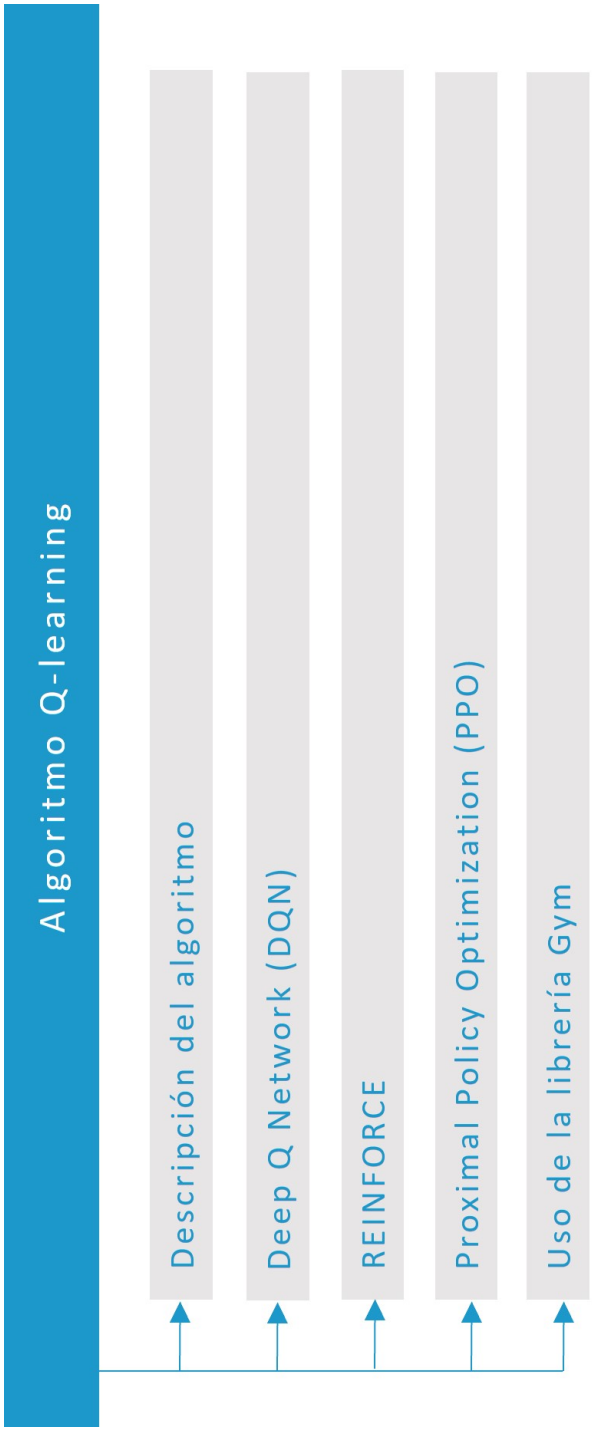
Aprendizaje Automático No Supervisado

Tema 10. Q-learning

Índice

Esquema



10.1. Introducción y objetivos

Ya vimos como el aprendizaje por refuerzo (RL) es un área del aprendizaje automático donde un agente inteligente interactúa con el entorno y crea una estrategia optimizando una recompensa acumulada para alcanzar un objetivo. Difiere del aprendizaje automático supervisado, ya que este último no interactúa con el entorno y se enfoca en encontrar patrones en datos existentes etiquetados para hacer predicciones. Por su parte, el aprendizaje no supervisado busca descubrir patrones ocultos en los datos sin etiquetas, mientras que el aprendizaje por refuerzo implica un proceso de prueba y error guiado por una recompensa.

El **aprendizaje por refuerzo** se puede dividir en los siguientes **pasos principales:**

- **Definición del entorno y el agente:** el entorno es todo lo que interactúa con el agente y puede ser dinámico o estático. El agente es el que toma decisiones y aprende a interactuar con el entorno.
- **Definición del espacio de estados y acciones:** los estados representan las diferentes situaciones en las que puede encontrarse el agente dentro del entorno. Las acciones son las decisiones o movimientos que el agente puede tomar en cada estado.
- **Definición de la función de recompensa:** la función de recompensa proporciona retroalimentación al agente sobre la calidad de sus acciones.
- **Política de aprendizaje:** es la estrategia que el agente sigue para seleccionar acciones en función del estado actual. Puede ser una función determinista o probabilística.
- **Interacción con el entorno:** el agente interactúa con el entorno mediante la ejecución de acciones, observando los estados resultantes y recibiendo recompensas. Este proceso se repite en un ciclo de percepción-acción.

- **Actualización de la política:** basándose en las experiencias obtenidas (estado, acción, recompensa y nuevo estado), el agente actualiza su política para mejorar su toma de decisiones futura.
- **Exploración vs. explotación:** el agente debe equilibrar la exploración de nuevas acciones para descubrir posibles mejores recompensas (exploración) y la utilización de acciones conocidas que ya han proporcionado buenas recompensas (explotación).
- **Evaluación y ajuste:** el rendimiento del agente es evaluado y la política puede ser ajustada si es necesario para mejorar el aprendizaje. Esto puede incluir ajustes en los parámetros del modelo, como la tasa de aprendizaje o el descuento de recompensas futuras.
- **Entrenamiento continuo o episodios múltiples:** el proceso de interacción, actualización y evaluación se repite en múltiples episodios hasta que el agente converja a una política óptima o satisfactoria.
- **Implementación y aplicación:** una vez que el agente ha aprendido una política efectiva, esta puede ser implementada y utilizada en aplicaciones reales para realizar tareas específicas de manera autónoma.

Estos pasos proporcionan un marco general para el aprendizaje por refuerzo, aunque los detalles específicos pueden variar según el problema y el algoritmo utilizado.

En este tema se presentará una de las principales técnicas de aprendizaje por refuerzo, **Q-learning.** El objetivo del Q-learning es aprender una serie de normas que le diga a un agente qué acción tomar bajo qué circunstancias. No requiere un modelo del entorno y puede manejar problemas con transiciones estocásticas y recompensas sin requerir adaptaciones.

Además, veremos **DQN** (Deep Q-Network) y **PPO** (Proximal Policy Optimization). DQN es un algoritmo que combina Q-learning con redes neuronales profundas para

aproximar la función Q, permitiendo que el agente aprenda a tomar decisiones en entornos complejos y de alta dimensionalidad; mientras que PPO es un algoritmo de optimización de políticas que mejora la estabilidad y eficiencia del entrenamiento al restringir la magnitud de las actualizaciones de políticas, logrando un equilibrio entre exploración y explotación.

Los **objetivos** del tema son:

- Comprender los fundamentos del Q-learning, DQN y PPO, sus características principales y cómo interactúan con su entorno para tomar decisiones autónomas.
- Implementar los algoritmos Q-learning, DQN y PPO en un entorno de simulación.
- Explorar aplicaciones prácticas de los algoritmos Q-learning, DQN y PPO.

10.2. Q-learning

Q-learning es un algoritmo de aprendizaje por refuerzo que permite a un agente aprender cómo actúa en un entorno, con el objetivo de maximizar una recompensa acumulada a largo plazo. En este algoritmo se tiene en cuenta una función que estima la utilizad de tomar una acción a en un estado s.

Se actualizan iterativamente los valores de Q haciendo uso de la ecuación de Bellman. El objetivo es aplicar una política que maximice la recompensa acumulada. La fórmula de actualización se muestra en la siguiente ecuación:

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

Donde:

- α es la tasa de aprendizaje, que controla la actualización de los valores Q en cada paso (0 < α ≤ 1).
- γ\gamma es el factor de descuento, que determina la importancia de las futuras recompensas (0 ≤ γ < 1).
- R es la recompensa recibida después de tomar la acción a en un estado s.
- $\max_{a'} Q\left(s',a'\right)$ es el valor máximo de Q para el siguiente estado s', considerando todas las posibles acciones a'.

Al ser un algoritmo de aprendizaje por refuerzo, los conceptos de estado, acción, recompensa son los mismos. Acá encontramos términos nuevos: episodio, valor óptimo de Q, estimación de Q, tabla de q y diferencias temporales.

- Episodio: es el final de una etapa, donde los agentes no pueden emprender nuevas acciones. Se produce cuando el agente ha alcanzado el objetivo o ha fracasado.

- $Q(_{s+1}, a)$: es el valor óptimo de Q esperado después de realizar la acción en un estado concreto.
- $Q(_{st}, _{At})$: es la estimación actual de $Q(_{s+1}, a)$.
- Tabla Q: el agente mantiene la tabla Q de conjuntos de estados acciones.
- Diferencias temporales (TD): se usa para estimar el valor esperado de $Q(_{s+1}, a)$. Hace uso del estado y la acción actual y el estado y acción anterior.

Algoritmo de aprendizaje Q

En la Figura 1 se representa el paso a paso del algoritmo.

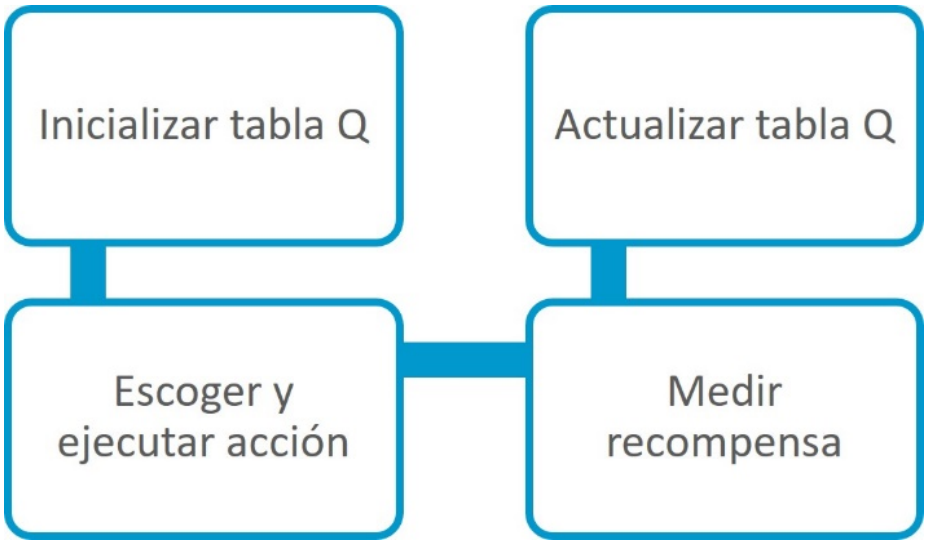


Figura 1. Algoritmo Q-learning. Fuente: elaboración propia.

En la Figura 2 se ilustra cómo un estado y una acción son introducidos y validados contra una tabla Q, la cual contiene diversos valores de estados y acciones con sus correspondientes valores óptimos de Q. El resultado que obtenemos es el valor óptimo de Q para ese estado y esa acción.

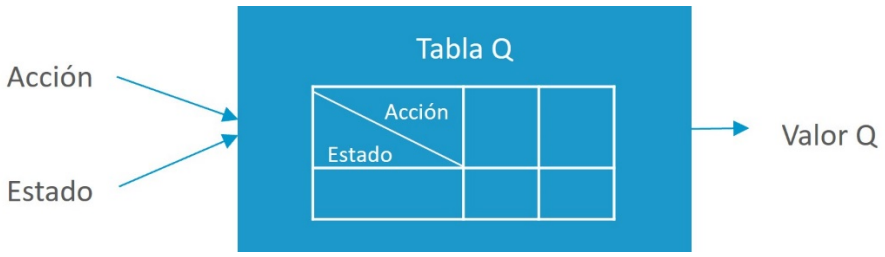


Figura 2. Resultado de aplicar la función Q en Q-learning. Se genera una tabla donde se cruzan los estados con las acciones realizadas y en su intersección se calcula la recompensa obtenida. Fuente: elaboración propia.

Aplicaciones prácticas del Q-learning

- El Q-learning se utiliza en robótica para que los robots aprendan a navegar y realizar tareas específicas de manera autónoma, como evitar obstáculos y recoger objetos (Barla, 2024).
- En el desarrollo de videojuegos, el Q-learning puede ayudar a crear inteligencia artificial para que los personajes puedan aprender y adaptarse al comportamiento de otro jugador (DeepMind, s. f.).
- Se puede aplicar Q-learning en sistemas de gestión de energía para optimizar el consumo y almacenamiento de energía en redes inteligentes y edificios inteligentes (Xiang G, 2018).
- Q-learning puede mejorar sistemas de recomendación, permitiendo que estos sistemas aprendan a ofrecer recomendaciones personalizadas en función del comportamiento del usuario (Jin, et al., 2018).
- En el control de tráfico, el Q-learning puede optimizar la sincronización de semáforos para reducir la congestión y mejorar el flujo de tráfico (Barla, 2024).

Veamos el algoritmo y el paso a paso utilizando el juego de tres en raya

Ejemplo del algoritmo Q-learning con el juego tres en raya

El juego de tres en raya, también conocido como «tic-tac-toe» en inglés o «triqui» en

algunas regiones, es un juego de mesa sencillo que se juega entre dos personas. El objetivo del juego es alinear tres de tus propias marcas (X o O) en una fila horizontal, vertical o diagonal en un tablero de 3x3.

Imaginemos un tablero de tres en raya en un estado inicial vacío y algunos otros estados posibles:

- Estado S0: [, , , , , , , ,].
- Estado S1: [X, , , , , , , ,].
- Estado S2: [X, O, , , , , , ,].
- Estado S3: [X, O, X, , , , , ,].

Acciones

Las acciones son las posiciones en las que un jugador puede colocar su ficha (de 0 a 8).

- **Inicializar la tabla Q:**

Estado	Acción 0	Acción 1	Acción 2	Acción 3	Acción 4	Acción 5	Acción 6	Acción 7	Acción 8
S0	0	0	0	0	0	0	0	0	0
S0	0	0	0	0	0	0	0	0	0
S1	0	0	0	0	0	0	0	0	0
S2	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...

Tabla 1. Ejemplo tres en raya. Fuente: elaboración propia.

- **Selección de acciones:**
  - Las acciones son las posiciones en las que un jugador puede colocar su ficha (0-8).
  - Después de tomar una acción, actualizamos el valor Q usando la fórmula vista anteriormente.

Ejecutar acciones

Supongamos que estamos en el estado S1 ([X, , , , , , , ,]), tomamos la acción 4 (colocar O en el centro) y nos movemos al estado S2 ([X, , , , O, , , ,]). La recompensa R es 0 porque aún no hay ganador.

Estado	Acción 0	Acción 1	Acción 2	Acción 3	Acción 4	Acción 5	Acción 6	Acción 7	Acción 8
S0	0	0	0	0	0	0	0	0	0
S1	0	0	0	0	0	0	0	0	0
S2	0	0	0	0	0	0	0	0	0

Tabla 2. Ejemplo tres en raya. Fuente: elaboración propia.

Actualizamos el valor Q para la acción 4:

Supongamos que α = 0.1, γ = 0.9 y la recompensa R es 0:

$$Q\left(S1,4\right) \leftarrow 0 + 0.1\left[0 + 0.9x \underset{a'}{max}\, Q\left(S2,a'\right) - 0\right]$$

$$Q\left(S1,4\right) \leftarrow 0 + 0.1\left[0 + 0.9x\, max\left(0 - 0\right)\right]$$

$$Q\left(S1,4\right) \leftarrow 0 + 0.1x0 = 0$$

Al actualizar la tabla queda con los mismos valores porque no hay ganador.

Repetir y aprender

Repetimos este proceso para 10000 partidas, donde la recompensa es 1 si un jugador gana. Después de varias actualizaciones podemos tener una tabla como la siguiente:

Estado	Acción 0	Acción 1	Acción 2	Acción 3	Acción 4	Acción 5	Acción 6	Acción 7	Acción 8
S0	0.5	0.2	0.1	0.3	0.4	0.1	0.2	0.1	0.1
S1	0	0	0	0	0.6	0	0	0	0
S2	0.2	0.3	0.1	0.4	0	0.2	0.1	0.1	0.1

Tabla 3. Ejemplo tres en raya. Fuente: elaboración propia.

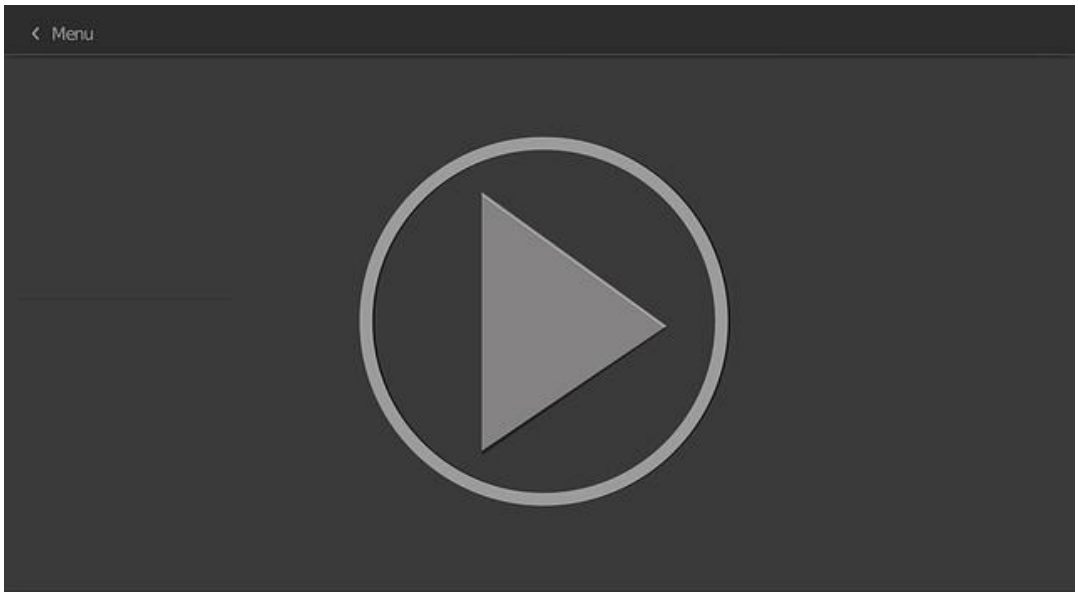
En este punto, el agente ha aprendido que, en el estado S1, la mejor acción es jugar en la posición 4, ya que tiene el valor Q más alto (0.6).

Avances en Q-learning

El Q-learning es un algoritmo de aprendizaje por refuerzo que busca encontrar la política óptima para maximizar las recompensas acumuladas en un entorno dado mediante la actualización iterativa de los valores Q. Una extensión avanzada de Q-learning es el **Deep Q Network (DQN),** que utiliza redes neuronales profundas para aproximar la función de valor Q, permitiendo manejar espacios de estados continuos y de alta dimensión de manera más eficiente.

Por otro lado, el **Proximal Policy Optimization (PPO)** es un método de aprendizaje por refuerzo basado en políticas que optimiza directamente la política mediante la actualización de parámetros en una dirección que maximiza la recompensa esperada mientras mantiene la estabilidad y eficiencia del entrenamiento mediante un control estricto sobre los cambios de la política. En el desarrollo del presente tema, detallaremos ambos algoritmos.

A continuación, veremos el vídeo ***Q-learning.***



Ideas clave

Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=c324a279-3c8e-4933-9b17-b1c5016f6db9

10.3. Deep Q Network (DQN)

En el primer apartado de este tema vimos el algoritmo Q-learning, que funciona muy bien en entornos simples, y la función $Q(s,a)$, que se puede representar como una matriz de valores. Cuando nos enfrentamos a miles o millones de estados, cientos de acciones distintas, no es viable utilizar una tabla de estados. Con el fin de solventar este problema, inventaron el algoritmo Deep Q-Network o DQN (Mnih et al., 2015). Los autores combinan el algoritmo Q-learning con redes neuronales profundas. Con esta combinación, los autores logran aproximar la función Q, evitando utilizar una tabla de estados. Se utilizan dos redes neuronales para estabilizar el proceso de aprendizaje:

- La primera red neuronal es la principal y está representada por los parámetros θ. Esta red se utiliza para estimar los valores-Q del estado s y acción a del estado actual: $Q(s,a;\theta)$. En esta red es donde ocurre el proceso de aprendizaje.

- La segunda red, denominada red neuronal objetivo, parametrizada con θ', tendrá la misma arquitectura que la red principal, pero se usa para aproximar los valores-Q del siguiente estado s' y la siguiente acción a'. Los parámetros no cambian durante varias iteraciones; después de varias iteraciones los parámetros de la red principal se copian a la red objetivo. De esta forma se transmite el aprendizaje de una a otra, haciendo que las estimaciones calculadas por la red objetivo sean más precisas.

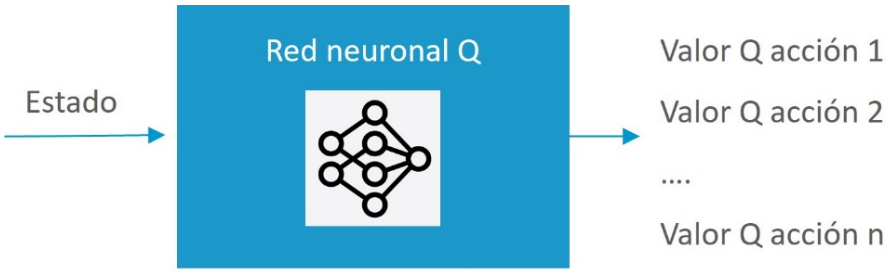


Figura 3. Resultado de aplicar la función Q en DQN. Fuente: elaboración propia.

DQN utiliza la red Q para estimar los valores Q para cada acción en un estado, y se elige la acción con el valor Q más alto. La red tiene como objetivo minimizar la pérdida del error cuadrático medio (MSE) entre los valores Q predichos y los valores Q objetivo, como se muestra en la siguiente ecuación:

$$Loss = \left(Q_{objetivo} - Q_{Predicha}\right)^2$$

Los valores Q objetivo se calculan utilizando la ecuación de Bellman con las recompensas obtenidas y el valor Q máximo del siguiente estado; básicamente lo mismo que se explicó para el algoritmo Q-learning. Sin embargo, en DQN difiere en que el Q objetivo lo estima la red neuronal objetivo, que es una copia de la red Q principal con sus parámetros congelados. Los parámetros congelados se actualizan periódicamente desde la red Q principal a la de destino y el intervalo de actualización es un hiperparámetro, utilizando la siguiente ecuación:

$$Q_{target} = R + \gamma max\left(Q_{next}\right)$$

Al igual que en Q-learning, la política optima π se obtiene de comenzar en un estado dado, seleccionar una acción con el valor más alto posible entre las estimadas por la red Q y continuar con el proceso a medida que el agente se mueve a través del entorno para maximizar la recompensa esperada.

10.4. Algoritmos REINFORCE

E l **algoritmo REINFORCE,** que es una abreviatura de «REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility» (Williams, 1992), es una técnica clásica de aprendizaje por refuerzo basada en el método del gradiente de políticas.

A diferencia del Q-learning, que se enfoca en aprender el valor de las acciones en un estado particular, REINFORCE se centra en aprender directamente una política; es decir, una función que mapea todos los estados-acciones. La política suele estar parametrizada mediante una aproximación de función, que estima la probabilidad de una acción dado un estado. El algoritmo actualiza iterativamente los parámetros de la política calculando estimaciones de gradiente y realizando un ascenso de gradiente para maximizar la recompensa acumulada esperada.

Un agente REINFORCE interactúa con el entorno siguiendo la política actual (normalmente la primera política es aleatoria porque los parámetros de la función de política se inicializan aleatoriamente). Como resultado de aplicar acciones de secuencia, obtiene una secuencia de recompensas y luego calcula el retorno. Esto generalmente se hace sumando las recompensas r desde el paso de tiempo actual hasta el final del episodio como se muestra en la siguiente fórmula:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-t-1} r_{T-1}$$

Donde r_t representa la recompensa en el paso de tiempo t, γ es el factor de descuento para recompensas futuras y T es el último episodio.

El siguiente paso es calcular el gradiente de la política, que representa el gradiente de la recompensa acumulada esperada con respecto a los parámetros de la política; utilizando la siguiente fórmula:

$$\Delta\theta = \alpha\nabla_\theta \log \pi_\theta (a_t | s_t R_t$$

Donde α es la tasa de aprendizaje, $\nabla_\theta \log \pi_\theta (a_t | s_t$ es el gradiente logarítmico de la política y R_t es la recompensa acumulada desde el tiempo t en adelante.

REINFORCE ofrece una perspectiva complementaria a Q-learning, proporcionando una forma efectiva de aprender políticas directamente mediante la optimización de gradientes de políticas.

Mientras que Q-learning se enfoca en calcular el valor de Q, que estima cuán beneficiosa es tomar una acción en un estado específico y seguir la mejor estrategia posible, REINFORCE aprende directamente cuál es la mejor estrategia para maximizar las recompensas acumuladas. REINFORCE ajusta su estrategia en la dirección que aumenta la probabilidad de tomar acciones que han resultado en mayores recompensas, usando un método llamado gradiente de políticas.

Algunos **conceptos claves** para tener en cuenta:

- **Política (π):** una política es una estrategia que el agente sigue para tomar decisiones. En REINFORCE, la política se parametriza como $\pi\theta (a \vee s)$, donde θ son los parámetros de la política, a es la acción y s es el estado.
- **Función de recompensa acumulada (R):** la recompensa acumulada es la suma de las recompensas recibidas a lo largo de un episodio. El objetivo del algoritmo es maximizar esta recompensa acumulada.
- **Gradiente de la política:** REINFORCE utiliza el gradiente de la política para ajustar los parámetros θ. La actualización de los parámetros se realiza en la dirección del gradiente de la recompensa esperada.

A continuación, veremos un ejemplo en Python del algoritmo REINFORCE:

```
import numpy as np
```

```
# Inicialización de parámetros de la política (por ejemplo, una red
neuronal)
theta = np.random.rand(n_parameters)

# Función de política parametrizada
def policy(state, theta):
    # Devuelve la acción basada en la política parametrizada por theta
    # Ejemplo con softmax:
    z = state.dot(theta)
    exp = np.exp(z - np.max(z))
    return exp / np.sum(exp)

# Entrenamiento con REINFORCE
def reinforce(env, theta, alpha, num_episodes):
    for episode in range(num_episodes):
        states, actions, rewards = [], [], []
        state = env.reset()
        done = False

        while not done:
            action = policy(state, theta)
            next_state, reward, done, _ = env.step(action)
            states.append(state)
            actions.append(action)
            rewards.append(reward)
            state = next_state

        # Cálculo de la recompensa acumulada
        G = np.zeros_like(rewards)
        for t in range(len(rewards)):
            G_sum = sum(rewards[t:])
            G[t] = G_sum

        # Actualización de los parámetros de la política
        for t in range(len(states)):
            theta += alpha * G[t] * np.gradient(np.log(policy(states[t],
theta)))
    return theta
```

10.5. Proximal Policy Optimization (PPO)

OpenAI publicó un artículo denominado *Algoritmos de optimización de políticas próximas* (PPO) como una nueva familia de métodos de gradiente de políticas (Schulman et al., 2017). Es conocido por ser robusto y eficiente. PPO se utiliza para entrenar agentes en entornos donde las decisiones se toman en base a políticas; es decir, reglas que determinan las acciones a tomar en cada estado del entorno.

Se destacan **tres puntos principales:**

- PPO alterna entre el muestreo de datos a través de la interacción con el medio ambiente y la optimización de una función objetivo-sustituta mediante un ascenso del gradiente estocástico.
- A diferencia de los métodos de gradiente de políticas estándar que realizan una actualización de gradiente por muestra de datos, PPO introduce una función objetivo novedosa que permite múltiples épocas de actualizaciones de minibatch.
- PPO ofrece alguno de los beneficios de la optimización de políticas de región confiable (TRPO), pero es más simple de implementar, más general y tiene una mejor complejidad de muestra empíricamente.

Conceptos clave en este algoritmo:

- Función de valor (V): mide lo bueno que es estar en un estado en particular.
- Ratio de probabilidad ($r_t\left(\theta\right)$): es el ratio entre la nueva política y la política antigua. Esto es clave en PPO para medir el cambio den la política.

$$r_t\left(\theta\right) = \frac{\pi_\theta\left(a_t \vee s_t\right)}{\pi_{\theta old}\left(a_t \vee s_t\right)}$$

- Clip objetivo: se introduce una función objetivo que limita (clip) los ratios de probabilidad para mantener actualizaciones de política dentro de un rango

razonable. La función objetivo es:

$$L^{CLIP}(\theta) = E_t\left[min\left(r_t(\theta)\,\widehat{A}_t, clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\widehat{A}_t\right)\right]$$

Aquí $\widehat{A}_t$ es la ventaja estimada y ϵ es un hiperparámetro que controla cuánto puede cambiar el ratio de probabilidad.

El algoritmo PPO se entrena de la siguiente manera:

- Recolección de datos, ejecutando la política actual para recolectar datos de interacción del agente con el entorno.
- Estimación de la ventaja $\widehat{A}_t$ para cada acción tomada.
- Actualización de la política utilizando el $L^{CLIP}(\theta)$ para asegurar que las actualizaciones no sean demasiado grandes.
- Este proceso se repite, ajustando la política gradualmente para mejorar el desempeño del agente.

Tabla comparativa entre Q-learning, DQN, REINFORCE y PPO

Algoritmo	Descripción	Ventajas	Desventajas	¿Cuándo usar?
Q-learning	Algoritmo de aprendizaje por refuerzo basado en una tabla Q que estima el valor de las acciones en cada estado.	Simplicidad y facilidad de implementación. Bueno para problemas discretos y de pequeño tamaño.	No escalable a problemas con grandes espacios de estado-acción.	Problemas discretos y pequeños, como juegos simples o navegación básica en robots.
DQN (Deep Q-Network)	Extiende Q-learning utilizando redes neuronales profundas para manejar espacios de estado continuos o muy grandes.	Capaz de manejar grandes espacios de estado. Uso exitoso en juegos complejos como Atari.	Más complejo y requiere mayor poder de cómputo. Puede ser inestable sin técnicas de estabilización.	Juegos complejos, problemas con espacios de estado continuos o grandes, como control robótico avanzado.
REINFORCE	Algoritmo de aprendizaje por refuerzo basado en la optimización de políticas. Actualiza las políticas directamente basándose en el gradiente de la recompensa esperada.	Simplicidad conceptual y flexibilidad. No requiere modelo del entorno.	Alta varianza en las actualizaciones, lo que puede llevar a una convergencia lenta.	Problemas donde es difícil modelar el entorno o donde las políticas directas son más intuitivas.
PPO (Proximal Policy Optimization)	Algoritmo de aprendizaje por refuerzo basado en la optimización de políticas. Introduce una técnica de optimización confiable para mejorar la estabilidad y eficiencia.	Estabilidad y eficiencia mejoradas. Buen rendimiento en una amplia gama de tareas.	Más complejo de implementar que Q-learning o REINFORCE.	Problemas complejos y variados, como simulaciones físicas, control de robots y juegos complejos.

Tabla 4. Tabla comparativa entre Q-learning, DQN, REINFORCE y PPO. Fuente: elaboración propia.

10.6. Implementación de aprendizaje por refuerzo en Python

Para la implementación utilizaremos la librería gym, con la que podemos crear y gestionar entornos de aprendizaje por refuerzo. Gym es una herramienta enfocada a investigadores fácil de usar, que permite probar y desarrollar algoritmos basados en el aprendizaje por refuerzo.

Veamos cómo crear y usar un entorno de gym:

```
¡pip install gym
import gym # Importar la librería gym.

# Crear el entorno
# Aquí se crea una instancia del entorno CartPole-v1.
#Este entorno es parte de la colección estándar de Gym.

env = gym.make('CartPole-v1')

# Reiniciar el entorno para obtener la primera observación
# Antes de comenzar a interactuar con el entorno, es necesario reiniciarlo
para #obtener la primera observación.

observation = env.reset()

#Interacción con el entorno

# Renderizar: Muestra una visualización del entorno (opcional y puede
ralentizar la ejecución).

# Acción Aleatoria: Selecciona una acción aleatoria del espacio de acciones
del entorno

# Ejecutar Acción: Ejecuta la acción seleccionada y devuelve la nueva
observación, la recompensa obtenida, si el episodio ha terminado y alguna
información adicional.

# Reiniciar: Si el episodio ha terminado (done es True), reinicia el
entorno.

for _ in range(1000):
```

```
    # Renderizar el entorno (opcional, puede ralentizar el proceso)
    env.render()

    # Seleccionar una acción aleatoria
    action = env.action_space.sample()

    # Ejecutar la acción en el entorno
    observation, reward, done, info = env.step(action)

    # Verificar si el episodio ha terminado
    if done:
        observation = env.reset()

#Cerrar el entorno

env.close()
```

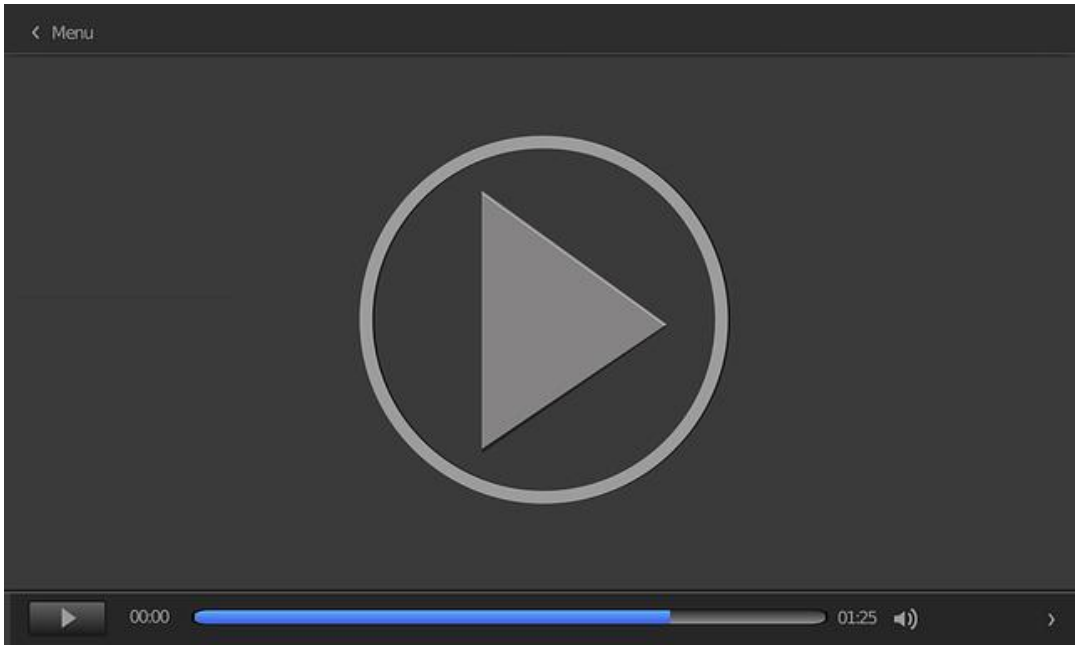
Ahora implementamos el algoritmo Q-learning creando un entorno gym.

Ejemplo del algoritmo REINFORCE en Python:

```
!pip install gym
import gym
import numpy as np

# Crear el entorno
env = gym.make('FrozenLake-v1', is_slippery=False)

# Inicializar la tabla Q
Q = np.zeros([env.observation_space.n, env.action_space.n])

# Parámetros del Q-Learning
alpha = 0.1 # Tasa de aprendizaje
gamma = 0.90 # Factor de descuento
epsilon = 0.1 # Probabilidad de exploración

num_episodes = 1000

for episode in range(num_episodes):
    state, _ = env.reset()
    done = False

    while not done:
        # Asegurarse de que el estado sea un entero
        state = int(state)
```

```
        if np.random.uniform(0, 1) < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q[state])

        next_state, reward, done, _, _ = env.step(action)

        # Asegurarse de que el siguiente estado sea un entero
        next_state = int(next_state)

        Q[state, action] = Q[state, action] + alpha * (reward + gamma *
np.max(Q[next_state]) - Q[state, action])

        state = next_state
# Mostrar la tabla Q final
print("Tabla Q final:")
print(Q)
```

A continuación, veremos el vídeo ***Métodos avanzados e implementación de aprendizaje por refuerzo.***



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=7d93ac29-98d8-4122-9d19-b1c5017ca7d2

10.7. Cuaderno de ejercicios

Ejercicio 1. Aplicación del Q-learning en el juego del tres en raya

Implementa el algoritmo Q-learning para jugar al tres en raya (tic-tac-toe). Inicializa la tabla Q y realiza las primeras tres actualizaciones de la tabla Q usando los siguientes parámetros: $\alpha = 0.1$ y $\gamma = 0.9$. Usa los estados y acciones dados a continuación:

- Estado inicial S0S_0S0: [, , , , , , , ,]\text{[, , , , , , , ,]}[, , , , , , , ,].
- Primera acción: coloca una «X» en la posición 0. Nuevo estado S1S_1S1: [X, , , , , , , ,]\text{[X, , , , , , , ,]}[X, , , , , , , ,].
- Segunda acción: coloca una «O» en la posición 4. Nuevo estado S2S_2S2: [X, , , , O, , , ,]\text{[X, , , , O, , , ,]}[X, , , , O, , , ,].
- Tercera acción: coloca una «X» en la posición 1. Nuevo estado S3S_3S3: [X, X, , , O, , , ,]\text{[X, X, , , O, , , ,]}[X, X, , , O, , , ,].

Solución

- Inicialización de la tabla Q:
  - Q(S0,acción)=0 para todas las acciones.
- Actualización tras la primera acción:
  - $Q(S0,0)=0+0.1\times[0+0.9\times\max(Q(S1,\text{acción})-0)]=0$.
- Actualización tras la segunda acción:
  - $Q(S1,4)=0+0.1\times[0+0.9\times\max(Q(S2,\text{acción})-0)]=0$.
- Actualización tras la tercera acción:
  - $Q(S2,1)=0+0.1\times[0+0.9\times\max(Q(S3,\text{acción})-0)]=0$.

La tabla Q permanece con valores iniciales debido a las recompensas de 0 en cada paso.

Ejercicio 2. Implementación de DQN para el CartPole

Implementa un agente DQN para el entorno `CartPole-v1` usando la librería gym. Describe brevemente cómo se inicializan las redes neuronales principal y objetivo, y cómo se actualizan.

Solución

- Inicialización de las redes:

```
import torch
import torch.nn as nn
import gym

class DQN(nn.Module):
    def __init__(self):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(4, 24)
        self.fc2 = nn.Linear(24, 24)
        self.fc3 = nn.Linear(24, 2)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)

# Redes principales y objetivo
policy_net = DQN()
target_net = DQN()
target_net.load_state_dict(policy_net.state_dict())
target_net.eval()
```

- Actualización de las redes. Cada cierto número de episodios, copiar los pesos de `policy_net` a `target_net`:

```
target_net.load_state_dict(policy_net.state_dict())
```

Ejercicio 3. Gradiente de la política con REINFORCE

Utilizando el algoritmo REINFORCE, calcula el gradiente de la política para un agente en un entorno simple. Supón que el agente recibió las recompensas [1, 0, -1] en un episodio. La política parametrizada está dada por θ = [0.5, 0.5]. Calcula la actualización de θ.

Solución

- Cálculo de las recompensas acumuladas Rt:

$R0=1+0+(-1)=0$

$R1=0+(-1)=-1, R2=-1$

- Gradiente logarítmico de la política (suponiendo una función política simple):

$\nabla\theta \log \pi\theta(at \mid st)=[0.5,0.5]$

- Actualización de θ:

$\theta \leftarrow \theta+\alpha \nabla\theta \log \pi\theta(at \mid st) Rt$

- Con α = 0.1:

$\theta 0 \leftarrow [0.5,0.5]+0.1[0.5,0.5] \times 0=[0.5,0.5] \theta 1 \leftarrow [0.5,0.5]+0.1[0.5,0.5] \times (-1)= [0.45,0.45] \theta 2 \leftarrow [0.45,0.45]+0.1[0.5,0.5] \times (-1)=[0.4,0.4]$

Ejercicio 4. PPO en un entorno simple

Describe cómo se implementa la función objetivo con clipping en PPO. Da un ejemplo numérico donde la probabilidad de la acción bajo la nueva política πθ es 0.6 y bajo la política antigua πθold que es de 0.4, con At=1.2 y ϵ=0.2.

Solución

- Cálculo del ratio de probabilidad rt(θ)r_t(\theta)rt(θ):

$$rt(\theta) = \frac{\pi\theta(at \mid st)}{\pi\theta old(at \mid st)} = \frac{0.6}{0.4} = 1.5$$

- Clipping del ratio:

clip(rt(θ),1−ϵ,1+ϵ)=clip(1.5,0.8,1.2)=1.2

- Función objetivo con clipping:

LCLIP(θ)=min(rt(θ)At,clip(rt(θ)),1−ϵ,1+ϵ)At=min(1.5×1.2,1.2×1.2)=min(1.8,1.44)=1.44

10.8. Referencias bibliográficas

Barla, N. (2024, abril 22). *Self-driving cars with convolutional neural networks (CNN).* Neptune.ai. https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn

DeepMind. (s. f.). *AlphaGo.* https://deepmind.google/technologies/alphago/

Jin, J., Song, C., Li, H., Gai, K., Wang, J. y Zhang, W. (2018). *Real-time bidding with multi-agent reinforcement learning in display advertising.* arXiv. https://arxiv.org/pdf/1802.09756.pdf

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. y Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529. https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. Y Klimov, O. (2017). *Proximal Policy Optimization Algorithms.* OpenAI. https://arxiv.org/pdf/1707.06347

Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning, 8*(3-4), 229–256.

Xiang, G. (2018). *Deep reinforcement learning for time series: playing idealized trading games.* arXiv. https://arxiv.org/ftp/arxiv/papers/1803/1803.03916.pdf

¿Qué es el aprendizaje por refuerzo y cómo trabaja?

Johnny Code. (2023, diciembre 2021). *Deep Q-Learning/Deep Q-Network (DQN) Explained | Python Pytorch Deep Reinforcement Learning* [Vídeo]. YouTube. https://www.youtube.com/watch?v=EUrWGTCGzlA

Explicación del paso a paso, un recorrido por el código y una demostración de cómo funciona Deep Q-learning (DQL). Usan DQL para resolver el muy simple entorno de aprendizaje por refuerzo Gymnasium FrozenLake-v1. Habla de las diferencias entre Q-learning y DQL, la política Epsilon-Greedy, la política Deep Q-Network (DQN), el Target DQN y Experience replay.

Entrenando el juego de Pokemon utilizando aprendizaje por refuerzo

Peter Whidden. (2023, octubre 2023). *Training AI to Play Pokemon with Reinforcement Learning* [Vídeo]. YouTube. https://www.youtube.com/watch?v=DcYLT37ImBY

Una simulación del juego Pokemon utilizando aprendizaje por refuerzo y viendo su comportamiento durante el entrenamiento.

Conceptos clave en el aprendizaje por refuerzo

OpenAI. (s. f.). *Part 1: Key Concepts in RL.* https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

OpenAI muestra una explicación de alto nivel de lo que hacen los algoritmos RL.

Introducción a la optimización de políticas

OpenAI. (s. f.). *Part 3: Intro to policy optimization.* https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

OpenAI analiza los fundamentos matemáticos de los algoritmos de optimización de políticas y conectaremos el material con un código de muestra.

Test

1. ¿Cuál es el objetivo principal del aprendizaje por refuerzo?

   A. Minimizar la pérdida de datos.

   B. Maximizar la precisión de la clasificación.

   C. Optimizar la recompensa acumulada a largo plazo.

   D. Reducir el tiempo de entrenamiento.

2. En el algoritmo Q-learning, ¿qué representa la variable γ (gamma)?

   A. La tasa de aprendizaje.

   B. El factor de descuento.

   C. La recompensa inmediata.

   D. La acción a tomar.

3. ¿Cuál es el propósito de la red objetivo en el algoritmo DQN?

   A. Reducir el tamaño de la tabla Q.

   B. Estabilizar el proceso de aprendizaje.

   C. Aumentar la tasa de aprendizaje.

   D. Minimizar la recompensa acumulada.

4. En el contexto de REINFORCE, ¿qué es el gradiente de la política?

   A. La diferencia entre las recompensas actuales y futuras.

   B. La actualización de los valores Q.

   C. El gradiente de la recompensa acumulada esperada con respecto a los parámetros de la política.

   D. La función de pérdida.

5. ¿Qué ventaja tiene PPO sobre otros métodos de gradiente de políticas?

A. Es más fácil de implementar y tiene mejor complejidad de muestra.

B. Requiere menos datos de entrenamiento.

C. No utiliza redes neuronales.

D. Minimiza la recompensa acumulada.

6. En Q-learning, ¿cómo se define un episodio?

A. Un conjunto de acciones sin recompensa.

B. El proceso de aprendizaje de un solo paso.

C. El final de una etapa donde el agente ha alcanzado el objetivo o ha fracasado.

D. La actualización de la red neuronal.

7. ¿Qué es la tabla Q en el contexto de Q-learning?

A. Una lista de recompensas acumuladas.

B. Un registro de todas las acciones realizadas.

C. Una matriz que mantiene los valores Q para todas las combinaciones de estados y acciones.

D. Una función de política.

8. ¿Cuál es el propósito de la función de pérdida en DQN?

A. Maximizar la recompensa inmediata.

B. Minimizar la diferencia entre los valores Q predichos y los valores Q objetivo.

C. Actualizar los valores de la tabla Q.

D. Seleccionar la mejor acción posible.

9. ¿Qué técnica utiliza REINFORCE para ajustar su estrategia?

A. Diferencias temporales.

B. Gradiente de políticas.

C. Factor de descuento.

D. Red objetivo.

10. En PPO, ¿qué se entiende por «clip objetivo»?

A. Una técnica para asegurar que las actualizaciones de la política no sean demasiado grandes.

B. Un método para inicializar la tabla Q.

C. Un proceso para reducir el número de episodios.

D. Una función para calcular la recompensa inmediata.