Por refuerzo

Aprendizaje Automático

Víctor de la Cueva
vcueva@itesm.mx

Introducción

- Cuando pensamos en aprendizaje quizá lo primero que se nos viene a la mente es aprender por medio de la interacción con el medio ambiente:
  - Un aprendizaje sin profesor
  - Un aprendizaje sin ejemplos (tal vez por observación)
  - Que se tiene que evaluar o probar si lo hacemos bien o no (de acuerdo a una función)
- Aprender por interacción es una idea fundamental que siempre se encuentra en casi todas las teorías del aprendizaje y la inteligencia.
- El enfoque computacional al aprendizaje por interacción se llama aprendizaje por refuerzo:
  - Está enfocado al aprendizaje dirigido por una meta a partir de la interacción

Aprendizaje por Refuerzo (RL)

- Es aprender qué hacer (i.e. cómo mapear situaciones a acciones) para maximizar una señal numérica de recompensa.
- Al que aprende (agente) no se le dan las acciones a tomar.
  - En su lugar debe descubrir cuáles acciones lo llevan a la mayor recompensa intentándolas
  - En los casos más interesantes las acciones pueden afectar no sólo la recompensa inmediata sino las situaciones siguientes y, por medio de ellas, todas las recompensas subsecuentes
- Estas son las dos características principales en el RL:
  - Búsqueda por prueba y error
  - Recompensa retrasada

El problema en RL

- RL no se define caracterizando los métodos de aprendizaje sino el problema.
  - Cualquier método que es bueno para resolver ese tipo de problema lo consideramos un método de RL.
- La idea básica es colocar un agente que aprende para interactuar con su ambiente con el propósito de alcanzar una meta:
  - Claramente, tal agente debe ser capaz de sensar el estado del ambiente y realizar acciones que afecten dicho estado
  - Debe tener también una meta (o metas) relacionada al estado
- La formulación debe incluir justamente estos 3 aspectos:
  - Sensado, acción y meta, en su forma más simple posible

Aprendizaje supervisado

- En problemas con interacción es normalmente impráctico obtener ejemplos del comportamiento deseado que sean tanto correctos como representativos de todas las situaciones en las cuales el agente tiene que actual.

Dilema explotación-exploración

- Uno de los retos que aparece en RL y no en otro tipo de aprendizaje es la compensación entre exploración y explotación:
  - Para obtener muchas recompensa, un agente de RL debe preferir acciones que ha intentado antes y que ha encontrado que son eficientes para producir recompensa (explotación)
  - Pero para producir tales acciones debe intentar acciones que no ha intentado antes (exploración)
- El agente debe intentar una variedad de acciones y progresivamente favorecer las que parezcan ser las mejores.

Elementos del RL

- Además del agente y el ambiente, se pueden identificar 4 subelementos de un sistema de aprendizaje con RL:
  - Una política (*policy*)
  - Una función de recompensa (*reward function*)
  - Una función de valor (*value function*)
  - (opcional) Un modelo del ambiente

La política

- Una política define la forma en la que el agente se comporta en un tiempo específico.
- A grandes rasgos, una política es un mapeo de estados percibidos en el ambiente a acciones a ser tomadas en esos estados:
  - Es lo que en psicología se llama un conjunto estimulo-respuesta o asociaciones
  - En algunos casos la política puede ser una simple función o tabla, la cual puede requerir un proceso (e.g. búsqueda)
  - La política es el "core" del agente de RL en el sentido de que ella sola es suficiente para determinar el comportamiento
  - En general, las políticas pueden ser estocásticas

La función de recompensa (rf)

- Una función de recompensa define la meta en el problema de RL:
  - A grandes rasgos, mapea cada estado percibido (par estado-acción) del ambiente a un solo número, una recompensa, indicando la deseabilidad de tal estado
  - El objetivo de un agente de RL solitario es maximizar la recompensa total que recibe a largo plazo
  - Define lo que es bueno y lo que es malo para el agente
  - Debe ser inalterable por parte del agente
  - Puede servir como base para alterar la política
    - Ej. Si una acción seleccionada por la política da como resultado una baja recompensa entonces la política se puede cambiar para seleccionar otra en el futuro)
  - En general, puede ser estocástica

La función de valor (vf)

- Mientras la rf indica qué es bueno en el estado inmediato sensado, una vf especifica qué es bueno a largo plazo:
  - En general, el valor de un estado es la cantidad total de recompensa que un agente puede esperar acumular en el futuro, iniciando en un dicho estado
    - Ej. Un estado puede tener una baja recompensa (inmediata) pero un gran valor porque los estados que lo siguen tienen altas recompensas
  - Las acciones se seleccionan con base en el valor
    - Se buscan acciones que nos lleven a estados con valores altos no con recompensas altas
  - Desafortunadamente, es más complicado determinar los valores que las recompensas
    - Las recompensas son básicamente dadas directamente por el ambiente mientras que los valores deben ser estimados y reestimados de las secuencias de observaciones que un agente hace a lo largo de toda su vida
  - El componente más importante de todos los algoritmos de RL es un método para estimar eficientemente los valores

El modelo del ambiente

- Es algo que mimetiza el comportamiento del ambiente
  - Ej. Dado un estado y una acción, el modelo puede predecir el siguiente estado resultante y la siguiente recompensa
- Son normalmente usados en problemas de planificación

Notación y antecedentes

- Para un completo entendimiento de los algoritmos de RL es necesario entender los Procesos de Decisiones de Markov (MDP) debido a que ellos son las base para la notación y los procesos usados en RL
- Iniciaremos es estudio de los MDP mediante su aplicación a Problemas de Decisiones Secuenciales (SDP).

Problemas de decisiones secuenciales

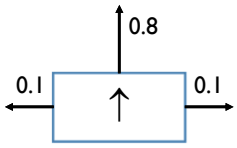
- La utilidad del agente depende de una secuencia de decisiones:
  - Los SDP incorporan utilidades, incertidumbre y sensado.
  - Incluyen los problemas de búsqueda y planificación como casos especiales

Ambiente de ejemplo

- Utilizaremos un ejemplo muy simple (*toy example*) para entender los conceptos de un SDP:
  - Un ambiente de 4 X 3 celdas
  - Tiene movimientos estocásticos para el agente:
    - La dirección indicada se produce con una probabilidad de 0.8 pero el agente se puede mover a la derecha o a la izquierda del movimiento solicitado con una probabilidad de 0.1, respectivamente.
  - La celda oscura no está disponible (obstáculo)
  - Tiene dos estados terminales con recompensas +1 y -1, respectivamente, y todos los demás tienen recompensa -0.4
  - Tiene un estado inicial s0
  - Una colisión con la pared deja al agente sin moverse (en el mismo cuadro)

Ambiente gráfico

3				+1
2				-1
1	Inicio			
	1	2	3	4



Condiciones

- La interacción termina cuando el agente alcanza un estado meta (indicado con +1 y -1)
- Las acciones posibles para el agente en cada estado s, denotadas por Acciones(s) o A(s), son:
  - Up (U)
  - Down (D)
  - Left (L)
  - Right (R)
- Se asume, por el momento, que el ambiente es completamente observable, tal que el agente siempre sabe en qué estado se encuentra

Salida

- Como la salida es estocástica denotamos

$$P(s'|s,a)$$

Como la probabilidad de alcanzar el estado s', si se encuentra en el estado s y se realiza la acción a.

- Asumimos que las transiciones son Markovianas, es decir, la probabilidad de alcanzar el estado s' a partir de s, depende sólo de s.

Función de utilidad

- Para completar la descripción del ambiente se debe especificar la función de utilidad, la cual dependerá de una secuencia de estados (historia de ambiente)
- Por ahora definimos que en cada estado s, el agente recibe una recompensa R(s), la cual puede ser positiva o negativa
- La utilidad para una secuencia de estados es (por ahora) la suma de las recompensas recibidas
  - Ej. Si alcanza el estado +1 después de 10 pasos, su utilidad total será 0.6
  - La recompensa negativa de -0.04 da al agente el incentivo de alcanzar el +1 lo más rápido posible

Proceso de decisión de Markov (MDP)

- Es un problema de decisiones secuenciales para un ambiente completamente observable y estocástico, con un modelo de trasiciones Markovianas y recompensas aditivas
- Consiste de:
  - Un conjunto de estados (con un estado inicia s_0)
  - Un conjunto de acciones A(s) en cada estado s
  - Un modelo de transición P(s'|s,a)
  - Una función de recompensa R(s)

Solución de un MDP

- Una solución a este tipo de problema es lo que debería hacer el agente en cada estado
- Una solución de este tipo es llamada una política
  - Es tradicional denotar una política por π
  - $\pi(s)$ es la acción recomendada por la política π en el estado s
- Si un agente tiene una política completa, sin importar cuál es la salida de una acción, el agente siempre sabe qué hacer a continuación.

Política óptima

- La calidad de una política se mide por la utilidad esperada de las secuencias de estados generadas por tal política
- Una política óptima es la que obtiene la mayor utilidad esperada
  - Se usa π^* para denotar la política óptima
- Dada π^*, el agente decide qué hacer consultando su percepción actual, la cual le dice el estado actual s, y ejecutando la acción $\pi^*(s)$
- Objetivo: encontrar una política óptima

Utilidad de una secuencia

- Para una secuencia estacionaria hay 2 formas coherentes de calcular su utilidad:
  - Recompensas aditivas

$$U_h([s_0,s_1,s_2,\ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

  - Recompensas con descuento

$$U_h([s_0,s_1,s_2,\ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

  donde γ es llamado factor de descuento y es un valor entre 0 y 1
- Si $\gamma = 1$, la recompensa con descuento se convierte en recompensa aditiva por lo que se trata en realidad de un caso especial

Políticas óptimas y la utilidad de los estados

- Si usamos recompensas con descuentos podemos comparar políticas mediante la comparación de utilidades esperadas obtenidas cuando se ejecutan dichas políticas
- Asumimos que el agente está en un estado inicial s y definimos S_t (variable aleatoria) como el estado que el agente alcanza en el tiempo t cuando se ejecuta una política particular π
  - Obviamente, $S_0 = s$, el estado en el que el agente está actualmente
- La distribución de probabilidad sobre las secuencias de estados $S_1, S_2, \ldots$, es determinada por el estado inicial s, la política π y el modelo de transición del ambiente.

Problemas en RL

- Los problemas en RL suelen dividirse en dos clases fundamentales dadas sobre el conocimiento que se tienen sobre el modelo (MDP) a tratar:
  - Conocimiento completo: se pueden aplicar directamente técnicas de Programación Dinámica (DP) ya que se conoce:
    - El conjunto de estados S
    - El conjunto de acciones A
    - La dinámica del entorno (dada por la función de transición T y la función de refuerzo R)
  - Conocimiento incompleto: hay dos enfoques:
    - Métodos basados en el modelo: Se aprende primero el modelo y luego se aplican las técnicas de DP
    - Métodos libres del modelo: Se aplican sin el conocimiento "a priori" del modelo

Programación Dinámica

- Se basa en el mantenimiento de las funciones de valor de cada estado:V(s) y/o Q(s,a)
- Obteniendo los valores óptimos de ambas funciones se obtiene una representación óptima de la política a seguir
- Dichas funciones de valor óptimas cumplen las ecuaciones de Bellman y formulan matemáticamente el principio de optimalidad.

Principio de optimalidad de Bellman

- Una política óptima tienen la propiedad de que cualesquiera que sean el estado inicial y la primera decisión tomada, el resto de decisiones deben construir una política óptima respecto al estado resultante de la primera decisión:

$$V^*(s) = \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\}$$

$$= \max_a \sum_{s'} T(s,a,s')[\,R(s,a) + \gamma V^*(s')]$$

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,| s_t = s, a_t = a\}$$

$$= \sum_{s'} T(s,a,s')[R(s,a) + \gamma \max_{a'} Q^*(s', a')$$

Para todo s∈S, a∈A y s'∈S

- Estas ecuaciones definen las funciones de valor en forma recursiva
- Las relaciones entre ambas funciones de valor viene dada por:

$$V^*(s) = \max_a Q^*(s,a)$$

Algoritmo de DP para encontrar π^*

- La naturaleza recursiva impide calcular las funciones de valor en un solo paso para casi todos los pares estado-acción.
- Se usan entonces algoritmos que convierten la recursividad en procesos iterativos
- Hay varios algoritmos para encontrar la política óptima de esta forma
- Dos de los más comunes son:
  - Iteración del valor (value iteration)
  - Iteración de la política (policy iteration)

Value Iteration Algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$, rewards $R(s)$, discount γ
ϵ, the maximum error allowed in the utility of any state
local variables: U, U', vectors of utilities for states in S, initially zero
δ, the maximum change in the utility of any state in an iteration

repeat
  $U \leftarrow U'$; $\delta \leftarrow 0$
  for each state s **in** S **do**
    $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
    if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \epsilon(1-\gamma)/\gamma$
return U

Value Iteration

- Inicializar $V(s)$ arbitrariamente. Por ejemplo, $V(s) = 0, \forall s \in \mathcal{S}$
- Repetir
  - $\Delta \leftarrow 0$
  - Para todo $s \in S$
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow \text{máx}_a \sum_{s'} \mathcal{T}(s, a, s')[\mathcal{R}(s, a) + \gamma V(s')]$
    - $\Delta \leftarrow max(\Delta, |v - V(s)|)$
- Hasta que $\Delta < \theta$ (un número positivo entero)
- Dar como salida una política π tal que
  $\pi(s) = \arg\text{máx}_a \sum_{s'} \mathcal{T}(s, a, s')[\mathcal{R}(s, a) + \gamma V(s')]$

Policy Iteration Algorithm

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \text{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Policy Iteration

- Inicialización: $V(s) \in \Re$ y $\pi(s) \in \mathcal{A}$ arbitrarios para todo $s \in \mathcal{S}$
- Repetir
  1. Evaluación de la Política
     - Repetir
       - $\Delta \leftarrow 0$
       - Para cada $s \in S$
         - $v \leftarrow V(s)$
         - $V(s) \leftarrow \sum_{s'} \mathcal{T}(s, \pi(s), s')[\mathcal{R}(s, \pi(s)) + \gamma V(s')]$
         - $\Delta \leftarrow max(\Delta, |v - V(s)|)$
     - Hasta que $\Delta < \theta$ (un número positivo entero)
  2. Mejora de la Política
     - *política_estable* $\leftarrow$ *cierto*
     - Para cada $s \in S$
       - $b \leftarrow \pi(s)$
       - $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{T}(s, a, s')[\mathcal{R}(s, a) + \gamma V(s')]$
       - Si $b \neq \pi(s)$, entonces *política_estable* $\leftarrow$ *falso*

  Hasta que *política_estable* = *cierto*

Métodos libres de modelo

- Se asume un desconocimiento completo de la dinámica del entorno (T y R) pero se conoce S y A
- Se trata de aprender la dinámica basados únicamente en la experiencia que se obtiene de interactuar con el entorno, para luego aplicar técnicas de DP
- Esta interacción se puede hacer de varias formas:
  - Métodos Montecarlo: el aprendizaje de la política se basa en la extracción de secuencias completas de comportamiento desde una situación inicial a una final
  - Métodos de diferencias temporales: la experiencia se usa paso a paso, es decir, para cada acción realizada por el agente
  - Métodos de TD(λ): hacen actualizaciones en función de trozos de longitud λ de esas secuencias
- Todos estos métodos se basan en las funciones de valor definidas anteriormente

Montecarlo

- Se basan en la actualización de las funciones de valor mediante secuencias o intentos completos de resolución del problema por parte del agente en una interacción directa con el entorno
- Dado que, en principio, el modelo del problema no es conocido, es necesario aprender la función de valor-acción en lugar de la función de valor-estado, dado que se debe estimar explícitamente el valor de cada acción con el fin de sugerir una política
- El objetivo de los métodos Montecarlo es estimar Q^*
- Uno de los principales algoritmos Montecarlo se denomina Montecarlo ES (*Montecarlo with Exploring Starts*)

Montecarlo ES

- Se utiliza el arreglo *ganancias(s,a)* para almacenar las ganancias que se han ido obteniendo al ejecutar la acción *a* desde el estado *s*
- Se utiliza el valor promedio de este arreglo para actualizar la función *Q*
- Este arreglo es la figura clave de los métodos Montecarlo
- Mantiene los dos pasos principales de Policy Iteration:
  - La actualización del valor de la política, en este caso mendiante la actualización de la función Q
  - La mejora de la política, a través de la actualización de π
- El arranque exploratorio significa que el primer par estado-acción de la secuencia se elige aleatoriamente, lo que permite que todos los pares sean visitados infinitas veces, asegurando así la convergencia del algoritmo

Montecarlo ES

- Inicializar, para todo $s \in \mathcal{S}$, $a \in \mathcal{A}$:
  - $Q(s,a) \leftarrow$valor arbitrario
  - $\pi(s) \leftarrow$ valor arbitrario
  - $ganancias(s,a) \leftarrow$ lista vacía
- Repetir para siempre
  1. Generar un episodio usando arranque exploratorio y π
  2. Para cada par (s,a) que aparece en el episodio
     - $R \leftarrow$ ganancia obtenida tras la primera ocurrencia del par (s,a)
     - Añadir R a $ganancias(s,a)$
     - $Q(s,a) \leftarrow promedio(ganancias(s,a))$
  3. Para cada s en el episodio
     $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Métodos de diferencias temporales (TD)

- Una de las principales aportaciones del RL es el aprendizaje por diferencia temporal (Temporal-Difference learning)
- Son una combinación de DP y MC:
  - Aprenden de la experiencia sin un modelo completo de la dinámica del entorno (MC)
  - Hacen estimaciones en función de otras estimaciones previamente aprendidas (DP)
- Los métodos de TD más conocidos son:
  - TD(0)
  - Q-Learning

TD(0)

- Es el método TD más sencillo
- Se basa en el aprendizaje de la función valor-estado
- Sin embargo, al igual que con MC, dado que no se conoce la dinámica del modelo, sería preferible aprender la función valor-acción

  - Inicializar $V(s)$ arbitrariamente
  - Repetir (para cada episodio)
    - $a \leftarrow$ acción dada por π y s
    - Ejecutar la acción a y observar el refuerzo recibido, r, y el estado siguiente, s'
    - $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    - $s \leftarrow s'$

  Hasta que s sea un estado terminal

Q-Learning

- Uno de los principales algoritmos desarrollados para aprender la función acción-valor (o función Q)
  - Aprende a partir de la experiencia generada durante la exploración del entorno
  - Esta experiencia se representa mediante las denominadas tuplas de experiencia <s,a,s´,r>, donde:
    - s es el estado actual
    - a es la acción ejecutada
    - s' es el estado al que se ha llegado
    - r es el refuerzo recibido
  - A partir de estas tuplas se utiliza la función siguiente para actualizar la tabla Q:

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

α es el parámetro de enfriamiento

Parámetro de enfriamiento

- Tanto en TD(0) como en Q-Learning se incluye un parámetro de enfriamiento α, que define la importancia que se da a las nuevas actualizaciones con respecto a los valores anteriores
- Este parámetro es sólo necesario cuando el dominio es estocástico:
  - Un dominio es estocástico cuando el resultado de ejecutar una acción sobre un mismo estado puede generar transiciones de estados o refuerzos distintos
  - Un dominio es determinístico si el efecto de una misma acción sobre un mismo estado es siempre el mismo
  - En dominios determinísticos α=1 y la actualización Q-Learning para entornos determinísticos es:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

Q-Learning

- Inicializar $Q(s,a)$ arbitrariamente
- Repetir (para cada episodio)
  - Inicializar s
  - Repetir (para cada paso del episodio)
    - Seleccionar una acción a a partir de s usando una política derivada de Q
    - Ejecutar la acción a observando el refuerzo inmediato recibido, r, y el siguiente estado, s'
    - Actualizar la entrada de la tabla, $Q(s,a)$ con la ecuación:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (11.13)$$

    - $s \leftarrow s'$

  Hasta que s sea un estado final

Exploración

- Si se selecciona la acción con la Q más alta y todas se inicializan en 0 entonces el algoritmo se va a "atascar" en el primer estado que se cambie a un valor mayor a 0.
- Para evitarlo, se debe permitir hacer algo de exploración, es decir, no siempre elegir la acción con mayor Q.
- Las técnicas de exploración se encuentran entre dos extremos para seleccionar la acción estando en un estado:
  - El más greedy es seleccionar la acción con Q más alta (cero exploración)
  - El más aleatorio es seleccionar siempre la acción con una probabilidad uniforme (la misma probabilidad para todas) entre todas las acciones.
- Hay muchas técnicas para hacer exploración. Tres opciones comunes son:
  - ε-greedy
  - Softmax
  - Inicio aleatorio

Técnica de exploración 1

- ε-greedy
  - Esta técnica define una probabilidad ε de seleccionar una acción en forma greedy y, por lo tanto, hay una probabilidad 1- ε de seleccionar una acción en forma aleatoria.
  - Desde luego que ε deberá ir tendiendo a 1 a medida que avance el entrenamiento, es decir, hacer más explotación que exploración cuando el sistema ya esté entrenado.
  - Una forma de hacer eso es definir la probabilidad proporcional a t (o inversamente proporcional a 1/t), donde t es el tiempo. De esta forma, a medida que el tiempo avanza la probabilidad de hacer la selección greedy aumenta.
  - Desde luego que ε es un hiperparámetro y esto implica tener todas las dificultades de su selección adecuada.

Técnicas de exploración 2 y 3

- Softmax
  - Esta técnica asigna a cada acción una probabilidad proporcional a su valor Q.
  - Las que tengan mayor valor Q tendrán mayor probabilidad de ser seleccionadas.
  - Esta es igual a la selección de Ruleta que se explicó para los Algoritmos Genéticos.
- Inicio Aleatorio.
  - Una forma muy simple de garantizar que todos los estados son explorados es la que utilizan los algoritmos Montecarlo, es decir, seleccionar al inicio una pareja (estado,acción) aleatoria (una casilla de la tabla) y partir la secuencia de ahí, usando Q, hasta llegar a un estado final.

Acciones inválidas

- Desde luego que hay acciones que son inválidas para algunos estados y ellas nunca se deberían dar. También hay varias formas de solucionarlo y dos de ellas son:
  - Que la acción deje al estado en el mismo estado. Se debe cuidar que no se quede "atascado" por siempre en dicho estado.
  - Que la selección de una acción inválida esté prohibida. En otras palabras, si se selecciona una acción inválida se debe volver a realizar la selección. Finalmente, esta opción es prácticamente la misma que la 1 sólo que, en esta caso, no se toma en cuanta en la secuencia y el valor de esas casillas siempre quedará en 0.

Referencias

- R.S. Sutton and A.G. Barto. Reinforcement Learning: An Introduction. MIT press (1998).
- S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. 3rd ed, Prentice Hall (2010).
- B.S. Araujo (ed). Aprendizaje Automático: conceptos básicos y avanzados. Pearson (2006).