Aprendizaje por refuerzo

Unidad II

Lic. Alondra Vanianinetl Berzunza Rodríguez

Unidad II - Aprendizaje por refuerzo

2.1 Conceptos básicos y beneficios del aprendizaje por refuerzo.

2.2 Casos de uso y desafíos.

2.3 Tipos de modelos de aprendizaje reforzado.

2.4 Funcionamiento y algoritmos comunes.

2.5 Aprendizaje reforzado con python.

2.1 Conceptos básicos y beneficios del aprendizaje por refuerzo

2.1 Conceptos básicos y beneficios

En aprendizaje por refuerzo (Reinforcement Learning en inglés) no tenemos una “etiqueta de salida”, por lo que no es de tipo supervisado y si bien estos algoritmos aprenden por sí mismos, tampoco son de tipo no supervisado.

2.1 Conceptos básicos y beneficios

Los problemas de ML supervisados y no supervisados son específicos de un caso de negocio en particular, sea de clasificación ó predicción, están muy delimitados.

En el mundo real contamos con múltiples variables que por lo general se interrelacionan y que dependen de otros casos de negocio y dan lugar a escenarios más grandes en donde tomar decisiones.

2.1 Conceptos básicos y beneficios

Una solución sería tener múltiples máquinas de ML supervisadas y que interactúan entre sí ó se puede cambiar el enfoque, y ahí aparece el Reinforcement Learning (RL) como una alternativa, tal vez de las más ambiciosas en las que se intenta integrar el Machine Learning en el mundo real, sobre todo aplicado a robots y maquinaria industrial.

El Reinforcement Learning entonces, intentará hacer aprender a la máquina basándose en un esquema de “premios y castigos” -cómo con el perro de Pablov- en un entorno en donde hay que tomar acciones y que está afectado por múltiples variables que cambian con el tiempo.

2.1 Conceptos básicos y beneficios

El aprendizaje por refuerzo (RL) es una técnica de ML que entrena al software para que tome decisiones y logre los mejores resultados.

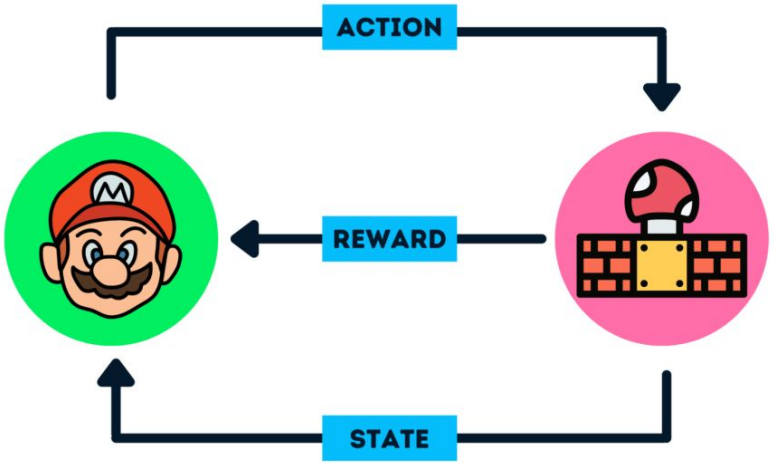
Imita el proceso de aprendizaje por ensayo y error que los humanos utilizan para lograr sus objetivos.

Las acciones de software que trabajan para alcanzar su objetivo se refuerzan, mientras que las que se apartan del objetivo se ignoran.

2.1 Conceptos básicos y beneficios

Los algoritmos de RL utilizan un paradigma de recompensa y castigo al procesar los datos. Aprenden de los comentarios de cada acción y descubren por sí mismos las mejores rutas de procesamiento para lograr los resultados finales. Los algoritmos también son capaces de funcionar con gratificación aplazada.

La mejor estrategia general puede requerir sacrificios a corto plazo, por lo que el mejor enfoque descubierto puede incluir algunos castigos o dar marcha atrás en el camino. El RL es un potente método que ayuda a los sistemas de inteligencia artificial (IA) a lograr resultados óptimos en entornos invisibles.



2.1 Conceptos básicos y beneficios

En los modelos de Aprendizaje Supervisado (o no supervisado) como redes neuronales, árboles, knn, etc, se intenta “minimizar la función coste”, reducir el error.

En cambio en el RL se intenta “maximizar la recompensa“. Y esto puede ser, a pesar de a veces cometer errores ó de no ser óptimos.

2.1 Conceptos básicos y beneficios

Beneficios.

- Sobresale en entornos complejos.
  - Los algoritmos de RL sin modelo se adaptan rápidamente a entornos que cambian continuamente y encuentran nuevas estrategias para optimizar los resultados.

- Requiere menos interacción humana.
  - El algoritmo aprende por sí mismo, pero permite integrar retroalimentación humana.

- Optimiza de acuerdo con objetivos a largo plazo.
  - Apto para escenarios en los que las acciones tienen consecuencias prolongadas. Es especialmente adecuado para situaciones del mundo real en las que no hay retroalimentación disponible de inmediato para cada paso.

2.1 Conceptos básicos y beneficios

Beneficios.

Ejemplo.

Las decisiones sobre el consumo o el almacenamiento de energía pueden tener consecuencias a largo plazo. El RL se puede utilizar para optimizar la eficiencia energética y los costos a largo plazo.

Con las arquitecturas adecuadas, los agentes de RL también pueden generalizar sus estrategias aprendidas en tareas similares pero no idénticas.

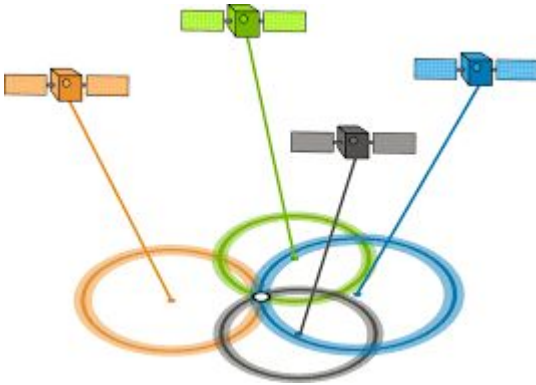
2.2 Casos de uso y desafíos.

2.2 Casos de uso.

- Personalización de marketing.
  - puede personalizar las sugerencias para los usuarios individuales en función de sus interacciones.
- Desafíos de optimización.
  - introduce el aprendizaje a partir de las interacciones para encontrar las mejores soluciones (o las más cercanas a las mejores) a lo largo del tiempo. Ej. gasto en la nube.
- Predicciones financieras.
  - pueden optimizar los rendimientos a largo plazo al considerar los costos de transacción y adaptarse a los cambios del mercado.

2.2 Casos de uso.

- Robots
- Maquinaria industrial
- Mantenimiento predictivo
- Webs personalizadas
- Entrenar sistemas de navegación de coches, drones o aviones
- Videojuegos



2.2 Desafíos.

- Practicidad.
  - Los entornos del mundo real cambian con frecuencia, de manera significativa y con advertencias limitadas.

- Interpretabilidad.
  - Con algoritmos de RL complejos, las razones por las que se tomó una secuencia particular de pasos pueden ser difíciles de determinar.

2.3 Tipos de modelos de aprendizaje reforzado.

2.3 Tipos de modelos de aprendizaje reforzado.

———

RL basado en modelos

- Entornos están bien definidos y no cambian
- Las pruebas en entornos reales son difíciles de realizar.

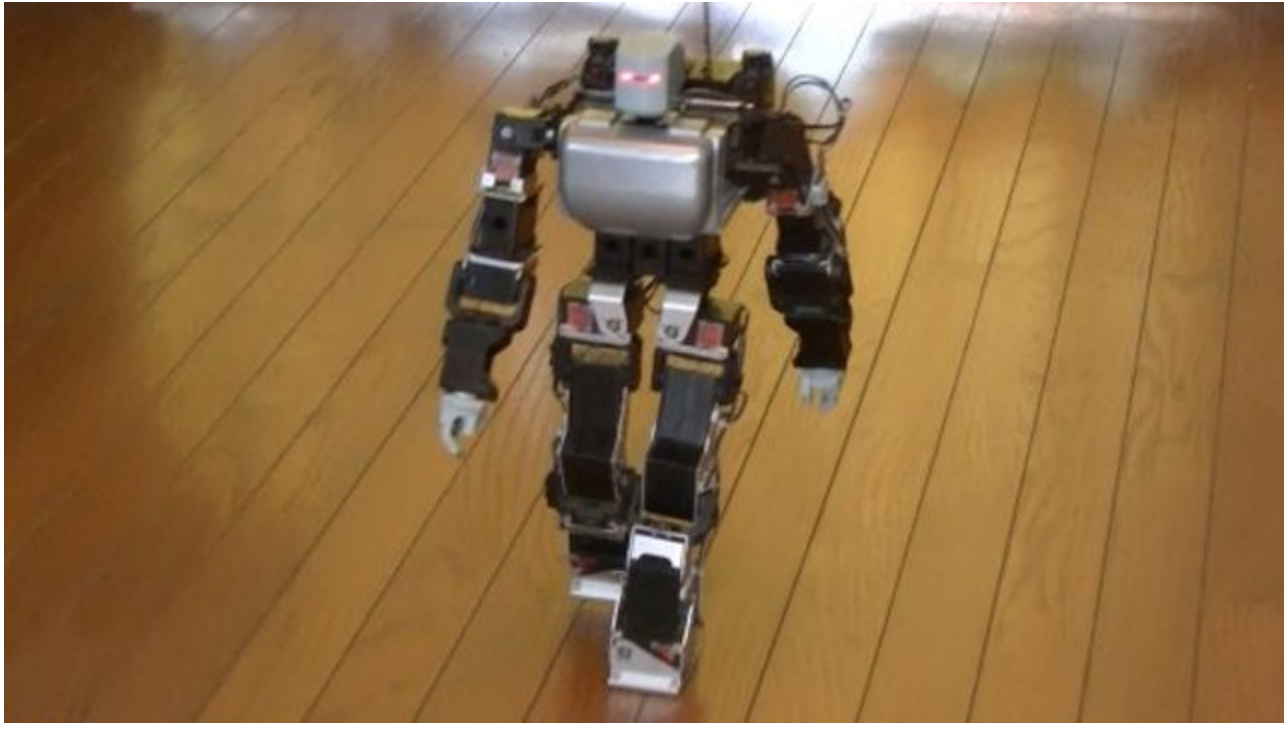
Primero, el agente crea una representación interna (modelo) del entorno. Utiliza este proceso para crear dicho modelo:

1. Toma medidas en el entorno y observa el nuevo estado y el valor de la recompensa.
2. Asocia la transición de acción-estado con el valor de la recompensa.

Una vez que el modelo está completo, el agente simula las secuencias de acción en función de la probabilidad de obtener recompensas acumuladas óptimas. A continuación, asigna valores a las propias secuencias de acción. De este modo, el agente desarrolla diferentes estrategias dentro del entorno para lograr el objetivo final deseado.

2.3 Tipos de modelos de aprendizaje reforzado.

Ejemplo



2.3 Tipos de modelos de aprendizaje reforzado.

RL sin modelo

- El entorno es grande, complejo y no se puede describir fácilmente.
- El entorno es desconocido y cambiante
- Las pruebas basadas en el entorno no presentan desventajas significativas.

El agente no construye un modelo interno del entorno y su dinámica. En su lugar, utiliza un enfoque de prueba y error dentro del entorno. Puntúa y anota los pares de estado-acción (y las secuencias de pares de estado-acción) para desarrollar una política.

2.3 Tipos de modelos de aprendizaje reforzado.

Ejemplo

2.3 Tipos de modelos de aprendizaje reforzado.

Aprendizaje reforzado vs aprendizaje supervisado

Aprendizaje supervisado.

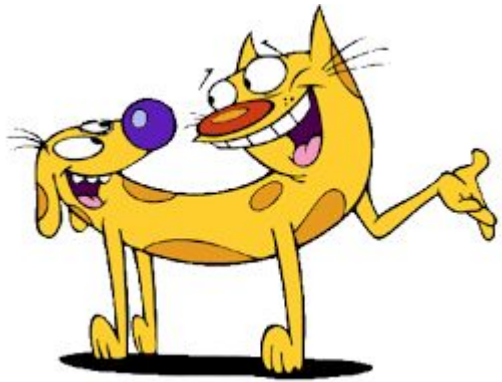
perro

gato

2.3 Tipos de modelos de aprendizaje reforzado.

Aprendizaje reforzado vs aprendizaje supervisado

Aprendizaje supervisado.



2.3 Tipos de modelos de aprendizaje reforzado.

Aprendizaje reforzado vs aprendizaje supervisado

Por el contrario, el RL tiene un objetivo final bien definido en forma de resultado deseado, pero no hay un supervisor que etiquete los datos asociados por adelantado.

Durante el entrenamiento, en lugar de intentar asignar las entradas a las salidas conocidas, asigna las entradas a los posibles resultados. Al recompensar los comportamientos deseados, se da importancia a los mejores resultados.

2.3 Tipos de modelos de aprendizaje reforzado.

Aprendizaje reforzado vs aprendizaje no supervisado

2.3 Tipos de modelos de aprendizaje reforzado.

Aprendizaje reforzado vs aprendizaje no supervisado

El RL tiene un objetivo final predeterminado. Si bien adopta un enfoque exploratorio, las exploraciones se validan y mejoran continuamente para aumentar la probabilidad de alcanzar el objetivo final. Puede enseñarse a sí mismo a alcanzar resultados muy específicos.

En los enfoques clásicos, se intenta “minimizar la función coste”, reducir el error.

En cambio en el RL se intenta “maximizar la recompensa“. Y esto puede ser, a pesar de a veces cometer errores ó de no ser óptimos.

2.4 Funcionamiento y algoritmos comunes.

2.4 Funcionamiento.

El proceso de aprendizaje de los algoritmos de aprendizaje por refuerzo (RL) es similar al aprendizaje por refuerzo animal y humano en el campo de la psicología del comportamiento.

2.4 Funcionamiento.

2.4 Funcionamiento.

Un algoritmo RL imita un proceso de aprendizaje similar. Prueba diferentes actividades para aprender los valores negativos y positivos asociados para lograr el resultado final de la recompensa.

2.4 Funcionamiento.

Componentes.

En el aprendizaje por refuerzo, hay algunos conceptos clave con los que debe familiarizarse:

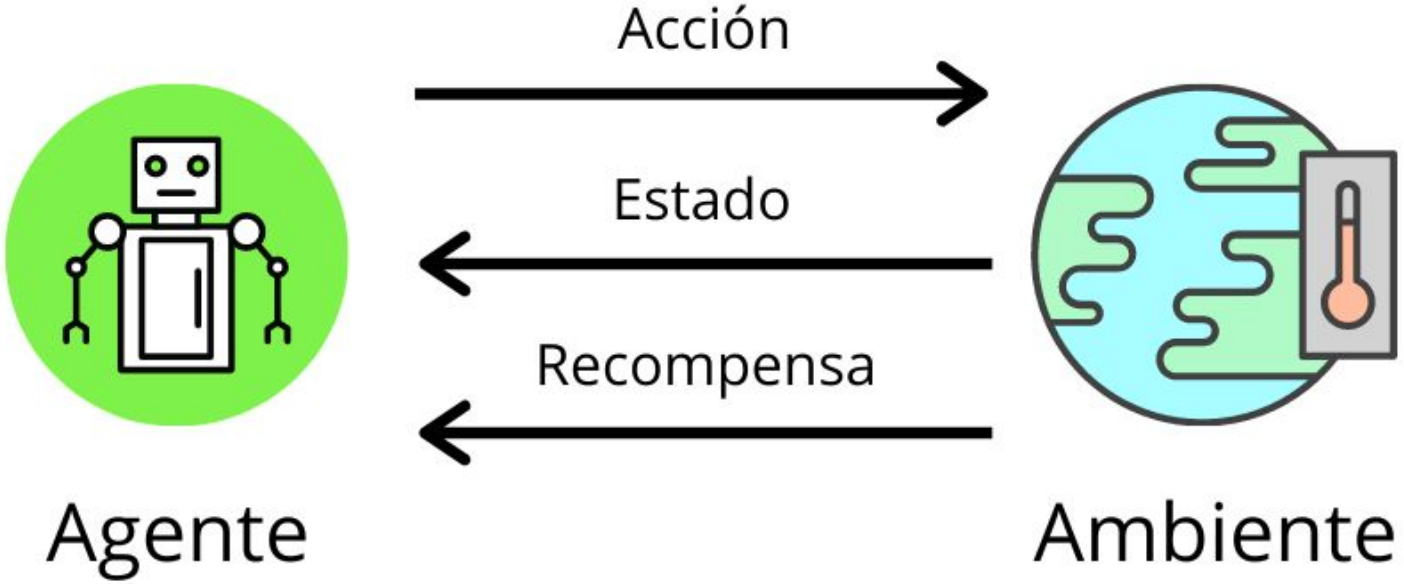
- El agente es el algoritmo ML (o el sistema autónomo)
- El entorno es el espacio de problemas adaptativo con atributos como variables, valores límite, reglas y acciones válidas

2.4 Funcionamiento.

Entre ellos hay una relación que se retroalimenta y cuenta con los siguientes nexos:

- La acción: es un paso que el agente de RL realiza para navegar por el entorno.
- El estado: es el medio ambiente en un momento dado.
- La recompensa: es el valor positivo, negativo o cero (en otras palabras, la recompensa o el castigo) por llevar a cabo una acción.
- La recompensa acumulada: es la suma de todas las recompensas o el valor final.

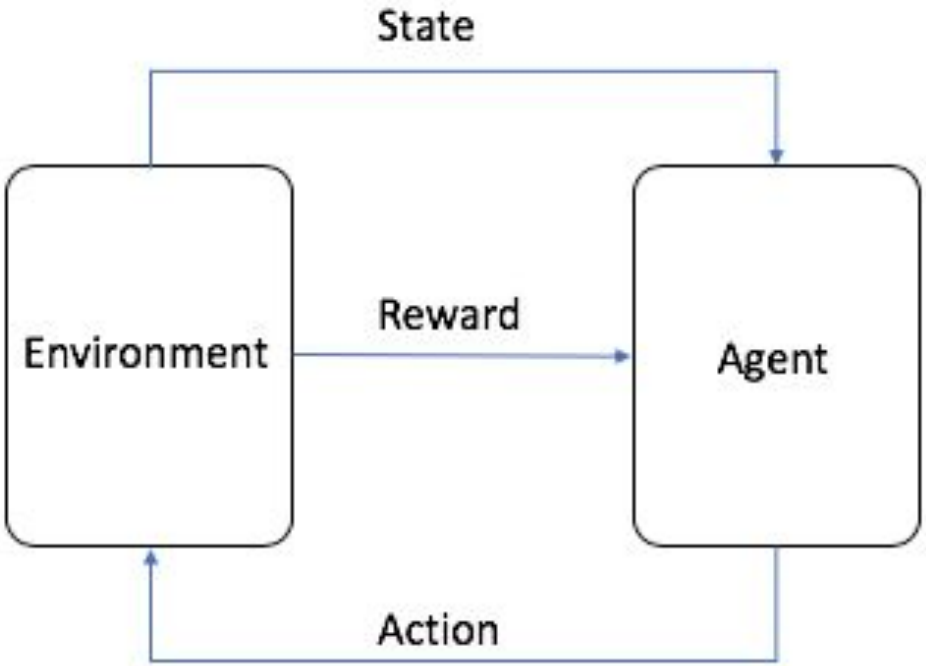
2.4 Funcionamiento.



2.4 Funcionamiento.

El aprendizaje por refuerzo se basa en el proceso de decisión de Markov, un modelo matemático de la toma de decisiones que utiliza intervalos de tiempo discretos. En cada paso, el agente lleva a cabo una nueva acción que da como resultado un nuevo estado del entorno. Del mismo modo, el estado actual se atribuye a la secuencia de acciones anteriores.

2.4 Funcionamiento.



2.4 Funcionamiento.

Mediante prueba y error y su movimiento por el entorno, el agente crea un conjunto de reglas o políticas “condicionales”. Las políticas ayudan a decidir qué acción tomar a continuación para obtener una recompensa acumulada óptima. El agente también debe elegir entre seguir explorando el entorno para obtener nuevas recompensas de estado-acción o seleccionar acciones conocidas con altas recompensas de un estado determinado. Esto se denomina compensación entre exploración y explotación.

2.4 Funcionamiento.

Cómo dijimos antes, el agente deberá tomar decisiones para interactuar con el ambiente, dado un estado. Pero, ¿de qué manera tomar esas decisiones?

2.4 Funcionamiento.

Una recompensa para un humano es algún estímulo que le de placer. Podría ser un aumento de sueldo, chocolate, una buena noticia. Para nuestro modelo de ML la recompensa es sencillamente un Score: un valor numérico.

2.4 Funcionamiento.

Al principio de todo, nuestro agente está “en blanco”, es decir, no sabe nada de nada de lo que tiene que hacer ni de cómo comportarse.

2.4 Funcionamiento.

Supongamos que la acción “A” nos recompensa con 100 puntos. El Agente podría pensar “genial, voy a elegir A nuevamente para obtener 100 puntos” y puede que el algoritmo se estanque en una única acción y nunca logre concretar el objetivo global que queremos lograr.

Dilema de exploración/explotación.

2.4 Funcionamiento.

POLITICAS

2.4 Funcionamiento.

Es probable que el agente “muera” ó pierda la partida las primeras… ¿mil veces?

Es decir, deberemos entrenar miles y miles de veces al agente para que cometa errores y aciertos y pueda crear sus políticas hasta ser un buen Agente.

¿FUERZA BRUTA?

2.4 Funcionamiento.

2.4 Funcionamiento.

¿Lo malo? Tenemos que usar la fuerza bruta para que el agente aprenda.

¿Lo bueno? Contamos con equipos muy potentes que nos posibilitan realizar esta tarea.

Estamos apuntando a un caso de uso mucho más grande y ambicioso que el de “sólo distinguir entre perritos y gatitos”

2.4 Funcionamiento - Ejemplo.

Si tenemos un número N de tragaperras, y cada una nos da una recompensa positiva con probabilidad p, y ninguna recompensa con probabilidad (1-p), ¿podemos crear un agente que maximice las recompensas escogiendo jugar siempre en la tragaperras que más beneficio nos vaya a proporcionar?

Tenemos un bandido con N brazos, y cada brazo tiene una probabilidad distinta de darnos una recompensa positiva. El objetivo es crear un agente que maximice esas recompensas.

2.4 Funcionamiento - Ejemplo.

Para este ejemplo, usaremos un bandido de N=5 brazos (5-armed bandit). Éstas serán las probabilidades de cada brazo de dar una recompensa positiva: [0.1, 0.3, 0.05, 0.55, 0.4]. Como podemos ver, la mejor acción entre estas cinco es tirar del cuarto brazo. Sin embargo, el agente no dispone de esta información. Por lo tanto, deberá probar a tirar de todos los brazos varias veces, e ir aprendiendo cuál de todos es el mejor. Cuando vaya acumulando más información, empezará a tomar mejores decisiones, y a recibir mejores recompensas más frecuentemente.

2.4 Funcionamiento - Ejemplo.

La política ε-voraz (será la política que decidirá qué acciones toma nuestro agente.) consiste en que el agente casi siempre tomará la mejor acción posible dada la información que posee. Sin embargo, de vez en cuando, con una probabilidad de ε, el agente tomará una acción completamente al azar. De esta forma, si tras la primera acción el agente ha obtenido una recompensa positiva, no se quedará atascado escogiendo esa misma acción todo el rato. Con probabilidad ε el agente explorará otras opciones. Este valor ε lo decidiremos nosotros, y será la forma que tengamos de equilibrar el problema de exploración y explotación.

2.4 Funcionamiento - Ejemplo.

La explotación consiste en maximizar las recompensas, por lo que el agente escogerá la mejor de las acciones cada vez.

Por ello, es importante equilibrar la exploración y la explotación: si solo exploramos dos de las cinco acciones posibles, no sabremos si las acciones que nunca hemos probado nos traerán mayores recompensas, por lo que la exploración es necesaria; y sin embargo, si nos pasamos todo el rato explorando todas las opciones una y otra vez, nunca utilizaremos ese conocimiento para poder escoger la mejor acción y conseguir la mayor recompensa posible.

2.4 Algoritmos más comunes.

Tenemos:

- Políticas: Es una tabla (n-dimensional) que le indicará al modelo “como actuar” en cada estado.
- Acciones: las diversas elecciones que puede hacer el agente en cada estado.
- Recompensas: si sumamos o restamos puntaje con la acción tomada.
- Comportamiento “avaro” (greedy) del agente:

  Es decir, si se dejará llevar por grandes recompensas inmediatas, ó irá explorando y valorando las riquezas a largo plazo.

2.4 Algoritmos más comunes.

El objetivo principal al entrenar nuestro modelo a través de las simulaciones será ir “rellenando” la tabla de Políticas de manera que las decisiones que vaya tomando nuestro agente obtengan “la mayor recompensa” a la vez que avanzamos y no nos quedamos estancados, es decir, pudiendo cumplir el objetivo global (ó final) que deseamos alcanzar.

2.4 Algoritmos más comunes - Tipos.

Entre las formas de clasificar los algoritmos de aprendizaje por refuerzo tenemos dos:

- Basados en el valor (Value-Based)
- Basados en la política (Policy-Based).

2.4 Algoritmos más comunes - Valued Based.

Los algoritmos que utilizan únicamente una función de valor o de acción-valor sin implementar una política de forma explícita.

Estos algoritmos no te dicen qué acción debes tomar explícitamente, sino que te indican cuánta recompensa recibirás desde cada estado o estado-acción.

2.4 Algoritmos más comunes.

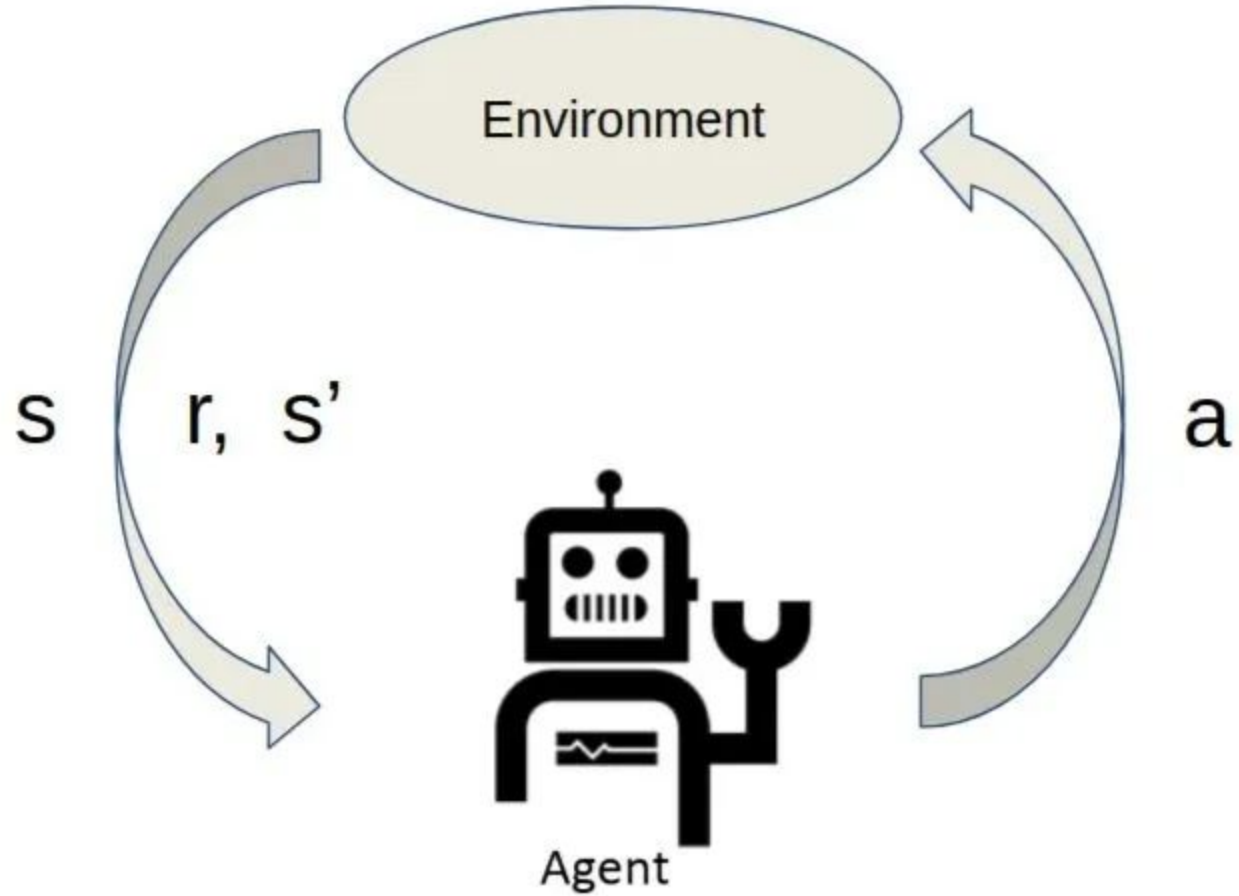
Hemos descrito el problema del bandido multibrazo, y hemos introducido varios conceptos, como el estado, la acción, la recompensa, etc.

Sin embargo, el problema del bandido multibrazo no representa el problema completo del aprendizaje por refuerzo.

2.4 Algoritmos más comunes.

En el problema completo de aprendizaje por refuerzo, el estado cambia cada vez que ejecutamos una acción.

El agente recibe el estado (state) en el que se encuentra el entorno (environment), el cual representaremos con la letra s (state). El agente ejecuta entonces la acción que elija, representada con la letra a (action). Al ejecutar esa acción, el entorno responde proporcionando una recompensa, representada con la letra r (reward), y el entorno se traslada a un nuevo estado, representado con s’ (next state).


Environment
s
r, s'
a
Agent

2.4 Algoritmos más comunes.

Por lo tanto, la acción que el agente escoja no debe sólo depender de la recompensa a que vaya a recibir a corto plazo. Debe elegir las acciones que a largo plazo le traerán la máxima recompensa (o retorno) posible en todo el episodio.

Este ciclo trae una secuencia de estados, acciones y recompensas, desde el primer paso del ciclo hasta el último: s1, a1, r1; s2, a2, r2; …; sT, aT, rT.

2.4 Algoritmos más comunes - Función de Valor.

Para cuantificar la recompensa que obtendrá el agente a largo plazo desde cada estado, introducimos la función de valor V(s). Esta función produce una estimación de la recompensa que obtendrá el agente hasta el final del episodio, empezando desde el estado s.

Si conseguimos estimar este valor correctamente, podremos decidir ejecutar la acción que nos lleve al estado con el valor más alto.

2.4 Algoritmos más comunes - Q - Learning

Para resolver el problema del aprendizaje por refuerzo, el agente debe aprender a escoger la mejor acción posible para cada uno de los estados posibles.

Para ello, el algoritmo Q-Learning intenta aprender cuánta recompensa obtendrá a largo plazo para cada pareja de estados y acciones (s,a).

2.4 Algoritmos más comunes - Q - Learning

A esa función la llamamos la función de acción-valor (action-value function) y este algoritmo la representa como la función Q(s,a), la cual devuelve la recompensa que el agente recibirá al ejecutar la acción a desde el estado s, y asumiendo que seguirá la misma política dictada por la función Q hasta el final del episodio.

Por lo tanto, si desde el estado s, tenemos dos acciones disponibles, a1 y a2, la función Q nos proporcionará los valores-Q (Q-values) de cada una de las acciones.

2.4 Algoritmos más comunes - Q - Learning

A la política la llamaremos “Q” por lo que:

Q(estado, acción)

Nos indicará el valor de la política para un estado y una acción.

Y para saber cómo ir completando la tabla de políticas nos valemos de la **ecuación de Bellman.**

2.4 Algoritmos más comunes - Q - Learning

Hay dos ratios que afectan a la manera en que influye esa recompensa:

- El ratio de aprendizaje, que regula “la velocidad” en la que se aprende.
- La “tasa de descuento” que tendrá en cuenta la recompensa a corto o largo plazo.

2.4 Algoritmos más comunes - DQN.

Q-Learning funciona muy bien cuando el entorno es simple y la función Q(s,a) se puede representar como una tabla o matriz de valores. Pero cuando hay miles de millones de estados diferentes y cientos de acciones distintas, la tabla se vuelve enorme, y no es viable su utilización.

Deep Q-Network o DQN. Este algoritmo combina el algoritmo Q-learning con redes neuronales profundas (Deep Neural Networks).

2.4 Algoritmos más comunes - DQN.

¿Para qué nos sirven las redes neuronales?

2.4 Algoritmos más comunes - DQN.

Este algoritmo usa redes neuronales para aproximar la función Q, evitando así utilizar una tabla para representar la misma.

- La red neuronal principal.
- La red neuronal objetivo.

La red objetivo se congela durante varias iteraciones, y después los parámetros de la red principal se copian a la red objetivo.

2.4 Algoritmos más comunes - DQN.

Ecuación de Bellman

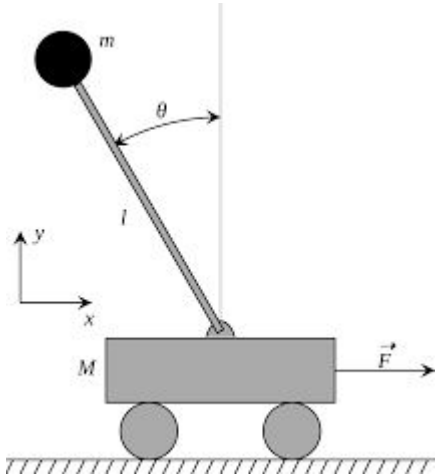
2.4 Algoritmos más comunes - DQN.

Para poder entrenar una red neuronal, necesitamos una función de pérdida o coste (loss or cost function), la cual definimos como el cuadrado de la diferencia entre ambos lados de la ecuación de Bellman.

Por lo tanto, ésta será la función que minimizaremos usando el algoritmo de descenso de gradientes (gradient descent), el cuál se ejecuta automáticamente si usamos una librería de diferenciación automática con redes neuronales, como TensorFlow o PyTorch.

2.4 Algoritmos más comunes - DQN - CartPole

En este entorno, el objetivo es mover el carro hacia la derecha y la izquierda, con el objetivo de equilibrar el palo. Y si el palo se tuerce más de 15 grados respecto al eje vertical, el episodio terminará y volveremos a empezar.



2.4 Algoritmos más comunes.

Los algoritmos de aprendizaje por refuerzo se pueden clasificar en varias familias. La primera de estas familias depende de si el algoritmo aprende cómo funciona el entorno de manera explícita o no.

Un algoritmo basado en el modelo tiene que aprender (o tener acceso a) todas las probabilidades de transición de un estado a otro. Como muchos entornos son estocásticos (probabilísticos) y sus dinámicas desconocidas, el algoritmo debe aprender el modelo detrás de estas transiciones probabilísticas. Una vez aprendidas, utilizará esa información para tomar mejores decisiones.

Por lo tanto, no solo es importante usar las recompensas en la toma de decisiones, sino también el modelo.

2.4 Algoritmos más comunes.

La segunda familia en la que se pueden clasificar los algoritmos son fuera-de-la-política (off policy) y dentro-de-la-política (on-policy).

Los algoritmos off-policy aprenden la función de valor, sin importar qué acciones tome el agente. Es decir, que la política de comportamiento (behavior policy) y la política objetivo (target policy) pueden ser distintas.

2.4 Algoritmos más comunes.

- La política de comportamiento: es la que utiliza el agente para explorar el entorno y recoger datos.

- La política objetivo: es la que el agente intenta aprender y mejorar.

2.4 Algoritmos más comunes.

En los algoritmos dentro-de-la-política, la política de comportamiento y la objetivo deben ser las mismas. Los algoritmos aprenden de los datos que deben recibir tras seguir la misma política que están aprendiendo.

2.4 Algoritmos más comunes.

Q-Learning como DQN son algoritmos libre de modelo y fuera-de-la-política.

2.4 Algoritmos más comunes - Double DQN.

El problema con el algoritmo DQN es que sobreestima las recompensas reales; es decir, los valores-Q que aprende piensan que van a obtener una recompensa mayor de lo que en realidad obtendrá.

Para solucionarlo, los autores del algoritmo Double DQN proponen un sencillo truco:

separar la selección y evaluación de la acción en dos pasos diferentes. En lugar de usar la ecuación de Bellman del algoritmo DQN, este algoritmo la cambia y se convierte en:

2.4 Algoritmos más comunes - Double DQN.

Primero la red neuronal principal θ decide cuál es la mejor acción entre todas las posibles, y luego la red objetivo evalúa esa acción para conocer su valor-Q. Este simple cambio ha demostrado reducir las sobreestimaciones y resultar en mejores políticas.

2.4 Algoritmos más comunes - Basados en política.

Estos algoritmos implementan una política que decide qué acción tomar **en cada momento de forma explícita.**

Definen una política que decide las probabilidades de tomar cada acción en cada estado.

Esta función NO nos dice cuánta recompensa recibirá el agente desde cada estado. No aprende de V(s) ni de Q(s,a).

2.4 Algoritmos más comunes - Basados en política.

Estos algoritmos definen solamente una función de política (Policy Function) $\pi(a|s)$ que estima la probabilidad de tomar cada posible acción desde cada estado.

2.4 Algoritmos más comunes.

Ventajas de algoritmos basados en valor.

1. Se pueden entrenar fuera de la política más fácilmente.

2. Bastante más eficientes respecto a la cantidad de datos necesaria para aprender (sample efficiency).

3. Menor varianza.

2.4 Algoritmos más comunes.

Ventajas de algoritmos basados en política.

1. Pueden representar acciones continuas, por lo que son mejores para entornos estocásticos y entornos de muchas dimensiones (high dimensional).

2. Optimizan directamente la función que nos interesa optimizar (la política).

3. Convergen más rápido.

2.4 Algoritmos más comunes - Basados en política.

¿Cómo optimizamos la política del agente?

Dos grupos:

- Basados en políticas: Se optimizan los parámetros maximizando indirectamente la función objetivo de aproximación local.
- Gradiente de políticas: Se busca directamente la política óptima realizando un ascenso de gradiente basado en el desempeño de la función objetivo.

2.4 Algoritmos más comunes - Política de Montecarlo.

También conocido como el algoritmo de Reinforce, es un método de gradiente de políticas, en el que se requiere aumentar la probabilidad de aquellas acciones que generan los máximos rendimientos. Para ello, se utiliza un rendimiento estimado de un episodio completo para actualizar la política del agente.

2.4 Algoritmos más comunes - Política de Montecarlo.

Este retorno R(τ) se calcula mediante un muestreo de Montecarlo. Recopilamos una trayectoria (muestras de experiencias) y calculamos el retorno descontado, y utilizamos esta puntuación para aumentar o disminuir la probabilidad de cada acción realizada en esta trayectoria. Si el retorno es bueno, todas las acciones se verán “reforzadas” al aumentar la probabilidad de que se realicen.

2.4 Algoritmos más comunes - Política de Montecarlo.

FUNCIONAMIENTO.

1. Comience con una política aleatoria que le indique qué acción tomar en una situación determinada.

2. Pruebe esta política interactuando con el entorno y que vea qué recompensa se obtiene en cada paso del tiempo.

2.4 Algoritmos más comunes - Política de Montecarlo.

FUNCIONAMIENTO.

3. Después de cada episodio, mira la recompensa obtenida en cada paso de tiempo y calcula el retorno para cada paso. El retorno es la suma de todas las recompensas que recibiste después de ese paso de tiempo.

4. Utiliza estos retornos para actualizar la política. La actualización indica la política que favorezca las acciones que obtuvieron un mayor retorno, al aumentar la probabilidad de realizarlas.

2.4 Algoritmos más comunes - Política de Montecarlo.

FUNCIONAMIENTO.

5. Repita los pasos 2 a 4 para varios episodios hasta que la política mejore en la tarea.

6. Utilice esta política mejorada para realizar la tarea prevista en el entorno.

NO NECESITA NINGÚN CONOCIMIENTO SOBRE UNA DINÁMICA DE ENTORNO.