

Aprendizaje reforzado para la toma de decisiones en entornos complejos

Algoritmos basados en
políticas

Alumno: Samuel David Pérez Brambila

Código: 222966286

Profesora: Karla Ávila Cárdenas

Sección: D01

Fecha de entrega: 10 de Noviembre de 2024

Aprendizaje Máquina

Comprensión del aprendizaje reforzado

Recordemos que en el aprendizaje supervisado, nos basamos en ejemplos de entrenamiento etiquetados, que proporciona un supervisor o una persona experta, y el objetivo es entrenar un modelo que pueda generalizar bien a ejemplos de prueba no vistos, no etiquetados. Por otro lado, en el aprendizaje no supervisado, el objetivo es aprender o capturar la estructura subyacente de un conjunto de datos, como en los métodos de agrupación y la reducción de la dimensionalidad; o aprender a generar nuevos ejemplos sintéticos de entrenamiento con una distribución subyacente similar.

RL es sustancialmente diferente del aprendizaje supervisado y no supervisado, por lo que a menudo se considera RL como la **“tercera categoría del aprendizaje automático”**.



Comprensión del aprendizaje reforzado

El elemento clave que distingue RL de otras subtarefas del aprendizaje automático, como el aprendizaje supervisado y el no supervisado, es que RL se centra en el concepto de aprendizaje por interacción. Esto significa que, en RL, el modelo aprende de las interacciones con el entorno para maximizar una función de recompensa.

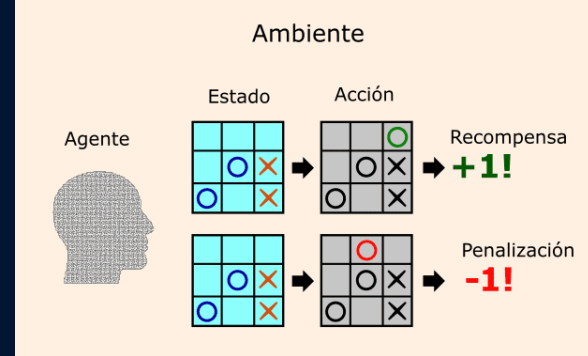
Mientras que la maximización de una función de recompensa está relacionada con el concepto de minimización de la función de pérdida en el aprendizaje supervisado, las etiquetas correctas para el aprendizaje de una serie de acciones no se conocen ni se definen de antemano en la RL, sino que deben aprenderse a través de las interacciones con el entorno para lograr un determinado resultado deseado, como puede ser ganar un juego.



Comprensión del aprendizaje reforzado

Con RL, el modelo (también llamado agente) interactúa con su entorno y, al hacerlo, genera una secuencia de interacciones que se denominan conjuntamente “episodio”. A través de estas interacciones, el agente recoge una serie de recompensas determinadas por el entorno. Estas recompensas pueden ser positivas o negativas, y a veces no se revelan al agente hasta el final del episodio.

Por ejemplo, imaginemos que queremos enseñar a una computadora a jugar al ajedrez y a ganar a jugadores humanos. Las etiquetas (recompensas) de cada movimiento individual de ajedrez realizado por el ordenador no se conocen hasta el final de la partida, porque durante el juego en sí no sabemos si un movimiento concreto dará lugar a la victoria o a la pérdida de esa partida. Solo al final de la partida se determina la retroalimentación. Esa respuesta sería probablemente una recompensa positiva si el ordenador ganara la partida porque el agente hubiera logrado el resultado general deseado; y, al contrario, se daría una recompensa negativa si el ordenador hubiera perdido la partida.



Comprensión del aprendizaje reforzado

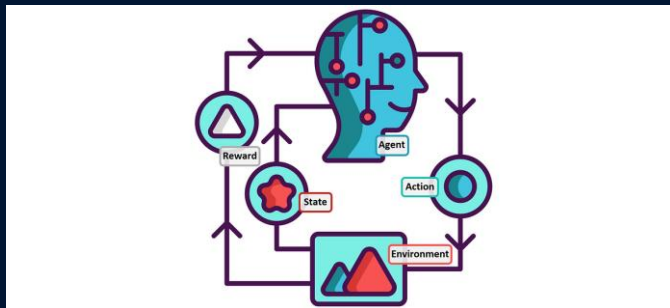
Además, considerando el ejemplo del juego de ajedrez, la entrada es la configuración en curso, por ejemplo, la disposición de las piezas individuales de ajedrez en el tablero. Dado el gran número de entradas posibles (los estados del sistema), es imposible etiquetar cada configuración o estado como positivo o negativo. Por lo tanto, para definir un proceso de aprendizaje, proporcionamos recompensas (o penalizaciones) al final de cada partida, cuando sabemos si hemos alcanzado el resultado deseado, si hemos ganado la partida o no.

Esta es la esencia de RL. En RL, no podemos o no enseñamos a un agente, ordenador o robot cómo hacer las cosas; solo podemos especificar lo que queremos que el agente consiga. Entonces, basándonos en el resultado de un ensayo concreto, podemos determinar las recompensas en función del éxito o el fracaso del agente.



Comprensión del aprendizaje reforzado

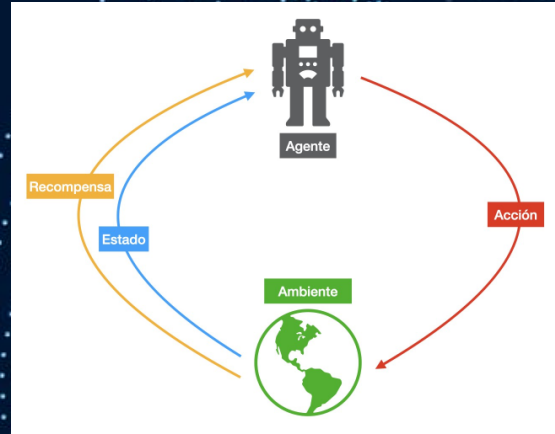
Uno de los aspectos que hace que el entrenamiento de los modelos de RL sea especialmente difícil es que las entradas consiguientes del modelo dependen de las acciones realizadas con anterioridad. Esto puede ocasionar todo tipo de problemas y, por lo general, da lugar a un comportamiento de aprendizaje inestable. Además, esta dependencia de la secuencia en RL crea el llamado **efecto retardado**, que significa que la acción realizada en un instante de tiempo t puede dar lugar a una recompensa futura que aparezca un número arbitrario de pasos después.



Definición de la interfaz agente-entorno de los sistemas de aprendizaje reforzado

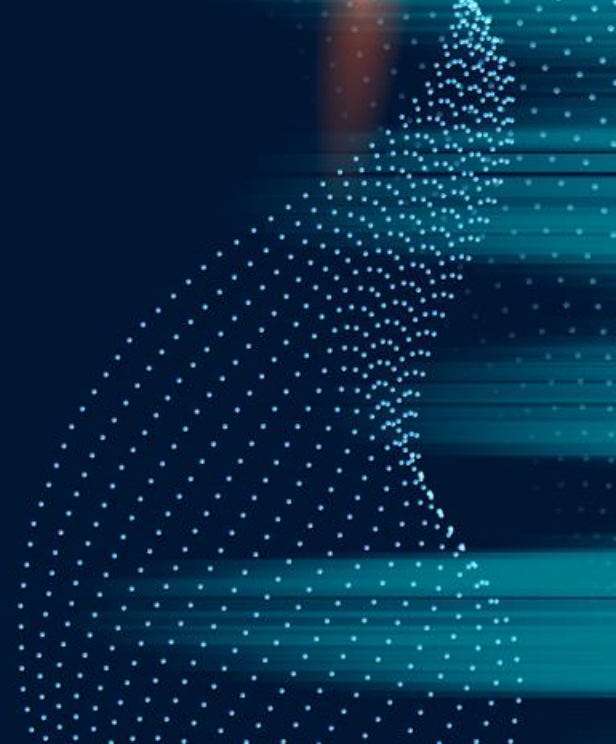
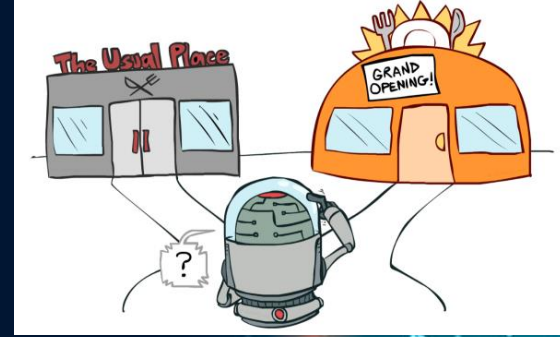
Formalmente, un **agente** se define como una entidad que aprende a tomar decisiones e interactúa con el entorno que le rodea realizando una acción. A cambio, como consecuencia de realizar una acción, el agente recibe observaciones y una señal de recompensa regida por el entorno. El **entorno** es todo lo que queda fuera del agente. El entorno se comunica con el agente y determina la señal de recompensa por la acción del agente, así como sus observaciones.

La **señal de recompensa** es la retroalimentación que recibe el agente al interactuar con el entorno, que suele presentarse en forma de valor escalar y puede ser positiva o negativa. El objetivo de la recompensa es indicar al agente su rendimiento. La frecuencia con la que el agente recibe la recompensa depende de la tarea o el problema en cuestión.



Definición de la interfaz agente-entorno de los sistemas de aprendizaje reforzado

Durante el proceso de aprendizaje, el agente debe probar diferentes acciones (**exploración**) para poder aprender progresivamente qué acciones preferir y realizar más a menudo (**explotación**) con el fin de maximizar la recompensa total y acumulada. Para entender este concepto, consideremos un ejemplo muy sencillo en el que un recién licenciado en informática con orientación a la ingeniería de software se plantea si empezar a trabajar en una empresa (explotación) o realizar un máster o un doctorado para aprender más sobre ciencia de datos y aprendizaje automático (exploración). En general, la explotación dará lugar a la elección de acciones con una mayor recompensa a corto plazo, mientras que la exploración puede dar lugar a una mayor recompensa total a largo plazo.



Fundamentos teóricos de RL

Procesos de decisión de Markov

En general, el tipo de problemas de los que se ocupa RL suelen formularse como procesos de decisión de Markov (Markov Decision Processes, MDP). El enfoque estándar para resolver los problemas MDP es el uso de la programación dinámica, pero RL ofrece algunas ventajas clave sobre la programación dinámica.

Sin embargo, la programación dinámica no es un enfoque viable cuando el tamaño de los estados (es decir, el número de configuraciones posibles) es relativamente grande. En estos casos, RL se considera un enfoque alternativo mucho más eficiente y práctico para resolver los MDP.

Formulación de los procesos de decisión de Markov

Los tipos de problemas que requieren el aprendizaje de un proceso de toma de decisiones interactivo y secuencial, donde la decisión en el instante de tiempo t afecta a las situaciones posteriores, se formalizan matemáticamente como MDP.



Fundamentos teóricos de RL

En el caso de las interacciones agente/entorno en RL, si denotamos el estado inicial del agente como S_0 , las interacciones entre el agente y el entorno dan lugar a una secuencia como la siguiente:

$$\{S_0, A_0, R_1\}, \{S_1, A_1, R_2\}, \{S_2, A_2, R_3\}, \dots$$

Aquí, S_t y A_t representan el estado y la acción realizada en el instante de tiempo t . R_{t+1} denota la recompensa recibida del entorno tras realizar la acción A_t . Nótese que S_t , R_{t+1} , y A_t son variables aleatorias dependientes del tiempo, que toman valores de conjuntos finitos predefinidos denotados por,

$$s \in \hat{S}, r \in \hat{R}, \text{ y } a \in \hat{A}$$

En MDP, estas variables aleatorias dependientes del tiempo, S_t y R_{t+1} , tienen distribuciones de probabilidad que solo dependen de sus valores en el instante de tiempo anterior, $t - 1$. La distribución de probabilidad para $S_{t+1} = s'$ y $R_{t+1} = r$ puede escribirse como una probabilidad condicional sobre el estado precedente (S_t) y la acción tomada (A_t) como sigue:

$$p(s', r | s, a) \stackrel{\text{def}}{=} P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$$



Fundamentos teóricos de RL

Esta distribución de probabilidad define completamente la dinámica del entorno (o modelo del entorno) porque, basándose en esta distribución, se pueden calcular todas las probabilidades de transición del entorno. Por lo tanto, la dinámica del entorno es un criterio central para clasificar los diferentes métodos de RL. Los tipos de métodos de RL que requieren un modelo del entorno o intentan aprender un modelo del entorno (es decir, la dinámica del entorno) se denominan **métodos basados en el modelo**, en contraposición a los métodos sin modelo.

Cuando se conoce la probabilidad $p(s', r | s, a)$, la tarea de aprendizaje puede resolverse con programación dinámica. Pero cuando no se conoce la dinámica del entorno, como ocurre en muchos problemas del mundo real, tendremos que adquirir un gran número de muestras interactuando con el entorno para compensar la dinámica desconocida.

Dos enfoques principales para tratar este problema son los métodos de Monte Carlo (MC) sin modelo y de diferencia temporal (TD). El siguiente gráfico muestra las dos categorías principales y las ramas de cada método:



Fundamentos teóricos de RL

La dinámica del entorno puede considerarse determinista si las acciones particulares para determinados estados se realizan siempre o nunca, es decir, $p(s', r|s, a) \in \{0, 1\}$. De lo contrario, en el caso más general, el entorno tendría un comportamiento estocástico. Para dar sentido a este comportamiento estocástico, consideremos la probabilidad de observar el estado futuro $S_{t+1} = s'$ condicionado por el estado actual $S_t = s$ y la acción realizada $A_t = a$. Esto se denota por:

$$p(s'|s, a) \stackrel{\text{def}}{=} P(S_{t+1} = s' | S_t = s, A_t = a)$$

Se puede calcular como una probabilidad marginal tomando la suma sobre todas las recompensas posibles:

$$p(s'|s, a) \stackrel{\text{def}}{=} \sum_{r \in R} p(s', r|s, a)$$

Esta probabilidad se denomina **probabilidad de transición de estado**. Según la probabilidad de transición de estado, si la dinámica del entorno es determinista, significa que cuando el agente realiza la acción $A_t = a$ en el estado $S_t = s$, la transición al siguiente estado será 100% segura.



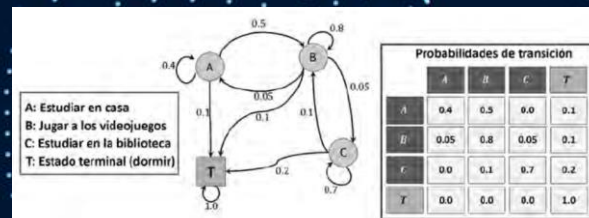
Fundamentos teóricos de RL

Visualización del proceso de Markov

Un proceso de Markov puede representarse como un grafo cíclico dirigido en el que los nodos del grafo representan los diferentes estados del entorno. Las aristas del gráfico (es decir, las conexiones entre los nodos) representan las probabilidades de transición entre los estados.

Por ejemplo, consideremos que un estudiante decide entre tres situaciones diferentes: (A) estudiar para un examen en casa, (B) jugar a videojuegos en casa o (C) estudiar en la biblioteca. Además, hay un estado terminal (T) para ir a dormir. Las decisiones se toman cada hora y, tras tomar una decisión, el estudiante permanecerá en la situación elegida durante esa hora concreta. Entonces, supongamos que cuando se queda en casa (estado A), hay un 50% de probabilidades de que el estudiante cambie esa actividad por jugar a videojuegos. Por otro lado, cuando el estudiante está en el estado B (jugando a videojuegos), existe una probabilidad relativamente alta (80 %) de que siga jugando a videojuegos en las horas siguientes.

La dinámica del comportamiento del alumno se muestra como un proceso de Markov en la siguiente figura, que incluye un grafo cíclico y una tabla de transición.



Fundamentos teóricos de RL

Tareas episódicas o continuas

A medida que el agente interactúa con el entorno, la secuencia de observaciones o estados forma una trayectoria. Hay dos tipos de trayectorias. Si la trayectoria de un agente puede dividirse en subpartes de tal manera que cada una comienza en el momento $t = 0$ y termina en un estado terminal S_T (en $t = T$), se trata de una **tarea episódica**.

Por otro lado, si la trayectoria es infinitamente continua sin un estado terminal, la tarea se denomina **tarea continua**.

En las tareas episódicas, un episodio es una secuencia o trayectoria que un agente realiza desde un estado inicial, S_0 , hasta un estado final, S_T :

$$S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_t, A_t, R_{t+1}, \dots, S_{t-1}, A_{t-1}, R_t, S_t$$

Fundamentos teóricos de RL

Terminología RL: retorno, política y función de valor

Retorno

El llamado retorno en el momento t es la recompensa acumulada obtenida durante toda la duración de un episodio. Recordemos que $R_{t+1} = r$ es la recompensa inmediata obtenida tras realizar una acción, A_t , en el momento t ; las recompensas posteriores son R_{t+2} , R_{t+3} , etc.

El retorno en el momento t puede calcularse a partir de la recompensa inmediata, así como a partir de las posteriores, de la siguiente manera:

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0} \gamma^k R_{t+k+1}$$

Aquí, γ es el factor de descuento en el rango $[0, 1]$. El parámetro γ indica cuánto valen las recompensas futuras en el momento en curso (tiempo t). Nótese que al fijar $\gamma = 0$, implicamos que no nos importan las recompensas futuras. En este caso, el retorno será igual a la recompensa inmediata, ignorará las recompensas posteriores a $t + 1$, y el agente será miope. Por otro lado, si $\gamma = 1$, el retorno será la suma no ponderada de todas las recompensas posteriores.



Fundamentos teóricos de RL

Terminología RL: retorno, política y función de valor

Además, hay que tener en cuenta que la ecuación del retorno puede expresarse de forma más sencilla utilizando la recursividad:

$$G_t = R_{t+1} + \gamma G_{t+1} = r + \gamma G_{t+1}$$

Esto significa que el retorno en el momento t es igual a la recompensa inmediata r más el retorno futuro descontado en el momento $t + 1$.

Política

Una política, normalmente denotada por $\pi(a|s)$, es una función que determina la siguiente acción a realizar, que puede ser bien determinista o estocástica (es decir, la probabilidad de realizar la siguiente acción). Una política estocástica tiene una distribución de probabilidad sobre las acciones que un agente puede tomar en un estado determinado:

$$\pi(a|s) \stackrel{\text{def}}{=} P[A_t = a | S_t = s]$$



Fundamentos teóricos de RL

Terminología RL: retorno, política y función de valor

Durante el proceso de aprendizaje, la política puede cambiar a medida que el agente va adquiriendo más experiencia. Por ejemplo, el agente puede partir de una política aleatoria, en la que la probabilidad de todas las acciones es uniforme; mientras tanto, se espera que el agente aprenda a optimizar su política para alcanzar la política óptima.

La política óptima $\pi_*(a|s)$ es la que produce el mayor rendimiento.

Función de valor

La función de valor, también llamada función de valor del estado, mide la bondad de cada estado, es decir, lo bueno o malo que es estar en un estado concreto. Ahora, basándonos en el retorno G_t , definimos la función de valor del estado s como el retorno esperado (el retorno medio de todos los episodios posibles) tras seguir la política π :

$$v_{\pi}(s) \stackrel{\text{def}}{=} E_{\pi}[G_t | S_t = s] = E_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$$

Fundamentos teóricos de RL

Terminología RL: retorno, política y función de valor

En una implementación real, solemos estimar la función de valor utilizando tablas de búsqueda, para no tener que volver a calcularla varias veces. En una implementación de Python, esto podría ser una lista o una matriz NumPy, cuyos índices se refieren a diferentes estados; o podría ser un diccionario de Python, donde las claves del diccionario asignan los estados a los valores respectivos.

Además, también podemos definir un valor para cada par estado-acción, que se denomina función de valor de la acción y se denota por $q_{\pi}(s, a)$. La función de valor de la acción se refiere al rendimiento esperado G_t cuando el agente se encuentra en el estado $S_t = s$ y realiza la acción $A_t = a$. Extendiendo la definición de la función de valor del estado a los pares estado-acción, obtenemos lo siguiente:

$$q_{\pi}(s, a) \triangleq E_{\pi}[G_t | S_t = s, A_t = a] = E_{\pi} \left[\sum_{k=0}^{\infty} \gamma^{k+1} R_{t+k+1} \middle| S_t = s, A_t = a \right]$$

Fundamentos teóricos de RL

Programación dinámica mediante la ecuación de Bellman

La ecuación de Bellman es uno de los elementos centrales de muchos algoritmos de RL. La ecuación de Bellman simplifica el cálculo de la función de valor, de manera que, en lugar de sumar a lo largo de múltiples pasos de tiempo, utiliza una recursión que es similar a la recursión para calcular el retorno.

Sobre la base de la ecuación recursiva para el retorno total $G_t = r + \gamma G_{t+1}$, podemos reescribir la función de valor como sigue:

$$\begin{aligned} v_{\pi}(s) &\stackrel{\text{def}}{=} E_{\pi}[G_t | S_t = s] \\ &= E_{\pi}[r + \gamma G_{t+1} | S_t = s] \\ &= r + \gamma E_{\pi}[G_{t+1} | S_t = s] \end{aligned}$$

Obsérvese que la recompensa inmediata r se elimina de la expectativa, ya que es una cantidad constante y conocida en el momento t .



Fundamentos teóricos de RL

Programación dinámica mediante la ecuación de Bellman

Del mismo modo, para la función de valor de la acción, podríamos escribir:

$$\begin{aligned} q_{\pi}(s, a) &\stackrel{\text{def}}{=} E_{\pi}[G_t | S_t = s, A_t = a] \\ &= E_{\pi}[r + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= r + \gamma E_{\pi}[G_{t+1} | S_t = s, A_t = a] \end{aligned}$$

Podemos utilizar la dinámica del entorno para calcular la expectativa sumando todas las probabilidades del siguiente estado s' y las correspondientes recompensas r :

$$v_{\pi}(s) = \sum_{a \in \bar{A}} \pi(a|s) \sum_{s' \in \bar{S}, r \in \bar{R}} p(s', r | s, a) [r + \gamma E_{\pi}[G_{t+1} | S_{t+1} = s']]$$

Ahora, podemos ver que la expectativa del retorno $E_{\pi}[G_{t+1} | S_t = s']$ es esencialmente la función de valor del estado $v_{\pi}(s')$. Por lo tanto, podemos escribir $v_{\pi}(s)$ como una función de $v_{\pi}(s')$:




$$v_{\pi}(s) = \sum_{a \in \bar{A}} \pi(a|s) \sum_{s' \in \bar{S}, r \in \bar{R}} p(s', r | s, a) [r + \gamma v_{\pi}(s')]$$

Fundamentos teóricos de RL

Programación dinámica mediante la ecuación de Bellman

A esta fórmula se le llama **ecuación de Bellman**, que relaciona la función de valor para un estado, s , con la función de valor de su estado posterior, s' . Esto simplifica enormemente el cálculo de la función de valor porque elimina el bucle iterativo a lo largo del eje temporal.

Bellman Equation

$V=0.81$ $s1$	$V=0.9$ $s2$	$V=1$ $s3$	 $s4$
$V=0.73$ $s5$	 $s6$	$V=0.9$ $s7$	 $s8$
$V=0.66$ $s9$	$V=0.73$ $s10$	$V=0.81$ $s11$	$V=0.73$ $s12$

Reinforcement Learning

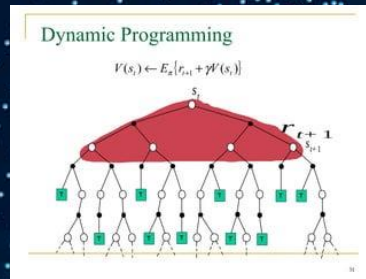
Algoritmos de aprendizaje reforzado



Comenzaremos con la programación dinámica, que asume que la dinámica de las transiciones (o la dinámica del entorno, es decir, $p(s', r | s, a)$) es conocida. Sin embargo, en la mayoría de los problemas de RL, este no es el caso. Para evitar la dinámica desconocida del entorno, se han desarrollado técnicas de RL que aprenden mediante la interacción con el entorno. Estas técnicas incluyen Monte Carlo (MC), el aprendizaje por diferencia temporal (Temporal Difference, TD) y los enfoques cada vez más populares de aprendizaje Q y aprendizaje Q profundo.

Programación dinámica

Debemos destacar que la programación dinámica no es un enfoque práctico para resolver problemas de RL. El problema de utilizar la programación dinámica es que supone un conocimiento completo de la dinámica del entorno, lo que suele ser poco razonable o poco práctico para la mayoría de las aplicaciones del mundo real. Sin embargo, desde un punto de vista educativo, la programación dinámica ayuda a introducir RL de forma sencilla y motiva el uso de algoritmos de RL más avanzados y complicados.



Algoritmos de aprendizaje reforzado

Hay dos objetivos principales que se buscan a través de las tareas descritas en las siguientes subsecciones:

1. Obtener la verdadera función de valor del estado, $v_{\pi}(s)$; esta tarea también se conoce como tarea de pronóstico, y se realiza con la evaluación de la política.
2. Encontrar la función de valor óptima, $v_{*}(s)$, que se realiza mediante la iteración de la política generalizada.

Evaluación de políticas: pronóstico de la función de valor con programación dinámica

Basándonos en la ecuación de Bellman, podemos calcular la función de valor para una política arbitraria con programación dinámica cuando se conoce la dinámica del entorno. Para calcular esta función de valor, podemos adaptar una solución iterativa, en la que partimos de $v^{(0)}(s)$, que se inicializa con valores cero para cada estado. Luego, en cada iteración $i + 1$, actualizamos los valores para cada estado basándonos en la ecuación de Bellman, que, a su vez, se basa en los valores de los estados de una iteración anterior, i:

$$v^{(i+1)}(s) = \sum_a \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a) [r + \gamma v^{(i)}(s')]$$

Algoritmos de aprendizaje reforzado

Mejora de la política mediante la función de valor estimada

Ahora que hemos calculado la función de valor siguiendo la política existente, queremos mejorarla. Esto significa que queremos encontrar una nueva política, π' , que, para cada estado, s , siguiendo π' , produzca un valor mayor o al menos igual que utilizando la política actual, π . En términos matemáticos, podemos expresar este objetivo para la política mejorada, π' , como:

$$v_{\pi'}(s) \geq v_{\pi}(s) \quad \forall s \in \hat{S}$$

Iteración de la política

Utilizando el algoritmo de mejora de la política descrito en la subsección anterior, se puede demostrar que la mejora de la política producirá con seguridad absoluta una política mejor, a menos que la política actual ya sea óptima (lo que significa que $v_{\pi}(s) = v_{\pi'}(s) = v_*(s)$ para cada $s \in \hat{S}$).

Algoritmos de aprendizaje reforzado

Iteración del valor

Hemos visto que repitiendo la evaluación de la política y la mejora de la política, podemos alcanzar la política óptima. Sin embargo, puede ser más eficiente si combinamos las dos tareas de evaluación y mejora de la política en un solo paso. La siguiente ecuación actualiza la función de valor para la iteración $i + 1$ basándose en la acción que maximiza la suma ponderada del valor del siguiente estado y su recompensa inmediata $(r + \gamma v^{(i)}(s'))$:

$$v^{(i+1)}(s) = \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v^{(i)}(s')]$$

En este caso, el valor actualizado para $v^{(i+1)}(s)$ se maximiza eligiendo la mejor acción de entre todas las posibles, mientras que en la evaluación de políticas, el valor actualizado utilizaba la suma ponderada sobre todas las acciones.



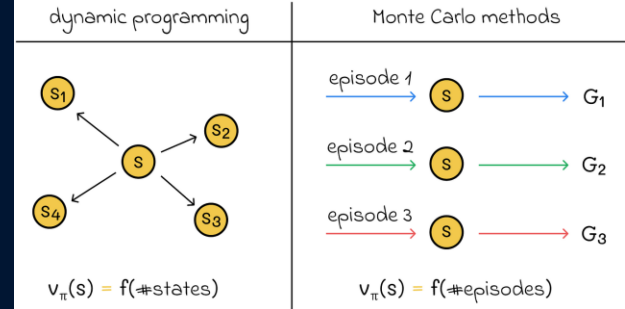
Algoritmos de aprendizaje reforzado

Aprendizaje reforzado con Montecarlo

Con los métodos MC, el proceso de aprendizaje se basa en la llamada experiencia simulada. Para RL basada en MC, definimos una clase de agente que sigue una política probabilística, π , y en base a esta política, nuestro agente realiza una acción en cada paso. El resultado es un episodio simulado.

Estimación de la función de valor del estado mediante MC

Después de generar un conjunto de episodios, para cada estado, s , se considera el conjunto de episodios que pasan todos por el estado s para calcular el valor del estado s . Supongamos que se utiliza una tabla de búsqueda para obtener el valor correspondiente a la función de valor, $V(S_t = s)$. Las actualizaciones de MC para estimar la función de valor se basan en el retorno total obtenido en ese episodio a partir de la primera vez que se consulta el estado s . Este algoritmo se denomina pronóstico de valor Monte Carlo de primera consulta.



Algoritmos de aprendizaje reforzado

Estimación de la función de valor de la acción mediante MC

Cuando se conoce la dinámica del entorno, podemos deducir fácilmente la función de valor de la acción a partir de la función de valor del estado mirando un paso adelante para encontrar la acción que proporciona el máximo valor, como se mostró en la sección de programación dinámica. Sin embargo, esto no es factible si la dinámica del entorno es desconocida. Para resolver este problema, podemos ampliar el algoritmo de estimación del pronóstico del valor de estado del MC de primera consulta. Por ejemplo, podemos calcular el rendimiento estimado para cada par estado-acción utilizando la función de valor de la acción. Para obtener este rendimiento estimado, consideramos las consultas a cada par estado-acción (s, a) , que se refiere a la consulta del estado s y a la realización de la acción a .

Sin embargo, surge un problema, ya que algunas acciones puede que no se seleccionen nunca, lo que da por resultado una exploración insuficiente. Hay algunas formas de resolverlo. El enfoque más sencillo es el llamado inicio exploratorio, que supone que cada par estado-acción tiene una probabilidad distinta de cero al principio del episodio.

Otro enfoque para tratar este problema de falta de exploración se denomina política de vacilación, que se discutirá en la siguiente sección sobre la mejora de la política.



Algoritmos de aprendizaje reforzado

Consecución de la política óptima mediante el control de MC

El control de MC se refiere al procedimiento de optimización para mejorar la política. De forma similar al enfoque de iteración de la política de la sección anterior (Programación dinámica), podemos alternar repetidamente entre la evaluación y la mejora de la política hasta alcanzar la política óptima. Así, partiendo de una política aleatoria, π_0 , el proceso de alternancia entre la evaluación y la mejora de la política puede ilustrarse como sigue:

$$\pi_0 \xrightarrow{\text{Eval.}} q_{\pi_0} \xrightarrow{\text{Improve}} \pi_1 \xrightarrow{\text{Eval.}} q_{\pi_1} \xrightarrow{\text{Improve}} \pi_2 \dots \xrightarrow{\text{Eval.}} q_* \xrightarrow{\text{Improve}} \pi_*$$

Mejora de la política: cálculo de la política codiciosa a partir de la función de valor de la acción

Dada una función de valor de la acción, $q(s, a)$, podemos generar una política codiciosa (determinista):

$$\pi(s) \triangleq \arg \max_a q(s, a)$$

Algoritmos de aprendizaje reforzado

Para evitar el problema de la falta de exploración, y para considerar los pares estado-acción no consultados como se ha comentado anteriormente, podemos dejar que las acciones no óptimas tengan una pequeña probabilidad (ϵ) de ser elegidas. A este tipo de política se la llama política ϵ -codiciosa, según la cual todas las acciones no óptimas en el estado s tienen una probabilidad mínima

$$\frac{\epsilon}{|A(s)|}$$

de ser seleccionadas (en lugar de 0), y la acción óptima tiene una probabilidad de

$$1 - \frac{(|A(s)| - 1) \times \epsilon}{|A(s)|}$$

(en lugar de 1).



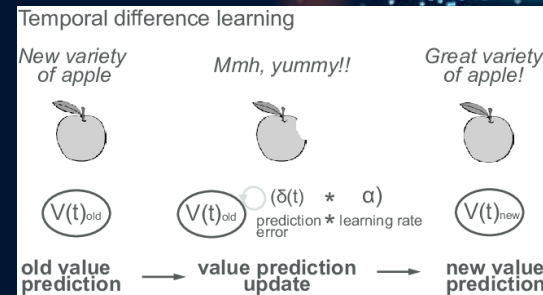
Algoritmos de aprendizaje reforzado

Aprendizaje por diferencia temporal

Al igual que la técnica MC, el aprendizaje mediante TD también se basa en el aprendizaje por experiencia y, por tanto, no requiere ningún conocimiento de la dinámica del entorno ni de las probabilidades de transición. La principal diferencia entre las técnicas TD y MC es que en MC hay que esperar al final del episodio para poder calcular el retorno total.

Sin embargo, en el aprendizaje de TD, podemos aprovechar algunas de las propiedades aprendidas para actualizar los valores estimados antes de llegar al final del episodio. A esta operación se la llama **bootstrapping**.

Al igual que el enfoque de programación dinámica y el aprendizaje basado en MC, consideraremos dos tareas: la estimación de la función de valor (que también se denomina **pronóstico de valor**) y la mejora de la política (que también se denomina **tarea de control**).



Algoritmos de aprendizaje reforzado

Pronóstico TD

Volvemos a consultar el pronóstico del valor por MC. Al final de cada episodio, podemos estimar el rendimiento para cada instante de tiempo. Por lo tanto, podemos actualizar nuestras estimaciones para los estados consultados como sigue:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Aquí, G_t se utiliza como el retorno objetivo para actualizar los valores estimados, y $(G_t - V(S_t))$ es un término de corrección añadido a nuestra estimación actual del valor $V(S_t)$. El valor α es un hiperparámetro que indica la tasa de aprendizaje, que se mantiene constante durante el aprendizaje.

En el aprendizaje TD, sustituimos el retorno real, $G_{t:T}$, por un nuevo retorno objetivo, $G_{t:t+1}$, lo que simplifica considerablemente las actualizaciones de la función de valor. La fórmula de actualización basada en el aprendizaje TD es la siguiente:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_{t:t+1} - V(S_t))$$



Algoritmos de aprendizaje reforzado

En este caso, el retorno objetivo, $G_{t:t+1} \stackrel{\text{def}}{=} R_{t+1} + \gamma V(S_{t+1}) = r + \gamma V(S_{t+1})$, es utilizando la recompensa observada y el valor estimado del siguiente paso inmediato. Nótese la diferencia entre MC y TD. En MC, $G_{t:T}$ no está disponible hasta el final del episodio, por lo que debemos ejecutar tantos pasos como sean necesarios para llegar hasta allí. Por el contrario, en TD solo necesitamos avanzar un paso para conseguir el retorno objetivo. A este también se le conoce como **TD(0)**.

Además, el algoritmo TD(0) puede generalizarse al llamado algoritmo TD de n pasos, que incorpora más pasos futuros; más exactamente, la suma ponderada de n pasos futuros. Si definimos $n = 1$, el procedimiento de TD de n pasos es idéntico al de TD(0), descrito en el párrafo anterior. Sin embargo, si $n \rightarrow \infty$, el algoritmo de TD de n pasos será el mismo que el algoritmo de MC. La regla de actualización para el algoritmo TD de n pasos es la siguiente:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_{t:t+n} - V(S_t))$$

Y $G_{t:t+n}$ se define como:

$$G_{t:t+n} \stackrel{\text{def}}{=} \begin{cases} R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) & \text{if } t+n < T \\ G_{t:T} & \text{otherwise} \end{cases}$$



Algoritmos de aprendizaje reforzado

Control de TD en la política (SARSA)

Para simplificar, solo consideramos el algoritmo de TD de un paso, o TD(0). Sin embargo, el algoritmo de control de TD en la política puede generalizarse fácilmente al TD de n pasos. Empezaremos por ampliar la fórmula de pronóstico con la que definimos la función de valor del estado para describir la función de valor de la acción. Para ello, utilizaremos una tabla de búsqueda, es decir, un array tabular 2D, $Q(S_t, A_t)$, que representa la función de valor del estado para cada par estado-acción. En este caso, tendremos lo siguiente:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Este algoritmo suele denominarse **SARSA**, en referencia a la quintuple que se utiliza en la fórmula de actualización.



Algoritmos de aprendizaje reforzado

Control de TD fuera de la política (Q-Learning)

Al utilizar el anterior algoritmo de control de TD de la política, vimos que la forma de estimar la función de valor de la acción se basa en la política que se utiliza en el episodio simulado. Después de actualizar la función de valor de la acción, se realiza un paso separado para la mejora de la política tomando la acción que tiene el valor más alto.

Un enfoque alternativo (y mejor) es combinar estos dos pasos. En otras palabras, imaginemos que el agente sigue la política π , generando un episodio con la quintuple transición actual. En lugar de actualizar la función de valor de la acción utilizando el valor de la acción de A_{t+1} que toma el agente, podemos encontrar la mejor acción, aunque no sea la elegida por el agente siguiendo la política en curso. (Por eso se considera un algoritmo fuera de la política).

Podemos modificar la regla de actualización para considerar el valor Q máximo variando diferentes acciones en el siguiente estado inmediato. La ecuación modificada para actualizar los valores Q es la siguiente:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

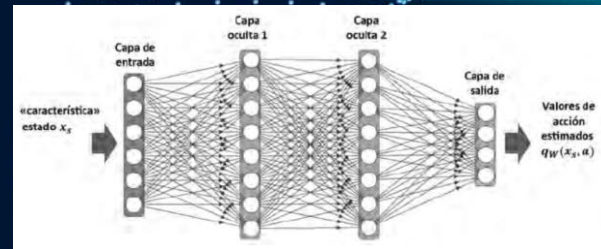


Un vistazo a Q-learning profundo

Hay que tener en cuenta que a veces el número de estados puede ser muy grande, posiblemente casi infinito. Además, puede que estemos tratando con un espacio de estados continuo en lugar de trabajar con estados discretos. Además, es posible que algunos estados no se consulten en absoluto durante el entrenamiento, lo que puede ser problemático a la hora de generalizar el agente para tratar con esos estados no vistos más adelante.

Para abordar estos problemas, en lugar de representar la función de valor en un formato tabular, para la función de valor de la acción, utilizamos un enfoque de aproximación de funciones. En este caso, definimos una función paramétrica, $v_w(x_s)$, que puede aprender a aproximar la verdadera función de valor, es decir, $v_w(x_s) \approx v_\pi(s)$, donde X_s es un conjunto de características de entrada (o estados destacados).

Cuando la función aproximadora, $q_w(x_s, a)$, es una red neuronal profunda (DNN), el modelo resultante se denomina Q-network profundo (Deep Q-Network). Para entrenar un modelo DQN, los pesos se actualizan según el algoritmo Q-learning. En la figura se muestra un ejemplo de modelo DQN, donde los estados se representan como características pasadas a la primera capa:



Un vistazo a Q-learning profundo

Entrenamiento de modelos DQN según el algoritmo Q-learning

Memoria de repetición

Utilizando el método tabular anterior para Q-learning, podíamos actualizar los valores de pares estado-acción específicos sin afectar a los valores de los demás. Sin embargo, ahora que aproximamos $q(s, a)$ con un modelo de NN, la actualización de los pesos de un par estado-acción probablemente afectará también a la salida de otros estados. Cuando se entrenan NN utilizando el descenso del gradiente estocástico para una tarea supervisada (por ejemplo, una tarea de clasificación), utilizamos múltiples ciclos (epochs) para iterar a través de los datos de entrenamiento varias veces hasta que convergen.

Esto no es factible en Q-learning, ya que los episodios cambiarán durante el entrenamiento y, como resultado, algunos estados que fueron consultados en las primeras etapas del entrenamiento tendrán menos probabilidades de ser consultados más tarde. Además, otro problema es que cuando entrenamos una NN, suponemos que los ejemplos de entrenamiento son IID (independientes e idénticamente distribuidos). Sin embargo, las muestras tomadas de un episodio del agente no son IID, ya que forman una secuencia de transiciones.

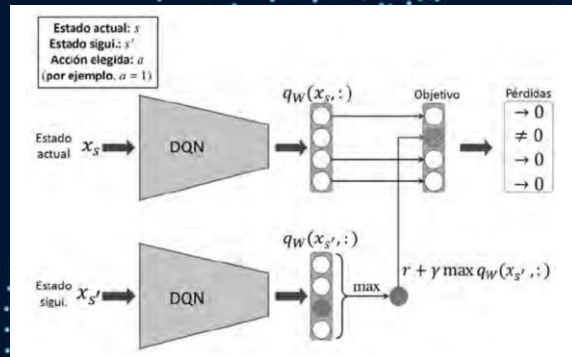


Un vistazo a Q-learning profundo

Para resolver estos problemas, a medida que el agente interactúa con el entorno y genera una quintuple transición $q_w(x_s, a)$, almacenamos un número grande (pero finito) de dichas transiciones en un búfer de memoria, a menudo llamado **memoria de repetición**. Después de cada nueva interacción (es decir, el agente selecciona una acción y la ejecuta en el entorno), la nueva quintuple transición resultante se añade a la memoria.

Determinación de los valores objetivo para el cálculo de la pérdida

Otro cambio necesario respecto al método tabular de Q-learning es cómo adaptar la regla de actualización para el entrenamiento de los parámetros del modelo DQN. Recordemos que una quintuple transición, T , almacenada en el lote de ejemplos, contiene $(x_s, a, r, x_{s'}, \text{done})$. Como se muestra en la figura, realizamos dos pasos hacia delante del modelo DQN. El primero utiliza las características del estado actual (x_s). Luego, el segundo paso hacia delante utiliza las características del siguiente estado ($x_{s'}$). Como resultado, obtendremos los valores de acción estimados, $q_w(x_s, :)$ y $q_w(x_{s'}, :)$, del primer y segundo paso hacia delante, respectivamente. Por lo tanto, según el algoritmo Q-learning, necesitamos actualizar el valor de la acción correspondiente al par estado-acción (x_s, a) con el valor objetivo escalar $r + \gamma \max_{a' \in A} q_w(x_{s'}, a')$. En lugar de formar un valor objetivo escalar, crearemos un vector del valor de la acción objetivo que retiene los valores de acción para otras acciones.

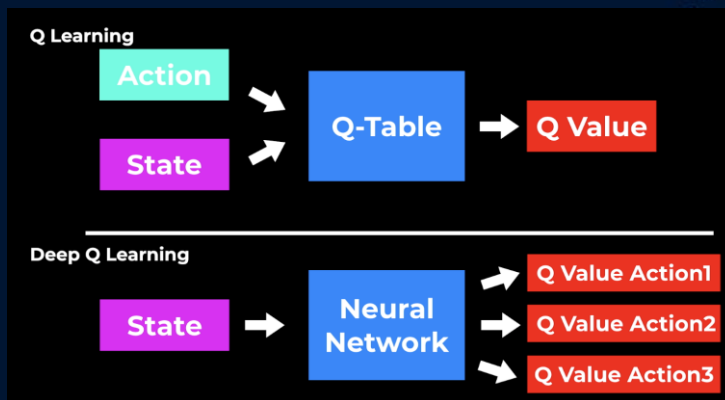


Un vistazo a Q-learning profundo

Lo tratamos como un problema de regresión, utilizando las tres cantidades siguientes:

- Los valores pronosticados actualmente, $q_w(x_{st};)$
- El vector de valores objetivo descrito
- La función de pérdida estándar de error cuadrático medio (MSE)

Como resultado, las pérdidas serán cero para cada acción excepto para a. Finalmente, la pérdida calculada se retropropagará para actualizar los parámetros de la red.



Bibliografía

- Raschka, S., Liu, Y. & Mirjalili, V. (2023). Aprendizaje reforzado para la toma de decisiones en entornos complejos. En *Machine Learning con PyTorch y Scikit-Learn* (pp. 723-768). Alfaomega. <https://udgmulti.bibliotecasdigitales.com/read/9786075760599/index>

