Tema 8 Aprendizaje Automático por Refuerzo

Autor: Ismael Sagredo Olivenza

8.1 Introducción aprendizaje por refuerzo

Hasta ahora hemos visto aprendizaje en base a un entrenamiento previo. Para realizar dicho entrenamiento, necesitamos unos datos de entrenamiento. En función de si sabemos la solución para cada uno de los ejemplos, hablamos de aprendizaje supervisado o no supervisado.

El aprendizaje por refuerzo es un tipo especial de aprendizaje automático **que se realiza de forma online**. Es decir, no se entrena previamente si no que aprende en base a prueba y error.
Hay una cierta supervisión del resultado obtenido ya que un evaluador (bien humano o una simulación) le indica si su comportamiento es correcto, **pero no le indica el resultado correcto**.

Estos algoritmos son muy útiles para aprender comportamientos en entornos poco conocidos y cambiantes. Los agentes que utilizan aprendizaje supervisado o no supervisado, necesitan entrenarse previamente. Muchas veces el re-entrenamiento es costoso y no son ágiles si el entorno cambia (hay que reentrenar) o si no se disponen de datos de entrenamiento.

El aprendizaje por refuerzo por el contrario, si se tiene una buena medida del rendimiento del agente, puede re-entrenar el agente si el entorno cambia, permitiendo adaptarse mejor al entrono.

8.2 Proceso de decisión de Markov

Este tipo de problemas juega con decisiones probabilísticas y decisiones no deterministas. Está principalmente asociado a lo que se donominan **problemas de decisión de Markov o (MDP)** donde:

- El entorno se encuentra en un conjunto de posibles estados S
- Existe una función de transición T(s,a) que es desconocida (si fuera conocida tendríamos una máquina de estados) de forma que dado un estado s del entorno y una acción a ejecutada en ese estado, el entorno cambia a un estado s'.
- Una función de refuerzo R(s,a) (normalmente también desconocida) pero que el entorno informa al agente.

El objetivo es encontrar una **política, preferentemente la óptima**, que permita seleccionar en cada estado s aquellas acciones que maximicen en el futuro el refuerzo.

El refuerzo, es un valor a futuro y no a presente, por ejemplo en un juego puede ser ganar la partida. En un robot que explora marte, encontrar las rocas que quiere analizar. Para que sea un proceso de Markov, la política (que acción debe realizar en cada estado) se basa solamente en el conocimiento del estado actual y no en el histórico de estados por los que ha pasado el agente.

8.3 Q-Learning

Es el algoritmo clásico de aprendizaje por refuerzo más conocido y que se basa en la definición anterior de proceso de Markov.

Sea un conjunto de estados S y un conjunto de acciones A para el agente. Dado un estado del entorno $s \in S$ y una acción $a \in A$ existe una transición $T(s, a) \rightarrow s' \in S$ y el entorno devuelve una recompensa o castigo $R(s, a)$ que devolverá un número entero o real.

El agente cuando toma la decisión inicialmente no sabe la consecuencia de la misma, **hasta que aprende la política que maximice una recompensa a largo plazo**.

La salida del algoritmo Q-Learning es una tabla de políticas Q(s,a) que contiene para cada situación s y acción a, el refuerzo esperado en el tiempo.

La condición de parada del algoritmo es un número de ciclos o comprobar que la matriz Q no varia demasiado entre dos iteraciones.

La matriz Q se va actualizando con cada acción del agente y la recompensa recibida siguiendo la siguiente ecuación (heurística de búsqueda)

Ecuación 1

$$Q_t(s,a) = \alpha[R(s,a) + \gamma \cdot max_{b \in A} Q_{t-1}(s',b)] + (1-\alpha)Q_{t-1}(s,a)$$

Donde:

- $Q_{t-1}(s, a)$: Es el valor que ya tenía la tabla. Este valor es ponderado por el learning rate α
- R(s,a): Es la recompensa directa del estado. Si llegamos a un estado que tiene una recompensa directa, está se usaría en este término de la expresión.
- $max_{b \in A} Q_{t-1}(s', b)$: es la recompensa máxima del estado destino de la transición. Cada acción posible a la que se llegue a s' tendrá previamente almacenado una recompensa. Nos quedaremos con el máximo de esas recompensas.

Esta fórmula permite que el entorno no sea determinista. En estos entornos no deterministas s' puede ser diferente. Por tanto debemos buscar el valor de la s' resultante y quedarnos con el refuerzo más alto que el estado s' obtuvo en el pasado.

- El parámetro γ (0..1) es el descuento que tiene la importancia dle refuerzo en el tiempo.
- El parámetro α (0..1) regula la importancia que se le quiere dar a la experiencia pasada. Si el entorno es totalmente determinista entonces $\alpha = 1$ y la ecuación quedaría como sigue:

Ecuación 2

$$Q_t(s,a) = R(s,a) + \gamma \cdot max_{b \in A}(s', b)$$

8.3.1 Razonamiento de Q-Learning

Se busca una política que maximice el refuerzo esperado en el tiempo. Es decir la suma de todos los refuerzos recibidos. Si conociéramos las reglas del entorno no haría falta inferir la tabla. Pero como no lo sabemos, ésta se calcula haciendo **programación dinámica**.

Hay que explorar el espacio de estados para conseguir encontrar la política optima, por tanto necesitamos repetir el proceso de aprendizaje no siempre tomando las decisiones aprendidas, para adquirir más conocimiento del entorno. La idea es utilizar una probabilidad e **(e-greedy)** de elegir la política o de elegir una acción Aleatoriamente. el valor de e puede ser dinámico.

8.3.2 Algoritmo

```
funcion QLearning (S,A,E, s, e) : Q(s,a)
S: conjunto de estados
A: conjunto de acciones
E: conjunto de recompensas para cada transición R(T(s,a) => s')
s: estado inicial
Q(s,a) tabla de refuerzo esperado inicializada inicialmente a 0
e: probabilidad de seguir la política
-----------------------------------------------------------
Mientras no se cumpla criterio de parada
  Mientras no alcance objetivo o n iteraciones
    a = SeleccionarAccion(e,Q,s)
    s', r = E(s,a);
    Q(s,a) = calcularQ(s,a,s',r) // Ecuación 1 o 2 si es o no determinista
devolver Q.
```

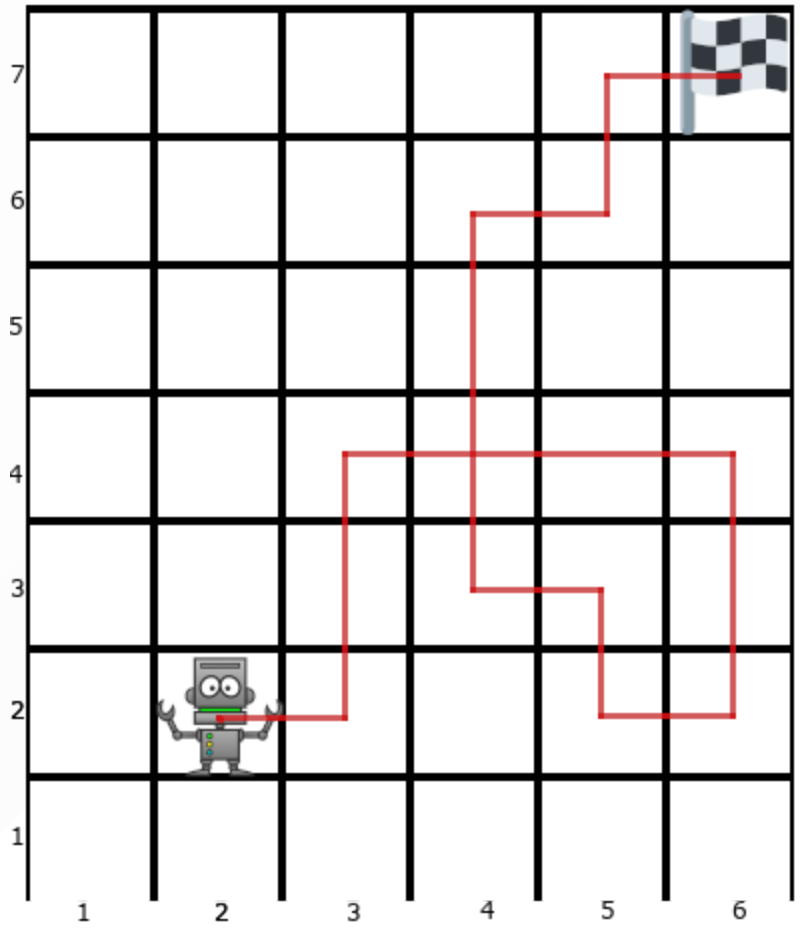
8.3.3 Ejemplo

Supongamos un robot que se mueve en un entorno bidimensional desconocido y que debe aprender el camino para llegar a la salida. Las acciones del robot serán:

- MoveUp, MoveDown, MoveLeft, MoveRight
- Se le proporciona un refuerzo de 1 cuando llega a la meta, 0 en el resto. Si hubiese zonas que hacen caer al robot recibirían un -1. El mapa para el ejemplo es una cuadrícula de 6x7 casillas.

Supongamos que el comportamiento sea determinista $\alpha = 1$ y $\gamma = 0.8$. Inicialmente e = 0 (siempre se moverá aleatoriamente)

Supongamos el siguiente movimiento aleatorio realizado por el robot:



Veamos como va cambiando la tabla Q en cada iteración.

Inicialmente Q = 0

Inicialmente todos los refuerzos serán cero menos el de la casilla Q(5,7) que será:

$$Q_1((5,7),RG) = R((5,7),RG) + 0.8max_b(Q_0((6,7),b)) = 1 + 0.8 \cdot 0 = 1$$

Se seguirá actualizando la tabla Q con la siguiente iteración:

$$Q_2((5,6),UP) = R((5,6),UP) + Max_b(Q((5,7),b)) = 0 + 0.8 * 1 = 0.8$$

En este caso la acción con reward 1 es Q((5,7),RG) que tiene un valor de 1

$$Q_3((4,6),UP) = R((4,6),UP) + Max_b(Q((5,6),b)) = 0 + 0.8 * 0.8 = 0.64$$

Q((5,6),UP) es la max. Asi sucesivamente hasta completar la matriz Q

La matriz Q tendrá una entrada por cada estado y cada acción como la siguiente:

s	UP	DW	LF	RG	r
6,7	0	0	0	0	0
5,7	0	0	0	1	0
5,6	0.8	0	0	0	0
4,6	0.64	0	0	0	0

8.3.4 Y una vez entrenado

El agente se guía por la política Q. Para cada estado habrá una acción con más refuerzo. Esa será la que elija. Si tiene un mismo refuerzo máximo para más de una acción, entonces elegirá una al azar.

El algoritmo puede estar entrenando constantemente modificando la tabla Q en cada iteración para adaptarse a los cambios del entorno.

8.3.5 Limitaciones de Qlearning

- El espacio de estados debe ser finito.
- Si es infinito o muy grande es muy complicado que la búsqueda encuentre una política óptima. Solución => Discretizar el espacio de estados y comprimirlo
- Aún asi, es muy probable que si el problema es complejo QLearning no encuentre una solución medianemente óptima. Soluciones:
  - Aprender la tabla Q con aprendizaje supervisado
  - Hacer una aproximación mixta de aprendizaje supervisado y online.

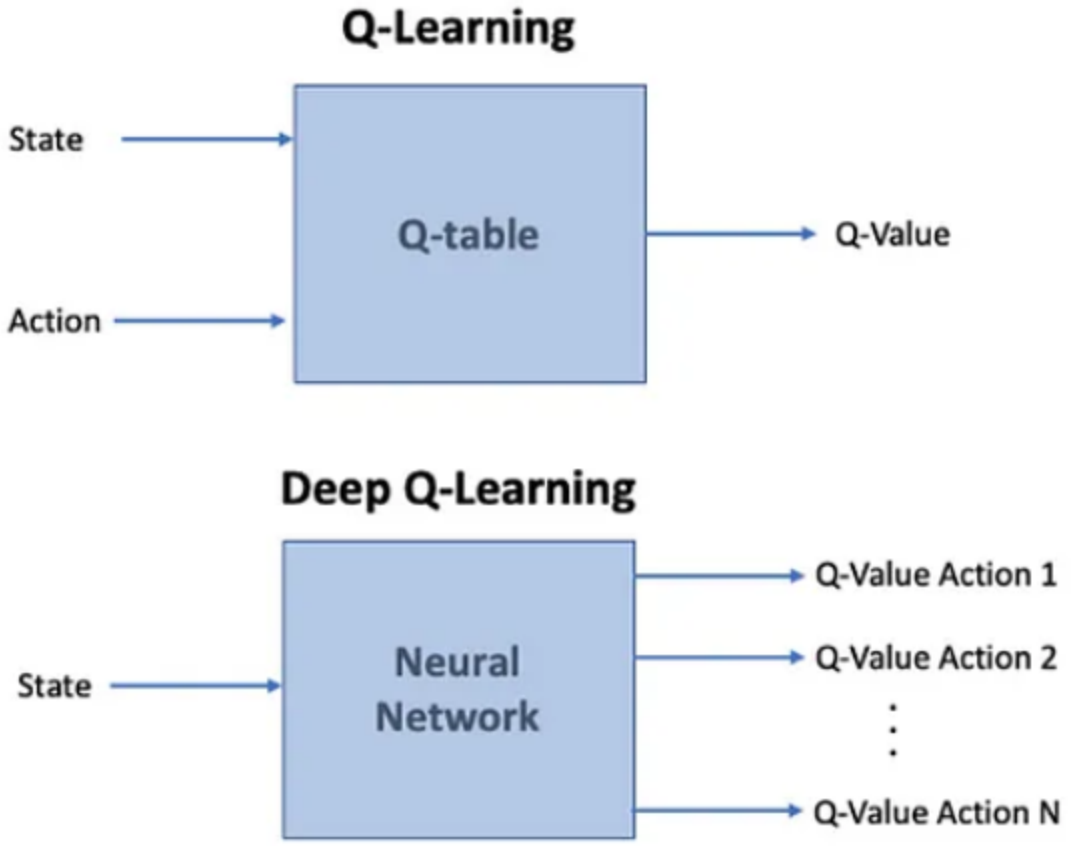
8.4 Aprendizaje por refuerzo profundo

Las limitaciones de Qlearning vienen del espacio de búsqueda y de la capacidad de representación. La tabla Q es demasiado grande para ser calculada cuando el problema crece en tamaño. Para ello existen dos aproximaciones que podemos realizar:

- La primera es aprender la tabla Q usando una red neuronal y entrenándola con ejemplos.
  - Se utiliza una red de neuronas separada para cada acción, cuya entrada es la lista de valores del estado, y cuya salida es valor de refuerzo de la acción

- La segunda es aprender la tabla Q sin un modelo. **Deep Q-Network (DQN)** utiliza esta aproximación y es la base de Alpha Zero y Alpha Start por DeepMind.

DQN sustituye la tabla Q por una red neuronal profunda. La red no predice el valor de Q para un estado dado, si no para todos los posibles estados (Vease figura)



8.4.1 ¿Cómo se entrena la red?

Las versiones Deep de Reinforcement learning son más inestables que las de Q-learning clásico o en las que se aprende la tabla Q con ejemplos. Los valores de la tabla Q oscilan mucho y se requiere una gran cantidad de datos para que converja.

Lo que hacemos para entrenar es usando la ecuación de Bellman, calcular los valores estimados y luego considerar que tenemos un problema de aprendizaje supervisado, entrenando la red con la solución obtenida por la ecuación de Bellman.

```
DQN(A)
A: Agente
-----
Net = inicializarQnetwork()
mientras noconverja
  buffer = moverAgenteFaseQLearning()
  results = Net.predict(buffer)
  error = results - buffer.rewards
  train(buffer,error)
```

El agente almacena en un bucle

8.4.3 Algunos trucos para mejorar la estabilidad

DeepMind cuando publicó su articulo incorporó una serie de optimizaciones que hacían que el entrenamiento funcionase mejor. Vamos a centrarnos en dos de ellas aunque podeis encontrar el articulo completo en: https://www.deepmind.com/publications/human-level-control-through-deep-reinforcement-learning

8.4.3.1 Experience Replay

Evita tener demasiada correlación entre los datos de entrenamiento. En vez de entrenar con los valores predichos por el algoritmo, lo que hacen es meterlos en un buffer y de ahí extraer un conjunto aleatorio de entrenamiento. Esto reduce la dependencia de los datos para entrenar la red. El buffer contiene una colección de tuplas de experiencia (s,a,r,s') anteriores del agente. Las tuplas se agregan gradualmente al buffer a medida que el agente interactúa con el entorno

8.4.3.2 Target Network

Las estimaciones de Q-Learning se hace en base a otras estimaciones y los estados muchas veces son muy similares. Esto hace que cambios pequeños puedan tener efectos colaterales en la red. Para reducirlo, lo que se hace es tener una red idéntica de respaldo y es con esta red con al que se obtiene los valores de Q, es decir, con la que se entrena la red principal. Esta red auxiliar se va actualizando con el entrenamiento de la principal, pero no constantemente si no cada cierto tiempo.

8.5 ML-Agents

Es un entorno de aprendizaje basado en DQL que está disponible en Unity.

https://unity-technologies.github.io/ml-agents/Getting-Started/

Video de ejemplo
https://youtu.be/zPFU30tbyKs?si=ktA1DRP2w6OnbY0U