Q_t(a)]$

- **Estrategia avariciosa (greedy):**
  - Consiste en escoger siempre la acción con mayor recompensa promedio estimada.
  - Pretende explotar al máximo las estimaciones, pero a costa de explorar poco el comportamiento de las acciones.
- **Estrategia ε-greedy**:
  - Consiste en asumir una probabilidad (1-ε) de escoger la acción con mayor estimación, y una probabilidad ε de escoger aleatoriamente entre todas las acciones.
  - Esta estrategia aumenta la exploración en el comportamiento de las acciones, lo que permite que las estimaciones de la recompensa promedio sean mejores y a largo plazo se consiga mejor recompensa.

- **Estrategia softmax:**
  - Se basa en escoger con mayor probabilidad a las acciones con una estimación más alta.
  - Utiliza la función de distribución de Boltzmann

$$\Pi(a) = \frac{e^{Q(a)/\tau}}{\sum_b e^{Q(b)/\tau}}$$

  - donde τ es una variable de control que en los sistemas físicos representa a la temperatura. Si la temperatura es 0 se obtiene la estrategia *greedy*. Al aumentar la temperatura se aumenta la exploración.
  - Se puede utilizar una temperatura constante, o bien una temperatura inicial alta (mucha exploración inicial) que vaya disminuyendo (mayor explotación al final). Esto se conoce como enfriamiento simulado.

- **Estrategia avariciosa con valores iniciales optimistas:**
  - Se basa en partir de unos valores estimados mucho mayores que las recompensas medias reales. De esta forma, al seleccionar una acción se obtiene una recompensa menor que la estimada y se reduce la estimación. Esto hace que la siguiente iteración no escoja esta acción (que dejaría de tener la estimación más alta) lo que favorece la exploración.
  - Esta estrategia consigue una alta exploración inicial seguida de una fuerte explotación final.

- **Estrategia de comparación por refuerzo (reinforcement comparison):**
  - Se basa en comparar la estimación de una acción con la media de todas.
  - Utiliza la regla de selección softmax pero usando la preferencia de la acción (p(a)) en lugar de la estimación de la recompensa media (Q(a)).

$$\Pi(a) = \frac{e^{p(a)}}{\sum_b e^{p(b)}}$$

  - La preferencia de cada acción se actualiza según la ecuación:

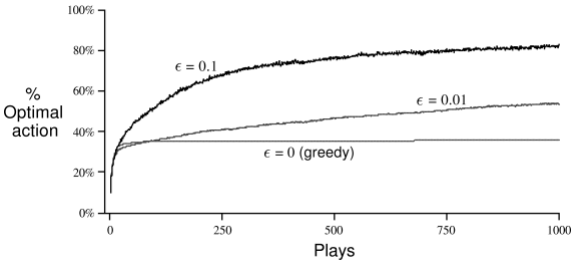
$$p_{t+1}(a) = p_t(a) + \beta \cdot (r_t - \overline{r}_t)$$

  - Donde r_t es la recompensa obtenida en la última selección, $\overline{r}_t$ es la recompensa media de todas las evaluaciones realizadas y β es una constante menor que 1.

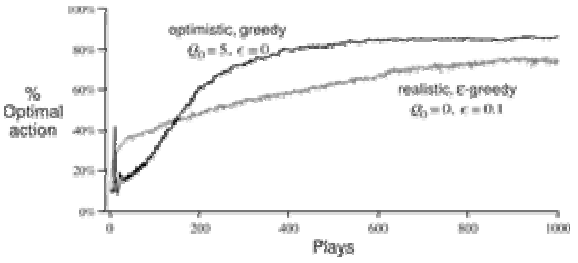
- **Métodos de persecución (pursuit)**
  - Se basa en almacenar tanto la estimación de la recompensa como la probabilidad de elección de cada acción.
  - La estimación de la acción se actualiza siguiendo el método habitual:
    - $Q_{t+1}(a) = Q_t(a) + (1/k+1) \cdot [r_{k+1} - Q_t(a)]$
  - La probabilidad de cada acción se actualiza de manera que se potencie la acción con mayor estimación (a*)
    - $\Pi_{t+1}(a^*) = \Pi_t(a^*) + \beta \cdot [1 - \Pi_t(a^*)]$
    - $\Pi_{t+1}(a) = \Pi_t(a) + \beta \cdot [0 - \Pi_t(a)]$; $a \neq a^*$

- **Ejemplo:**
  - Vamos a suponer un problema de elección entre 10 acciones.
  - La recompensa promedio de cada acción (Q*(a)), se obtiene por medio de una distribución normal de Gauss.
  - La recompensa asociada a cada acción viene dada por una distribución normal, centrada en Q*(a).
  - Se va a calcular la recompensa obtenida en 1000 pasos.
  - Se va a repetir el cálculo anterior 2000 veces para estudiar el comportamiento medio de las diferentes estrategias.

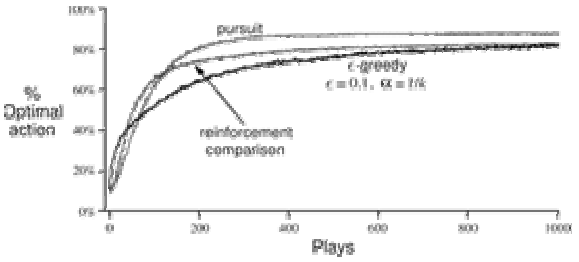
- **Comparación entre *greedy* y *ε-greedy*:**



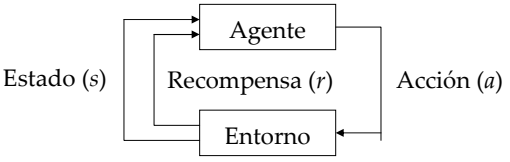
- **Comparación entre *ε-greedy* y *greedy* con valores iniciales optimistas:**



- **Comparación entre *ε-greedy*, *reinforcement comparison* y *pursuit*:**



- **Esquema general**:



- **Consideraciones**:
  - El entorno puede no corresponder exactamente a un entorno físico. Por ejemplo, en un robot los sensores se consideran parte externa al agente.
  - La recompensa se calcula fuera del agente.
  - Se asume que el comportamiento del esquema es discreto, es decir, que está formado por una secuencia de pasos. El caso continuo se trata con intervalos temporales.

- Cada paso consiste en estudiar el estado del entorno (s) y seleccionar una acción (a). El entorno responde con un nuevo estado (s') y una recompensa (r).
- **Política (Π)**: Expresa el comportamiento del agente. Refleja la probabilidad de escoger la acción a al leer el estado s ($\Pi(s,a)$). El objetivo del aprendizaje por refuerzo es modificar la política para maximizar la recompensa recibida a largo plazo.
- **Retorno (R_t)** : recompensa acumulada a partir del paso t.
  - $R_t = r_{t+1} + r_{t+2} + \ldots + r_T$

  donde T es el instante final. El objetivo del aprendizaje por refuerzo es maximizar el retorno esperado.

- **Problema episódico**: problema en el que existe un estado final. Por ejemplo, un juego en el que se gana o se pierde. Un **episodio** consiste en una secuencia de pasos desde un estado inicial a un estado final.
- **Problema continuo:** problema en el que no existe un estado final. El agente continúa su funcionamiento de forma ilimitada ($T \rightarrow \infty$) .
- **Retorno con descuento**: recompensa calculada con un factor de descuento $\gamma \in [0,1]$. El factor de descuento se introduce para evitar que los problemas continuos manejen recompensas infinitas.
  - $R_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \ldots$
- Para unificar el tratamiento de los problemas continuos y episódicos se asume que a partir del estado final, todas las transiciones conducen al estado final y tienen recompensa 0.

- **Propiedad de Markov**: consiste en que la evolución del entorno dependa exclusivamente de su estado y de la acción realizada. Es decir, la evolución del entorno no depende de los estados anteriores ni de las acciones anteriores. Se dice que el entorno *no tiene memoria*.
- **Proceso de Decisión de Markov (MDP)**: es un problema de aprendizaje por refuerzo en el que el entorno cumple la propiedad de Markov.
- **Proceso de Decisión de Markov finito**: es un MDP en el que el espacio de estados y de acciones es finito.

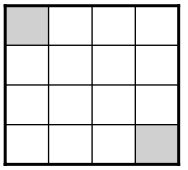
- Un MDP finito se caracteriza por el comportamiento del entorno y por el calculo de las recompensas**:**
  - $P^a_{s,s'}$ (probabilidad de alcanzar s' a partir de s por la acción a)
  - $R^a_{s,s'}$ (recompensa asociada a la transición $s \rightarrow s'$ por la acción a)
- **Función de Valor $V^{\Pi}(s)$**: retorno esperado para el estado s siguiendo la política Π.
- **Función de Acción-Valor $Q^{\Pi}(s,a)$**: retorno esperado para el estado s si se toma la acción a y se sigue la política Π.
- Estas funciones están relacionadas:
  - $V^{\Pi}(s) = \Sigma_a \, \Pi(s,a) \cdot Q^{\Pi}(s,a)$
  - $Q^{\Pi}(s,a) = \Sigma_{s'} \, P^a_{s,s'} \, [R^a_{s,s'} + \gamma \cdot V^{\Pi}(s') \,]$

- **Ecuación de Bellman para la función de Valor**:
  - $V^{\Pi}(s) = \Sigma_a \Pi(s,a) \cdot \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot V^{\Pi}(s')]$
- **Ecuación de Bellman para la función Acción-Valor**:
  - $Q^{\Pi}(s,a) = \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot \Sigma_{a'} \Pi(s',a') \cdot Q^{\Pi}(s',a')]$
- Políticas óptimas: son aquellas que consiguen el mayor retorno.
  - $V^*(s) = \max_{\Pi} V^{\Pi}(s)$
  - $Q^*(s,a) = \max_{\Pi} Q^{\Pi}(s,a)$
- **Ecuaciones de optimalidad de Bellman**
  - $V^*(s) = \max_a \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot V^*(s')]$
  - $Q^*(s,a) = \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot \max_{a'} Q^*(s',a')]$

- Es una solución al problema de aprendizaje por refuerzo.
- Requiere el conocimiento de $P^a_{s,s'}$ y $R^a_{s,s'}$.
- Se transforma la ecuación de Bellman en un algoritmo incremental para calcular la función de valor de una política
  - $V^{\Pi}_{k+1}(s) = \Sigma_a \Pi(s,a) \cdot \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot V^{\Pi}_k(s')]$
- Esta ecuación converge a $V^{\Pi}(s)$ con $k \rightarrow \infty$. Normalmente se para cuando $V^{\Pi}_{k+1}(s) - V^{\Pi}_k(s) < \delta$, siendo δ un valor pequeño.
- A partir de la función de valor se recalcula la política
  - $\Pi'(s) = a \quad tal\ que \quad \max_a \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot V^{\Pi}_{k+1}(s')]$

- El algoritmo de *Iteración de Políticas* (*Policy Iteration*) consiste en una sucesión de iteraciones que permiten calcular la función de valor asociada a una política y, a continuación, mejorar la política a partir de la función de valor:
  - $\Pi_0 \rightarrow V^{\Pi 0} \rightarrow \Pi_1 \rightarrow V^{\Pi 1} \rightarrow \Pi_2 \rightarrow V^{\Pi 2} \rightarrow \ldots \rightarrow \Pi^* \rightarrow V^*$
- El cálculo de la función de valor puede ser muy costoso ya que puede requerir de muchas pasadas. Una modificación de este algoritmo se conoce como *Iteración de Valores* (*Value Iteration*), que consiste en realizar un único paso de evaluación de la política seguido de una actualización de la política
  - $V^{\Pi}_{k+1}(s) = \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot V^{\Pi}_k(s')]$ *tal que* $a = \Pi_k(s)$
  - $\Pi_{k+1}(s) = a$ *tal que* $\max_a \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot V^{\Pi}_{k+1}(s')]$

- Ejemplo
  - Un entorno con 16 estados.
  - 2 estados finales.
  - 4 acciones (izquierda, derecha, arriba y abajo).
  - La recompensa es 0 al llegar a un estado final y -1 al realizar cualquier transición.
  - En los bordes, las acciones dejan el entorno en el mismo estado (es decir, que no se sale del tablero).
  - La política inicial es elegir cualquier acción de forma equiprobable. ($\Pi(s,a) = 0.25$)
  - La función de valor inicial es 0 en todos los estados.



- (1ª evaluación de la política)

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

- (2ª evaluación de la política)

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

- (3ª evaluación de la política)

0	-2.4	-2.9	-3
-2.4	-2.9	-3	-2.9
-2.9	-3	-2.9	-2.4
-3	-2.9	-2.4	0

- (10ª evaluación de la política)

0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0

- (Evaluación de la política en ∞)

0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0

- (Actualización de la política)

	←	←	← ↓
↑	← ↑	← ↓	↓
↑	↑ →	↓ →	↓
↑ →	→	→	

- Nota: esta política se podía haber obtenido desde la 3ª evaluación

- La Programación Dinámica exige conocer el comportamiento del entorno ($P^a_{s,s'}$ y $R^a_{s,s}$) para poder evaluar y actualizar las políticas.
- ¿Qué ocurre cuando el comportamiento del entorno es desconocido?
- Los métodos de Monte Carlo consisten en evaluar una política estudiando las recompensas obtenidas por el agente al actuar sobre el entorno.
- Para eso es necesario realizar numerosos episodios partiendo de diferentes estados iniciales, lo que permite obtener una estimación del retorno esperado a partir de los diferentes estados.
- Cada iteración del algoritmo de Monte Carlo supone la realización de un episodio completo siguiendo la política marcada y, a continuación, actualizar la estimación considerando el retorno obtenido en el episodio.

- El objetivo es aprender la política óptima, lo que se puede realizar a partir de una estimación de la función de valor (V(s)) o de la función acción-valor (Q(s,a)).
- La actualización de una política a partir de la función de valor es la siguiente:
  - $\Pi_{k+1}(s) = a \quad tal\ que \quad \max_a \Sigma_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \cdot V^{\Pi}_{k+1}(s')]$
- La actualización de una política a partir de la función acción-valor es la siguiente:
  - $\Pi_{k+1}(s) = a \quad tal\ que \quad \max_a Q_{k+1}(s,a)$
- En los métodos de Monte Carlo resulta más adecuado utilizar la función de acción-valor (Q(s,a)), ya que no requiere almacenar el modelo de comportamiento del entorno ($P^a_{s,s'}$ y $R^a_{s,s}$).

- La utilización de la función acción-valor (Q(s,a)) tiene algunos inconvenientes:
  - Requiere más memoria que la función de valor (V(s)), ya que para cada estado debe mantener un valor para cada posible acción.
  - Una política avariciosa siguiendo una cierta función acción-valor provoca que para cada estado sólo se evalúa una acción, lo que deja gran parte del espacio sin explorar.
- Para evitar lo segundo, al actualizar la política se utilizan las estrategias presentadas en el problema de los bandidos de N brazos.

- Algoritmo para estimar la función acción-valor:
  a) Generar un episodio siguiendo la política Π.
  b) Para cada estado s que aparezca en este episodio
     - R = retorno alcanzado desde el estado s
     - a = acción realizada en dicho estado
     - Añadir R a la lista de retornos obtenidos desde s con a
     - Q(s,a) = valor medio de la lista
  c) Repetir (a) y (b)
- Existen dos formas de estimar la función acción-valor:
  1) Utilizar tan sólo la primera ocurrencia de cada estado (first-visit)
  2) Utilizar todas las ocurrencias del estado (every-visit).
- Se puede demostrar que la primera versión converge a la estimación correcta de la función acción-valor. La segunda opción aprovecha más información de cada episodio, pero su convergencia es más lenta.

- Los métodos de Monte Carlo permiten aprender la política óptima sin necesidad de conocer el comportamiento del entorno, pero requiere realizar episodios completos para poder actualizar la estimación de V(s) o de Q(s,a).
- La Programación Dinámica permiten actualizar los valores de V(s) o de Q(s,a) estudiando los estados vecinos, sin necesidad de realizar episodios completos, pero necesitan conocer el comportamiento del entorno.
- El aprendizaje por diferencias temporales es una mezcla de estos dos métodos que se basa en utilizar la recompensa obtenida en cada iteración (como Monte Carlo) y la evaluación de los estados vecinos (como la Programación Dinámica).

- El algoritmo de diferencias temporales más sencillo es el siguiente:
  - Tras una transición de s a s' con recompensa r
  - $V(s) = r + V(s') = V(s) + [r + V(s') - V(s)]$
- Si consideramos un factor de descuento γ,
  - $V(s) = V(s) + [r + \gamma \cdot V(s') - V(s)]$
- ¿Qué sentido tiene esta ecuación?
  - $[V(s) - V(s')]$ representa la recompensa esperada en la trasición $s \rightarrow s'$
  - $r - [V(s) - V(s')]$ representa la diferencia entre la recompensa esperada y la encontrada.
  - La ecuación corresponde a una actualización basada en la diferencias encontradas en cada instante.
- Para asegurar la convergencia es necesario considerar un factor $\alpha<1$
  - $V(s) = V(s) + \alpha \cdot [r + \gamma \cdot V(s') - V(s)]$

- El algoritmo anterior se conoce como TD(0).
- El factor a garantiza, además, que el agente se adapte a entornos no estacionarios, es decir, entornos en los que $P^a_{s,s'}$ y $R^a_{s,s}$ dependen del tiempo.
- Algoritmo TD(0):

  Para cada episodio, comenzando en s, repetir
  - a = Π(s) // Acción elegida por la política en el estado s
  - (r, s') = EjecutarAcción(a) // Recompensa y estado tras la transición
  - $V(s) = V(s) + \alpha \cdot [r + \gamma \cdot V(s') - V(s)]$
  - s = s'

  Hasta finalizar episodio

- Algoritmo basado en la función acción-valor (Sarsa):
  - $Q(s,a) = Q(s,a) + \alpha \cdot [r + \gamma \cdot Q(s',a') - Q(s,a)]$
- Algoritmo Sarsa:
  - Inicializar s
  - a = Π(s) // Acción elegida por la política en el estado s

  Repetir
  - (r, s') = EjecutarAcción(a) // Recompensa y estado tras la transición
  - a' = Π(s') // Acción elegida por la política en el estado s'
  - $Q(s,a) = Q(s,a) + \alpha \cdot [r + \gamma \cdot Q(s',a') - Q(s,a)]$
  - s = s', a=a'

  Hasta finalizar episodio

- La filosofía de TD(0) y Sarsa es desarrollar muchos episodios para evaluar una política. Una vez evaluada correctamente se actualiza la política y se vuelve a evaluar. Para generar los episodios se utiliza la política a evaluar. Esto se conoce como métodos *on-policy*.
- Se denominan métodos *off-policy* a aquellos en los que la actualización de los valores no se basa en la política a evaluar sino en una búsqueda directa de la política óptima.
- El algoritmo Q-learning se basa en esta idea:
  - $Q(s,a) = Q(s,a) + \alpha \cdot [\, r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \,]$

- Algoritmo Q-learning:

  Para cada episodio, siendo s el estado inicial, repetir
  - $a = \Pi(s)$ // Elegir acción utilizando una política (p.e. ε-greedy)
  - $(r, s') =$ EjecutarAcción(a) // Recompensa y estado tras la transición
  - $Q(s,a) = Q(s,a) + \alpha \cdot [\, r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a) \,]$
  - $s = s'$

  Hasta finalizar episodio

- Los algoritmos TD(0), Sarsa y Q-learning realizan una estimación del retorno esperado (R_t) de un estado en un único paso:
  - $R_t^{(1)}(s_t) \approx r_{t+1} + \gamma \cdot V_t(s_{t+1})$
- Los métodos de Monte Carlo se basan en estimar el retorno esperado utilizando un episodio completo
  - $R_t(s_t) \approx r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \ldots$
- Una modificación interesante es considerar diferencias temporales con un número de pasos mayor.
  - $R_t^{(2)}(s_t) \approx r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot V_t(s_{t+2})$
  - $R_t^{(3)}(s_t) \approx r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot V_t(s_{t+3})$
- Esto se conoce como algoritmos TD(1), TD(2), etc.

- El algoritmo TD(λ) consiste en considerar el retorno esperado como una media ponderada de todos estos retornos:
  - $R_t^{\lambda}(s_t) = (1-\lambda) \cdot \Sigma\, \lambda^{n-1} R_t^{(n)}$
- Si λ=0 la ecuación anterior corresponde a TD(0). Si λ=1 la ecuación corresponde al método de Monte Carlo.
- Algoritmo TD(λ):

  Para cada episodio, comenzando en s, repetir
  - a = Π(s)
  - (r, s') = EjecutarAcción(a)
  - $\delta = [\, r + \gamma \cdot V(s') - V(s)\,]$
  - e(s) = e(s)+1
  - Para todo s, { $V(s) = V(s) + \alpha \cdot \delta \cdot e(s)$; $e(s) = \gamma \cdot \lambda \cdot e(s)$; }
  - s = s'

  Hasta finalizar episodio