UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

Estudio experimental de diversos algoritmos de aprendizaje por refuerzo

Experimental study of some algorithms of Reinforcement Learning

Realizado por
Alejandro Jesús García Carrasco

Tutorizado por
José Luis Pérez de la Cruz

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2020

Agradecimientos

En primer lugar, quiero agradecer a mis seres queridos por el apoyo y el cariño durante todo este tiempo, en especial a mi actual pareja Alison por su apoyo incondicional todos los días y animarme a conseguir lo que me propongo. A mi tutor Jose Luis por su gran ayuda en este trabajo. Y por último a todos mis compañeros de carrera.

Resumen

Los algoritmos desarrollados en este trabajo pretenden resolver mediante aprendizaje por refuerzo tres entornos distintos (Taxi, Cartpole y MountainCar) proporcionados por el framework OpenAI Gym [1] y hacer una comparativa del rendimiento de estos algoritmos. Para ello, una vez implementados los distintos algoritmos, se ejecutarán un número determinado de veces con el fin de poder evitar el factor aleatoriedad y poder hacer una comparativa sobre las mismas condiciones.

El código a desarrollar se podría dividir principalmente en dos partes, el Agente contiene la mayor parte de código y es el “cerebro”, el encargado de resolver el problema y llegar encontrar una solución dado un entorno. Y una segunda parte encargada de reportar los datos del entrenamiento del agente, tanto en forma de gráfica para ver el rendimiento, como en forma de tabla para poder hacer una mejor comparativa.

Para la implementación se utilizan el framework OpenAI Gym, Python 3.7 [2] así como diversas librerías y el entorno PyCharm [3].

Palabras clave: Aprendizaje por refuerzo, Python, algoritmo, QLearning, Doble-QLearning, Sarsa, Expected-Sarsa, alfa, épsilon.

Abstract

The algorithms developed in this work aim to solve through reinforcement learning three different environments (Taxi, Cartpole and MountainCar) provided by the OpenAI Gym framework [1] and to make a comparison of the performance of these algorithms. For this purpose, once the different algorithms have been implemented, they will be executed a certain number of times in order to avoid randomness and to be able to make a comparison under the same conditions.

The code to be developed could be mainly divided into two parts, the Agent, which contains most of the code, the "brain", the one in charge of solving the problem and finding a solution given an environment. And a second part in charge of reporting the agent's training data, both in graphical form to see the performance, and in table form to make a better comparison.

For the implementation, the OpenAI Gym framework, Python 3.7 as well as several libraries and the PyCharm environment are used.

Keywords: Learning by reinforcement, Python, algorithm, QLearning, Double-QLearning, Sarsa, Expected-Sarsa, alpha, epsilon.

Indice

Indice de figuras

Indice de tablas

1

Introducción

1.1 Historia de la inteligencia artificial

Aunque los primeros referentes históricos se remontan a los años 30 con Alan Turing, considerado el padre de la inteligencia artificial, es considerado que el punto de partida data del año 1950, cuando Turing publica un artículo en la revista Mind [4], donde se cuestionaba si las máquinas podrían llegar a pensar, para ello proponía un método para determinar si una máquina puede pensar o no, lo que se conoce como el Test de Turing, que aún a día de hoy sigue siendo objeto de estudio e investigaciones.

Dicho test consiste en poner a una persona a evaluar una conversación entre un humano y una máquina diseñada para generar respuestas similares a las de un humano. La persona sabe a priori que hay un humano y una máquina, pero no sabe cuál es cuál. La persona debe ser capaz de distinguir la máquina del humano, si no es capaz, se dice que la máquina ha pasado el test de Turing.

No obstante fue en 1956 cuando John McCarthy, Marvin Minsky y Claude Shannon usaron el término de inteligencia artificial durante la conferencia de Darthmounth para referirse a "la ciencia e ingeniería de hacer máquinas inteligentes, especialmente programas de cálculo inteligentes". Estos tres científicos vaticinaron que para 1970, estaríamos rodeado de inteligencias artificiales por todo el mundo.

Nada más lejos de la realidad, ya que no fue hasta los años 90 cuando realmente se empezó a tener un progreso notorio en esta área, a causa de las grandes inversiones realizadas por las empresas tecnológicas con el fin de mejorar la capacidad de procesamiento y análisis de la ingente cantidad de datos que se estaban empezando a generar en el mundo digital.

Uno de los grandes hitos de la inteligencia artificial llegó en el año 1997, cuando la empresa tecnológica IBM (*International Business Machines*) , demostró cómo un sistema informático era capaz de vencer al ajedrez al en su momento campeón del mundo Gari Kaspárov. Dicho sistema se denominó *Deep Blue* y sirvió como base para que la sociedad y toda la industria tecnológica cobrara conciencia de la importancia y las posibilidades de la inteligencia artificial.

Desde ese momento hasta la fecha actual, las empresas no han parado de invertir en esta área, como bien se observa en la Figura 1, tomada del informe anual independiente publicado por el Human-Centered Artificial Intelligence Institute de la Universidad de Stanford [5] en el que se presentan los resultados de los avances en materia de investigación sobre inteligencia

artificial que recoge datos de la última década, se puede observar en la figura 1 claramente el crecimiento que ha tenido la financiación privada en inteligencia artificial (a partir de ahora AI, del inglés Artificial Intelligence).

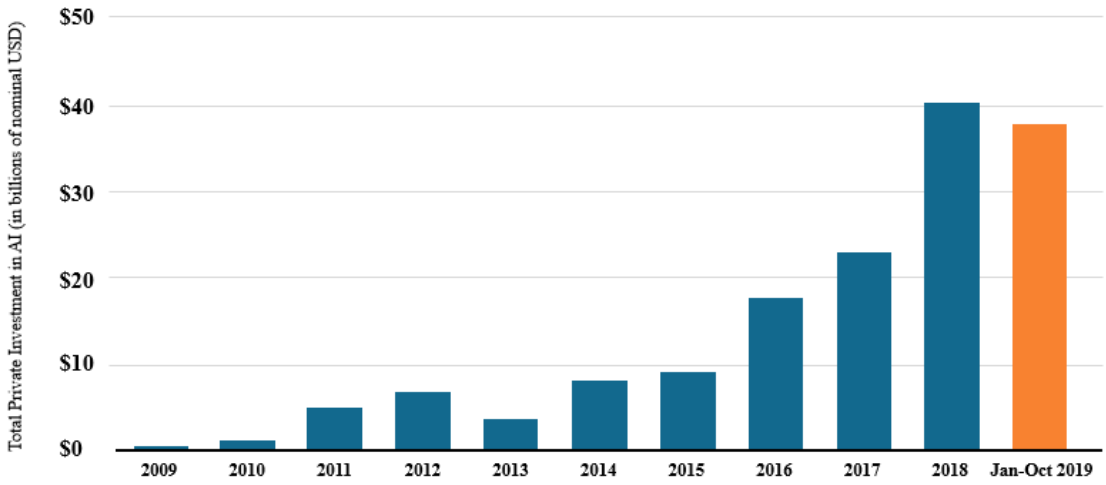

Figura 1. Inversión privada en inteligencia artificial (en miles de millones de dólares). Tomada de [5].

Las distintas técnicas utilizadas para la inteligencia artificial han ido evolucionando a la vez que han aparecido nuevas. En la década de 1980, los sistemas basados en el conocimiento ganaron muchos seguidores, pero a medida que se avanzaba, los investigadores se toparon con el problema de que había que codificar demasiadas reglas, lo que aumentó los costes y redujo mucho el avance de la inteligencia artificial.

Ante este problema, el aprendizaje automático se convirtió en una solución. En lugar de ser las personas quienes codifiquen manualmente cientos de miles de reglas, se programa a las máquinas para que extraigan dichas reglas automáticamente a partir de un grupo de datos. De esta manera, se abandonó los sistemas basados en conocimiento y se han ido refinando las técnicas de aprendizaje automático hasta la fecha actual.

1.2 Objetivos y tecnologías

En el presente trabajo se desarrollarán diversos algoritmos de inteligencia artificial, todos ellos pertenecientes a la rama del aprendizaje por refuerzo. El objetivo es aplicar dichos algoritmos a diferentes problemas y entornos, para poder hacer una comparativa en cuanto a eficiencia y poder sacar así conclusiones sobre qué tipo de algoritmo se adecua mejor a cada tipo de problema.

Para ello, todo el código se desarrollará en el lenguaje Python, en concreto en su versión 3.7. También se usará el framework OpenAI Gym, de la empresa OpenAI. Esta es una compañía de investigación de inteligencia artificial sin fines de lucro. Uno de los objetivos de esta organización es colaborar libremente con otras instituciones e investigadores al hacer sus investigaciones abiertas al público.

El framework OpenAI Gym proporciona una gran cantidad de entornos sobre los que probar nuestra Inteligencia artificial, y esta es una de las grandes ventajas de este framework,

que al proporcionarnos el entorno, lo único que tenemos que diseñar y desarrollar es el agente. Esto es de gran ayuda a los desarrolladores e investigadores, no teniendo así que perder tiempo en codificar y desarrollar los distintos entornos pudiendo así centrarse en el "cerebro" de la IA. OpenAI Gym, proporciona una API con gran cantidad de información y recursos accesibles para todo el mundo (https://gym.openai.com/docs/).

2

Aprendizaje por refuerzo

El campo del aprendizaje automático es la rama de la inteligencia artifical que engloba un conjunto de técnicas que permite hacer aprender a las máquinas a partir de su entorno. Dicho entorno puede considerarse como el conjunto de datos que el algoritmo recibe durante la etapa de entrenamiento. Existen diversos tipos de aprendizaje automático, entre los que destacan:

- **Aprendizaje supervisado.** Se basa en un conjunto de técnicas que permite realizar predicciones a partir de datos de entrenamiento. Dichos datos consisten en pares, en los que una componente son los datos de entrada, y la otra, los resultados deseados.

- **Aprendizaje no supervisado.** Estos algoritmos tienen como objetivo inferir una serie de patrones a partir de un conjunto de datos no etiquetados.

- **Aprendizaje por refuerzo.** El principal objetivo es determinar qué acciones debe escoger un agente de software en un entorno dado con el fin de maximizar una recompensa.

En este presente trabajo nos centraremos en esta última técnica, el aprendizaje por refuerzo, también conocido como RL, de sus siglas en inglés (Reinforcement Learning). En este tipo de aprendizaje, el entorno es formulado generalmente como un proceso de decisión de Markov (MPD). La principal característica de un MDP es que al cambiar de un estado S_t a un estado S_{t+1} tras haber realizado una acción *a*, el estado S_{t+1} solo depende del estado anterior S_t y de la acción *a*, pero no de los estados anteriores a éste. Expresado de otra forma, tal y como nos muestra la propiedad de Markov (Figura 2):

$$\mathbb{P}\left[S_{t+1} \middle| S_t\right] = \mathbb{P}\left[S_{t+1} \middle| S_1, \ldots, S_t\right]$$

Figura 2. Propiedad de Markov.

La cual significa que el estado actual S_t contiene toda la información relevante de los estados pasados (S_1,….., S_t) por lo tanto no es necesario tener información de dichos estados pasados.

Los procesos de decisión de Markov se definen mediante una función de transición, llamada matriz de transición de estados, la cual nos muestra cual sería la probabilidad de pasar de un estado a otro tras tomar una acción. A su vez, al pasar de un estado a otro se obtiene una recompensa inmediata, que viene definida por la función de recompensa.

El aprendizaje por refuerzo consiste en que el agente debe aprender qué acciones tomar para cumplir un objetivo en una tarea maximizando la recompensa obtenida. Dicho de otro modo, consiste en que el agente hace un mapeo entre estados-acciones, y mediante prueba y error, maximiza el valor de cada par para indicar la acción correcta en cada estado. El agente a priori no conoce qué acciones le llevarán en un estado determinado a obtener la recompensa total más alta, sino que debe descubrirlo y aprender. Esto lo consigue mediante exploración-explotación, cuestión que abordaremos más adelante en la siguiente sección.

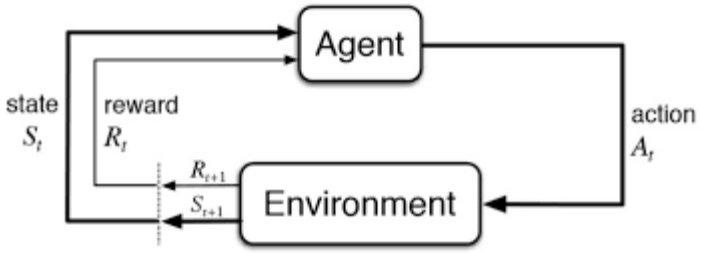

Figura 3. Esquema básico de un proceso de Decisión de Markov.

2.1 Exploración vs Explotación

Uno de los desafíos más grandes que existen en el aprendizaje por refuerzo es el balance entre explotación y exploración.

- **Explotación.** Como su propio nombre indica, el agente debe explotar lo que le ha funcionado y ha aprendido en el pasado.

- **Exploración.** El agente explora el entorno, realizando acciones aleatorias, lo que lo lleva a descubrir el entorno.

Tomar alguna de las dos opciones exclusivamente llevará a error, si el agente únicamente se dedica a explorar, pero no tiene en cuenta lo que va aprendiendo, se dedica a tomar únicamente acciones aleatorias, por otro lado, si agente explota pero no explora lo suficiente, puede que tome una serie de acciones y puede no llegar a descubrir todas las acciones posibles.

Es por ello que estas dos opciones deben estar balanceadas progresivamente, y para ello en el presente trabajo usaremos la estrategia conocida como Epsilon-greedy para la toma de decisiones, que tiene el siguiente funcionamiento (Figura 4):

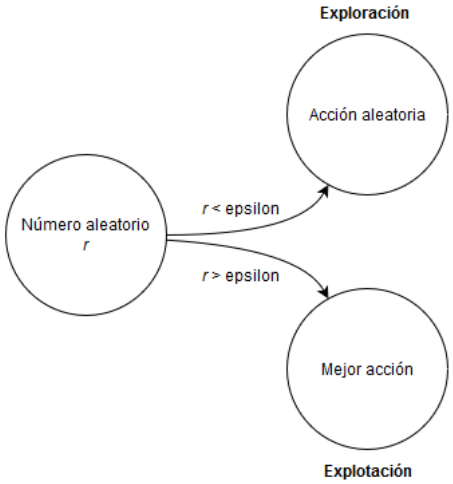

Figura 4. Exploración vs Exploración.

1. Se genera un número aleatorio *r*.

2. En función del valor de *r* se toma una acción:

   Si *r* < épsilon, acción aleatoria
   Si *r* > épsilon, mejor acción posible

3. Se disminuye el valor de épsilon.

4. Volver a iterar.

De esta manera, si omitimos el paso tres, tenemos la estrategia denominada épsilon greedy, en el que la probabilidad de ejecutar una acción aleatoria se mantiene constante.

Si además incluimos el paso tres, tenemos la estrategia conocía como épsilon greedy con enfriamiento estadístico. Actuando de esta forma lo que conseguimos es que en las primeras fases del entrenamiento, el agente tenga más en cuenta la exploración, y según va avanzando, descubriendo el entorno y aprendiendo, tenga en cuenta los datos obtenidos para su explotación.

Por tanto, el criterio del agente a la hora de elegir la siguiente acción, vendrá representado por una distribución de probabilidad P que aumenta conforme avance el tiempo, tal como se observa en la figura 5.

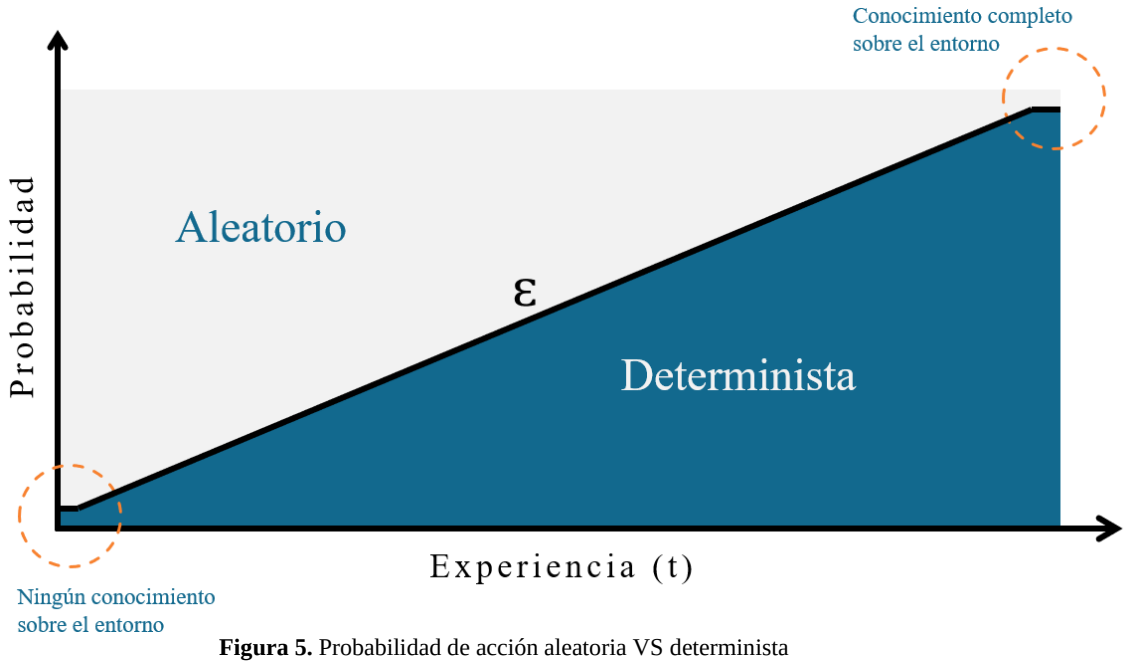

Figura 5. Probabilidad de acción aleatoria VS determinista

2.2 Elementos del Aprendizaje por Refuerzo

Existen varios elementos que intervienen en el Aprendizaje por refuerzo, los cuales de detallarán a continuación.

2.2.1 Función de recompensa

La función de recompensa define el objetivo en un problema de aprendizaje por refuerzo. En cada iteración en la que se toma una acción *a* y se cambia de estado, el agente recibe del entorno un estímulo o recompensa en forma de número, que bien puede ser positiva o negativa. Mediante esta recompensa se establece que estados son negativos y cuales son beneficiosos para el agente. Por hacer un símil con la vida real, un ejemplo podría ser cuando se está educando a un niño pequeño, que a priori no sabe que acciones son las correctas y cuales no, y por ello cuando el niño hace algo mal, se le da un estímulo negativo, bien regañándole, bien en forma de castigo, de esta forma el niño asocia que esa acción, le lleva a un estado perjudicial, y por el lado contrario, si el niño hace algo bueno, por ejemplo cuando dice alguna palabra nueva, se le recompensa, de esta forma el niño asocia acciones "buenas" y acciones "malas", creando así una conducta. Volviendo al algoritmo, las recompensas deben ser cuantificadas en base a una función que tome como entrada el estado del entorno y las acciones seleccionadas.

2.2.2 Política

Formalmente, una política π, es un mapeo entre los estados y las probabilidades de seleccionar cada posible acción. La política determina la manera en la que el agente se comporta en un momento o estado determinado, se podría decir que es lo que hace que el agente

tome la opción óptima en cada estado. La política puede venir dada por una función que devuelve la acción a tomar, o mediante una tabla conteniendo el par estado-mejor acción.

2.2.3 Función de Valor

Otro elemento a la hora de resolver un problema de aprendizaje por refuerzo es la función de valor. El valor de una política π se representa por $V^{\pi}(s)$. Se define $V^{*}(s)$ como el valor máximo posible de $V^{\pi}(s)$. Una política que alcanza estos valores óptimos en cada estado se llama óptima.

Esto es lo que lo diferencia de la función de recompensa, ya que la función de recompensa señala que será lo mejor en un plazo inmediato, mientras que para largo plazo la encargada es la función de valor.

El valor representa la cantidad de recompensa que puede esperar el agente en el futuro, iniciando desde el estado en el que se encuentra. Es decir la recompensa es lo que se debe buscar, y se hace mediante la función de valor, el valor solo sirve para encontrar una mayor recompensa, las acciones se guían en los valores para conseguir una mayor recompensa, es por eso que se buscan acciones que puedan tener un mayor valor, no mayor recompensa, porque dichas acciones son las que en el largo plazo obtendrán una mayor recompensa. Un claro ejemplo puede ser un estado en el que se otorgue una recompensa inmediata de pequeño valor (o incluso valor negativo), sin embargo, dicho estado puede ser el punto de partida para llegar a otros estados que proporcione una recompensa mucho mayor.

Dado un estado *s*, y una acción a, $Q(s,a)$ representa el valor retornado al tomar una acción a en el estado s, por consiguiente, $Q^{\pi}(s,a)$ representa el valor retornado al tomar una acción a en el estado s siguiendo la política π. Dado que el valor de un estado es el máximo que le da la acción, entonces, $V^{\pi}(s) = \max Q^{\pi}(s,a)$. De igual manera $Q^{*}(s,a)$ es el máximo valor que se puede obtener para cualquier política de $Q^{\pi}(s,a)$

Desgraciadamente, aprender la función de valor es mucho más complejo que determinar la función de recompensa, y esto es debido a que la recompensa los otorga el entorno, mientras que la función de valor viene determinado por las múltiples observaciones del agente en el entorno. Por ello, uno de los elementos más importantes en un problema de aprendizaje por refuerzo, es determinar una función de valor correctamente.

2.2.4 Modelo

Un proceso de decisión de Markov queda definido por el modelo, es decir, las probabilidades de transición y las recompensas inmediatas. Ahora bien, normalmente cuando se aplica un algoritmo de aprendizaje, este modelo no es conocido. Entonces se habla de un algoritmo de aprendizaje *model-free*.

El último componente el cual es opcional, es el modelo del entorno, en el cual se puede estudiar a qué estados se llegan cuando se eligen determinadas acciones al estar situado en estados previos. Los métodos que usan modelos se denominan métodos basados en modelos, mientras que existen métodos que no están basados en modelos, los cuales se denominan *model-free*, y son explícitamente agentes de ensayo-error.

3

Algoritmos

En esta sección se detallarán los diferentes algoritmos desarrollados para el presente trabajo. Todos ellos son algoritmos específicos de lo que se denomina como diferencia temporal. El principal punto en común que tienen estos algoritmos es que vamos a considerar la opción tabular, es decir, al agente se alimenta de lo que se conoce como la Q-tabla, que no es más que una tabla donde se almacenan todos los posibles pares estado-acción, con su correspondiente valor, lo que indica la idoneidad de elegir una acción en un estado determinado. La tabla 1 muestra como puede ser un ejemplo de una Q-Tabla.

Q-Tabla		Acciones				
		Acción 1	Acción 2	⋯	Acción N-1	Acción N
Estados	Estado1	0,65	0,49	⋯	0,37	0,35
	Estado 2	0,78	0,41	⋯	0,24	0,74
	⋯	⋯	⋯	⋯	⋯	⋯
	Estado N-1	0,38	0,64	⋯	0,46	0,56
	Estado N	0,76	0,40	⋯	0,62	0,63

Tabla 1. Ejemplo Q-Tabla

3.2 Q-Learning

El primer algoritmo se trata de Q-Learning, el cual es un algoritmo *model-free*. El objetivo de Q-Learning es aprender una serie de normas que le digan al agente, cual es la acción idónea en un determinado estado. Q-Learning encuentra una política óptima en el sentido de que maximiza el valor esperado de la recompensa total empezando desde el estado actual. Este puede identificar una política óptima en la selección de acción para cualquier proceso de decisión de Markov finito (PDMF del inglés finite Markov decision process), dado un tiempo de exploración infinito y una política parcialmente aleatoria.

Q-Learning es un algoritmo *off-policy* (al contrario que SARSA, el cual detallaremos posteriormente), y eso es debido a que actualiza sus valores Q, también conocidos como Q-valores, actualiza Q(s,a) utilizando el mejor Q-valor del estado s' al que se llega ejecutando la acción a Q(s',a), lo que quiere decir que estima la recompensa futura descontada total olvidandose de que va a seguir una política greedy.

Cada valor asociado a cada par estado-acción, se almacena en lo que se conoce como la Q-tabla, y la regla de actualización que se sigue es la siguiente:

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

En esta función hay varios parámetros que influyen y se han de tener en cuenta:

- **Ratio de aprendizaje (α).** También conocido como alfa o en inglés como *learning rate*. Este establecerá y determinará hasta qué punto la información adquirida sobrescribe la información vieja. Un ratio de aprendizaje igual a 0 hace que el agente no aprenda, aprovechando así únicamente el conocimiento previo, mientras que en el caso opuesto, con un ratio de aprendizaje igual a 1 hace que el agente solo tenga en cuenta la información más reciente, ignorando el conocimiento previo, para explorar nuevas posibilidades)

- **Factor de descuento (γ).** También conocido en inglés como *discount factor*. Este determina la importancia de recompensas futuras. Un factor de 0 hará que el agente no tenga en cuenta posibles recompensas futuras, considerando así únicamente las recompensas actuales. Mientras que un factor de 1 hará que el agente "luche" por una recompensa alta a largo plazo.

El algoritmo 1 muestra en pseudo código el funcionamiento de el algoritmo Q-learning.

Algoritmo 1. Q- Learning (off-policy TD) para estimar $\pi \approx \pi_*$

Inicializar $\mathrm{Q}(s, a)$, para todo $s \in S$, $a \in A(s)$, arbitrariamente, $\mathrm{Q}(terminal - state, \cdot) = 0$

for (cada episodio):
  Inicializar S_t
  for (cada paso en episodio actual):
    Elegir A_t de S_t utilizando política derivada de Q (i.e., ε-greedy)
    Ejecutar acción A, observar R, S_{t+1}
    $\mathrm{Q}(S_t, A_t) = \mathrm{Q}(S_t, A_t) + \alpha[R + \gamma\, max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
    $S_t \leftarrow S_{t+1}$
  Hasta S estado final
end

3.2 Doble Q-Learning

Una variación del algoritmo Q-learning es el Doble Q-learning (Double Q-learning en inglés), esta variación surge a raíz de que tal y como explica Hado van Hasselt en su artículo Double Q-Learning [6], el algoritmo Q-learning se desempeña muy mal en algunos entornos estocásticos, y este bajo rendimiento se debe a una gran sobreestimación de los valores de

acción debido al uso de $max_a Q(S_{t+1}, a)$ en Q-learning, a continuación lo expondremos con más detalle.

3.2.1 El problema

El ejemplo que se muestra a continuación, está tomado el libro de Sutton y Barto [7]. Consideremos un MDP con cuatro estados (Figura 6), de los cuales dos son estados terminales (C y D). El estado inicial es siempre A, a partir del cual se pueden tomar dos acciones, ir a la izquierda (al estado B) o ir a la derecha (al estado C), ambas acciones devuelven una recompensa de 0, pero en este caso la correcta es moverse a la derecha, e ir al estado terminal C con una recompensa de 0.

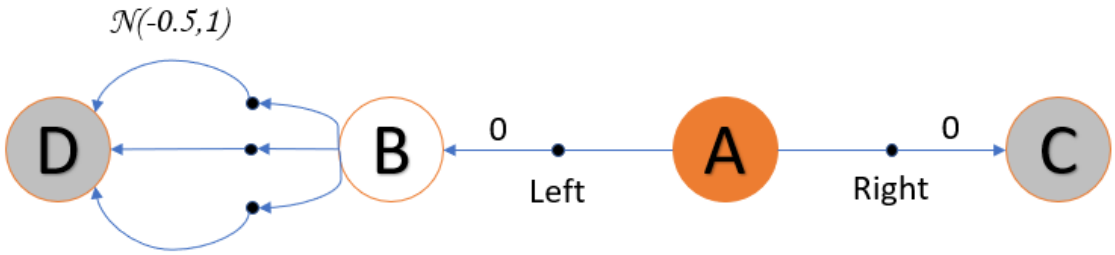

Figura 6. Ejemplo de MDP descrito.

El estado B tiene una serie de acciones que mueven al agente al estado terminal D. Sin embargo (y aquí es donde empieza el problema), la recompensa R de cada acción de B a D es un valor aleatorio que sigue una distribución normal con media -0.5 y una varianza de 1.0. Esto quiere decir que el valor esperado de R es negativo (-0.5), por lo que en un gran número de experimentos, el valor promedio de R es menor que 0.

Sobre estas condiciones, está claro que partiendo desde el estado inicial A, la opción correcta es moverse hacia C. Sin embargo, debido a que algunos de los valores de R son positivos, Q-learning será "engañado" para moverse a la izquierda y así maximizar la recompensa, pero esto es una mala decisión, porque incluso si funciona para algunos episodios, a la larga se garantiza una recompensa negativa.

Entonces, ¿por qué sobrestima Q-learning? Para responder a esta pregunta, planteemos el siguiente escenario:

Se tiene X1 y X2, dos variables aleatorias que representan la recompensa al tomar dos acciones en el estado B. Dado que son variables aleatorias, calcularemos sus valores esperados E(X1) y E(X2). Sin embargo dado que no es posible saber sus valores esperados, lo que haremos será usar estimaciones de esos valores esperados al calcular el promedio incremental de μ_1 y μ_2. Esas estimaciones son imparciales porque a medida que aumenta el número de iteraciones y de muestras, el promedio de todo el conjunto de valores se acerca a E(X1) y E(X2) como se muestra en la siguiente tabla:

Iteración	X1	μ_1	X2	μ_2	Max(μ)
1	3	3	-8	-8.00	3.00
2	-2	0.50	-5	-6.50	0.50
3	4	1.67	-8	-7.00	1.67
4	-1	1.00	-10	-7.75	1.00
5	-5	-0.20	5	-5.20	-0.20
6	2	0.17	-7	-5.50	0.17
7	-6	-0.71	-3	-5.14	-0.71
8	-3	-1.00	3	-4.13	-1.00
9	8	0.00	8	-2.78	0.00
10	-6	-0.60	9	-1.60	-0.60
11	-9	-1.36	2	-1.27	-1.27
12	9	-0.50	-1	-1.25	-0.50
13	1	-0.38	2	-1.00	-0.38
14	-1	-0.43	8	-0.36	-0.36

Tabla 2. Sobrestimación de Q-Learning.

De esta tabla podemos sacar los siguientes datos:

- *E(X1)* = -0.43
- *E(X2)* = -0.36
- *Max(E(X))* = -0.36
- *E(Max(μ))* = 0.09

De estos datos se puede deducir lo siguiente, dado que Q-learning utiliza $max_a Q(S_{t+1}, a)$, representado en la tabla por Max(μ), se puede ver claramente que *E(Max(μ))* es diferente de *Max(E(X))*, o lo que es lo mismo, la estimación de los máximos es diferente del máximo de las estimaciones. Esto indica que Max(μ) no es un buen estimador para *Max(E(X))*, y esto es porque se encuentra sesgado. Dicho de otro modo, al actualizar $Q(S_t, a)$ con $max_a Q(S_{t+1}, a)$, no se está moviendo hacia el valor esperado de las acciones en el estado B, que es -0.5.

3.2.2 La solución

La solución propuesta es hacer una estimación doble de los valores con lo que se evita la sobrevaloración. Para ello se tienen dos funciones de Q-value, Q_1 y Q_2, cada una se actualiza con los valores de la otra respectivamente. Esto se realiza con el fin de reducir el sesgo que surge de usar una única función Q. De este modo, las dos funciones pueden compartir experiencias que no tienen en común y así poder explorar mejor. En el algoritmo 2 se describe en pseudo código el funcionamiento del algoritmo Doble Q-learning.

Algoritmo 2. Doble Q- Learning

Inicializar $Q_1(s,a)$, $Q_2(s,a)$, para todo $s \in S$, $a \in A(s)$, tal que, $Q(terminal-state,\cdot) = 0$

for (cada episodio):
  Inicializar S_t
  for (cada paso en episodio actual):
    Elegir A_t de S_t utilizando política ε-greedy de $Q_1 + Q_2$
    Ejecutar acción A_t, observar R, S_{t+1}
    Con probabilidad 0.5:
      $Q_1(S_t,A_t) = Q_1(S_t,A_t) + \alpha\left[R + \gamma Q_2(S_{t+1}, argmax_a Q_1(S_{t+1},a)) - Q_1(S_t,A_t)\right]$
    else:
      $Q_2(S_t,A_t) = Q_2(S_t,A_t) + \alpha\left[R + \gamma Q_1\left(S_{t+1}, argmax_a Q_2(S_{t+1},a)\right) - Q_2(S_t,A_t)\right]$
    $S_t \leftarrow S_{t+1}$
  Hasta S estado final
end

Por todo lo descrito anteriormente, el algoritmo Doble Q-learning, puede ser una mejor opción, según que tipo de problema se vaya a afrontar.

3.3 SARSA

Otros de los algoritmos que derivan del método basado en diferencia temporal y que se desarrolla en el presente trabajo es el de SARSA, el cual es un método on-policy, es decir, va estimando los valores correspondientes a la política que está ejecutando. Es aquí donde se diferencia de Q-learning, y es que en este, la actualización de los valores no se basa en la política a evaluar sino en una búsqueda directa de la política óptima, lo que se denomina off-policy.

La regla de actualización de los valores para el algoritmo SARSA es la siguiente:

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

Esta actualización se lleva a cabo en cada transición de un estado S_t a otro S_{t+1}. En la regla de actualización anterior, se pueden observar que se utilizan los elementos $\langle S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}\rangle$ lo que da lugar a su nombre SARSA.

En este caso, dado que sí se tiene en cuenta la política para elegir una acción, el agente aprenderá que por determinados caminos puede encontrarse con ciertos peligros, dado que en un estado puede haber caminos siguientes buenos y malos, pero si siempre se tiene en cuenta

la acción óptima, el agente no ve los caminos malos, como es el caso de Q-learning. Este comportamiento se observa muy bien en el problema conocido como Cliff walking, (caminar por el acantilado en español). El problema es el siguiente (ejemplo tomado del libro de Sutton y Barto [7]), el agente tiene que llegar desde un estado inicial S, a un estado final o meta M, el agente puede tomar las siguientes acciones, moverse arriba, abajo, izquierda o derecha, cada paso que da, otorga una recompensa de -1, excepto si se mueve hacia el "acantilado", lo que le retorna una recompensa de -100 y devuelve al agente al estado inicial S.

Figura 7. Ejemplo del problema Cliff Walking. Q-learning vs SARSA.

Resolviendo este problema mediante el algoritmo SARSA o mediante Q-learning, se puede observar en la figura 6 como el agente toma dos caminos diferentes. Se observa como el camino óptimo (Q-learning, de color naranja) es el más eficiente, dado que resuelve el problema en un menor número de pasos, a diferencia del camino más seguro (SARSA, de color azul), que a pesar de resolverlo en un mayor número de pasos, es más seguro, ya que se aleja lo más posible del acantilado.

El funcionamiento de SARSA se describe a continuación en el algoritmo 3.

Algoritmo 3. Sarsa (on-policy TD control)

Inicializar Q(s, a), para todo $s \in S, a \in A(s)$, arbitrariamente, Q$(terminal - state, \cdot) = 0$

for (cada episodio):
  Inicializar S_t
  Elegir A de S_t utilizando política derivada de Q (i.e., ε-greedy)
  for (cada paso en episodio actual):
    Ejecutar acción A_t, observar R, S_{t+1}
    Elegir A_{t+1} de S_{t+1} utilizando política derivada de Q (i.e., ε-greedy)
    $Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
    $S_t \leftarrow S_{t+1}$
    $A_t \leftarrow A_{t+1}$
  Hasta S estado final
end

3.4 Expected SARSA

Una variante del Algoritmo anterior SARSA es el denominado Expected SARSA, el cual, utiliza el valor esperado, para ello utiliza la siguiente regla de actualización.

$$\mathrm{Q}(S_t, A_t) = \mathrm{Q}(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum \pi(a|S_{t+1}) Q(S_{t+1}, \mathrm{a}) - Q(S_t, A_t)]$$

El algoritmo Expected SARSA ofrece mejores resultados que SARSA ya que elimina la aleatoriedad de A_{t+1} , a cambio de un mayor coste de computación.

4

Entornos

Como se ha hablado anteriormente, OpenAI Gym proporciona una gran cantidad de entornos, desde algunos más sencillos con tan solo un par de acciones discretas y unos cuantos estados, a algunos más complejos con acciones continuas y un espacio de estados continuo también.

En este apartado pasaremos a describir los distintos entornos que se van a utilizar en este trabajo, los cuales son todos deterministas.

4.1 Taxi-v3

La figura 7 muestra como se vería gráficamente el entorno Taxi-v3. En este entorno hay 4 localizaciones nombradas por diferentes letras (R, G, Y, B), el objetivo es que el taxi (celda amarilla en la figura 8) tiene que coger al pasajero marcado de color azul (G en este caso) y llevarlo a su destino (B, marcado de color violeta en este caso). Una vez el taxi suelta al pasajero en su destino, la tarea se ha completado con éxito y el episodio termina.

Figura 8. Entorno Taxi-v3.

Cada desplazamiento del taxi, le devuelve una recompensa de -1, excepto por soltar al pasajero en su destino, el cual se ve recompensado con +20, o tomar una acción ilegal (recompensa de -10), que puede ser o bien intentar moverse fuera de los límites, o ejecutar una acción que no tenga sentido, como por ejemplo recoger al pasajero en una localización donde este no se encuentre, recoger al pasajero una vez esté ya esté en el taxi, etc. Ejecutar este tipo

de acciones ilegales no terminan el episodio, el episodio se da por terminado únicamente cuando el taxi ha recogido al pasajero y lo ha llevado a su destino.

En este entorno el agente puede tomar 6 diferentes acciones:

- Desplazarse al sur
- Desplazarse al norte
- Desplazarse al este
- Desplazarse al oeste
- Coger un pasajero
- Soltar un pasajero

El espacio de estados está compuesto por un total de 500 estados diferentes, ya que hay 25 posibles posiciones para el taxi, 5 posibles localizaciones para el pasajero (incluido el caso cuando al pasajero está en el taxi) y 4 posibles localizaciones destino.

Para facilitar el desarrollo, cada vez que el agente toma una acción, el entorno devuelve lo que se denomina una observación que no es más que un estado, el cual está compuesto por conjunto de datos que se obtienen del entorno. En este entorno una observación está formada por la tupla (fila_taxi, columna_taxi, localización pasajero, destino).

4.2 CartPole-v1

En la figura 9 se muestra como se ve gráficamente el entorno CartPole-v1. Un poste está unido a un carro, que se mueve sin fricción a lo largo de una pista. El péndulo comienza en una posición vertical y el objetivo es evitar que se caiga aumentando y reduciendo la velocidad del carro, desplazándose de izquierda a derecha.

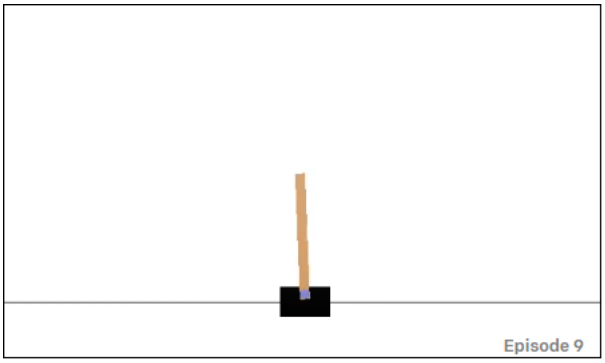

Figura 9. Entorno CartPole-v1.

En este entorno un episodio se ve terminado cuando ocurre una de las siguientes situaciones:

- El ángulo del poste es mayor o menor que 12º.
- La posición del carro es mayor o menor que 2.4 (salirse fuera de la pantalla)
- La duración del episodio llega a 200 iteraciones.

La recompensa en este entorno se obtiene de una manera muy sencilla, en cada iteración que el episodio no haya terminado, el agente obtiene +1 de recompensa. O lo que es lo mismo, cada paso que dé el agente en que el poste no caiga, o el carro no se salga de la pantalla, este recibe una recompensa positiva, hasta llegar al límite de 200.

Para mantener el poste recto y evitar que el carro se salga de los límites de la pantalla, el agente puede tomar dos acciones, mover el carro a la izquierda o a la derecha.

En cada iteración, el entorno nos devuelve una tupla con la siguiente información (Posición del carro, Velocidad del carro, Ángulo del poste, Velocidad del poste en la punta). En la siguiente tabla se muestran los límites superiores en inferiores de las 4 variables de la observación.

Observación	Min	Max
Posición del carro	-2.4	2.4
Velocidad del carro	-Inf	Inf
Ángulo del poste	~ -41.8°	~ 41.8°
Velocidad del poste en la punta	-Inf	Inf

Tabla 3. Máximos y mínimos de las 4 variables de la observación.

En este entorno, y dado que un estado está compuesto por las variables de la tabla 3, las cuales son variables continuas, el espacio de estados, que no es más que el conjunto de estados posibles, es infinito, lo que supone un problema que se abordará en más adelante.

4.3 MountainCar-v0

En la figura 10 se observa el entorno MountainCar-v0. Un coche se encuentra en una pista unidimensional posicionada entre dos “montañas”. El objetivo es subir la montaña a la derecha y llegar a la meta (bandera amarilla), sin embargo el motor del coche no es demasiado fuerte como para escalar la montaña de una sola vez, por lo tanto la única forma de llegar a la meta es conducir de un lado a otro para coger impulso.

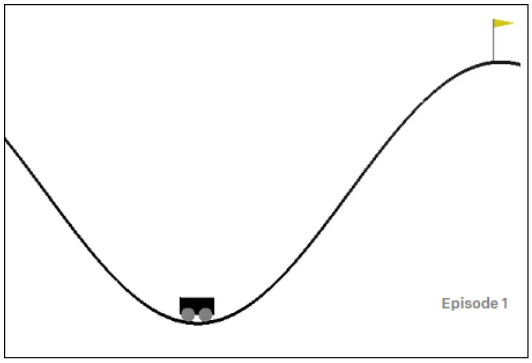

Figura 10. Entorno MountainCar-v0.

El agente puede tomar 3 acciones diferentes, moverse a la izquierda, moverse a la derecha, o no moverse.

El agente obtiene una recompensa de -1 por cada paso (o time step), hasta que alcanza la posición 0.5 de la meta. El episodio termina o bien cuando al agente consigue llegar a la meta, o cuando se itera 1000 veces.

La observación que nos devuelve el entorno está compuesta por una tupla (posición, velocidad) en la que la posición es un valor continuo que puede ir desde -1.2 hasta 0.6 y la velocidad es otro valor continuo, cuyo valor puede ir desde -0.07 hasta 0.07.

5

Resultados

Una vez descrito tanto los distintos algoritmos que se usarán en este trabajo, como los diferentes problemas a resolver, en este apartado se hará una comparativa del rendimiento al aplicar los algoritmos y variar los parámetros.

Antes de nada, mencionar que las gráficas que se verán a continuación han sido tratadas para facilitar su comprensión y sacar conclusiones de una forma más clara, y es que en el aprendizaje del agente, el cual no es un aprendizaje lineal, puede no interpretarse bien los resultados, es por ello que se ha calculado la media en cada episodio respecto a los episodios anteriores con el objetivo de "suavizarla", es decir, para representar la recompensa obtenida en el episodio 200 por ejemplo, se calcula la media de las recompensas de los últimos n episodios. En la tabla 4 se muestra un ejemplo sencillo del funcionamiento.

Episodio	Recompensa en el episodio actual	Media (2 ultimos episodios)
1	10	10
2	40	25
3	13	26,5
4	60	36,5
5	20	40
6	130	75
7	40	85
8	90	65
9	110	100
10	140	125

Tabla 4. Ejemplo sencillo de datos antes de suavizar y después de aplicar la media.

En las figuras 11 y 12 se observan las gráficas generadas para estos datos, se puede observar claramente, como en la figura 12, los datos son más homogéneos.

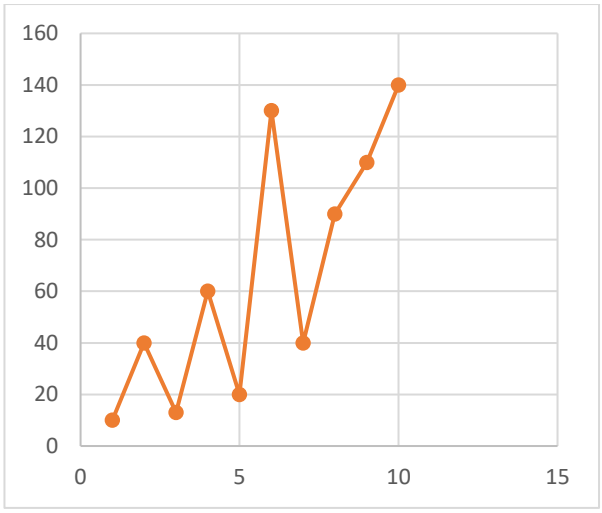

Figura 11. Ejemplo antes de suavizar los datos.

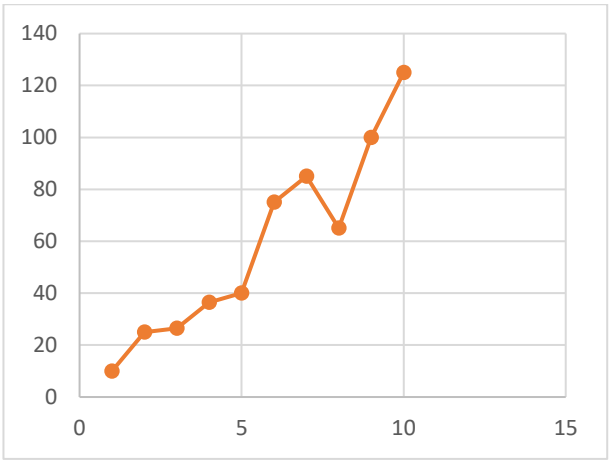

Figura 12. Ejemplo después de suavizar los datos.

La figura 13 muestra un ejemplo de cómo es el aprendizaje realmente de dos agentes, y la figura 14 muestra la misma grafica después de la corrección.

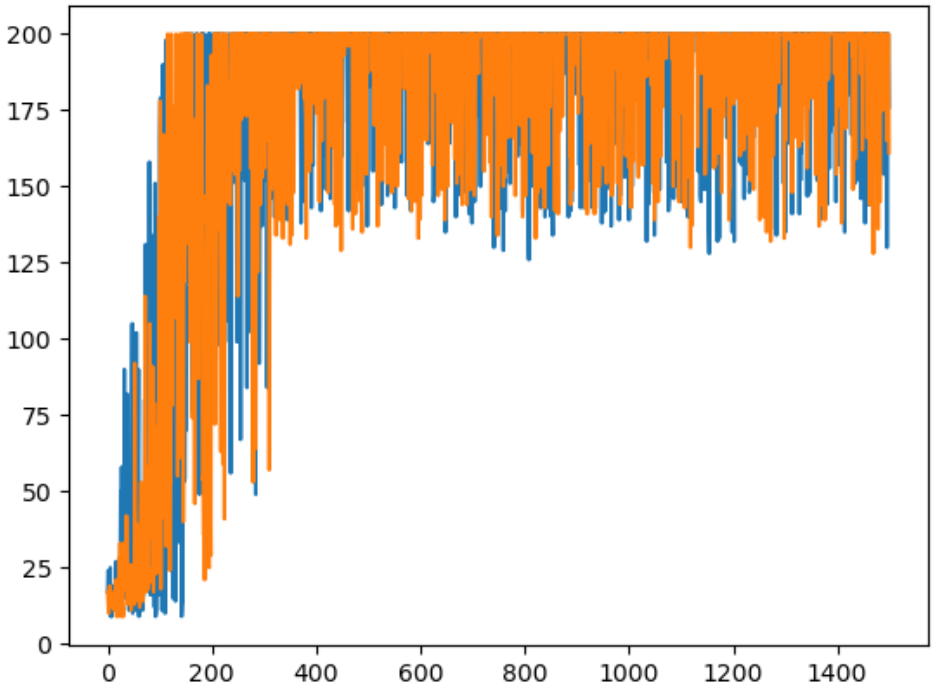

Figura 13. Evolución del aprendizaje de dos agentes distintos.

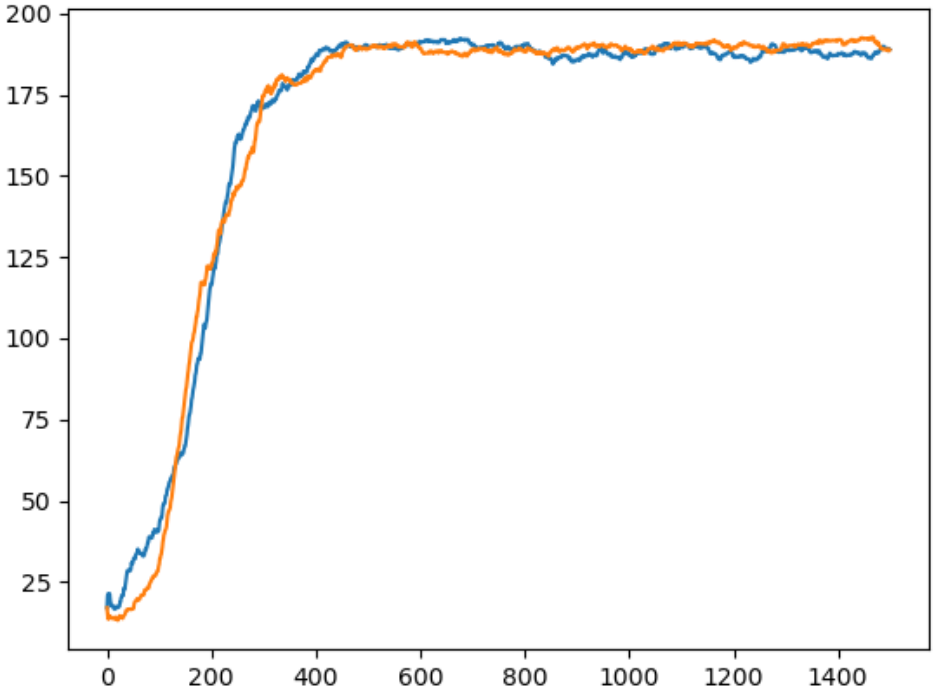

Figura 14. Evolución del aprendizaje después de corrección.

5.1 Taxi-v3

Para este entorno a fin de poder medir la eficiencia de los distintos algoritmos, se han evaluado los siguientes parámetros en cada algoritmo:

1. Medir la eficacia del aprendizaje, esto lo medimos con el número de episodios que son necesarios para que el valor se estabilice.

2. Recompensa obtenida cuando se ha acabado el aprendizaje (la cual está relacionada con el número de pasos)

Para poder calcular el primer parámetro, se ha entrenado a cuatro agentes distintos (uno por cada algoritmo) durante cien mil episodios. Una vez entrenado nuestro agente, su Tabla donde almacena los valores para cada par estado-acción ha tomado los valores óptimos para tomar las decisiones correctas. Tras esto, se han ejecutado de nuevo cien mil episodios, pero estos con intención de medir cual es el número de pasos necesarios para resolver el problema, es decir, cada vez que el agente lleva al pasajero a su destino correcto, se mide el número de pasos que se han necesitado, y se resetea el entorno, iterando así una y otra vez (esta vez sin alterar la Q-Tabla). Tras esto se hace una media de los pasos necesarios y la puntuación obtenida. Nótese que el número de pasos y la puntuación están correlacionados: a menor número de pasos necesarios para resolver el problema, mayor puntuación, ya que como se explicó anteriormente, cada paso devuelve una recompensa de -1. Es por esto que en lugar de calcular la recompensa obtenida, calculamos el número de pasos necesarios para resolver el problema, porque a pesar de medir realmente lo mismo, consideramos que el número de pasos es más descriptivo.

Para poder calcular el número de episodios necesarios en los que el agente resuelve el problema, primeramente es necesario definir cuando se ha resuelto el problema. En este caso, tal y como se detallaba en la sección 4.1 donde se describía el problema del Taxi, este se da por resuelto una vez el agente (Taxi) recoge al pasajero y lo lleva a su destino. Pero dado que nuestro agente puede llevar a cabo esta tarea de forma aleatoria en un episodio temprano, necesitamos poder medirlo de una forma en que se reduzca la aleatoriedad. Para ello, se da por resuelto el problema, cuando el agente lleva al pasajero a su destino en un número inferior a 15 pasos, teniendo en cuenta los últimos 50 episodios, es decir en cada episodio, se calcula el número de pasos necesarios para llevar al pasajero a su destino, si la media de los últimos 50 episodios es inferior a 15, el agente ha logrado resolver el problema. Con esto reducimos la aleatoriedad. Pero dado que el agente puede resolver el problema en un determinado número de episodios, no quiere decir que siempre lo hará en ese mismo número, es por eso que se ha entrenado al agente desde cero unas 50 veces, y tras esto se hace una media del número de episodios necesarios que ha necesitado para resolver el problema, con esto medimos con mayor exactitud la segunda métrica.

Una vez descritas las diferentes métricas a tener en cuenta, antes de pasar a comparar los algoritmos entre sí, es necesario encontrar los parámetros alfa y épsilon que mejor se ajustan a cada algoritmo. Para ello, se han llevado a cabo diferentes pruebas variando dichos parámetros, y he aquí los resultados para los 4 algoritmos.

Antes de comenzar destacar que aunque en las gráficas siguientes no están todos los pares posibles de alfa-épsilon, ya que son infinitos, y el incluir demasiados pares dificultaría la

visualización y comprensión de dichas gráficas, se ha optado por diferentes combinaciones manteniendo bien alfa = 0.9 o alfa = 0.1 ya que en pruebas previas se ha observado como para valores intermedios de alfa, se obtienen peores resultados.

Q-Learning. Para este algoritmo se puede observar en la figura 15 claramente en color negro como lo que da mejor resultados es tener un valor de **alfa = 0.9**, y **épsilon = 0.2**. Tras diversas pruebas realizadas con distintos valores de alfa y épsilon, es lo que mejor resultados ha dado.

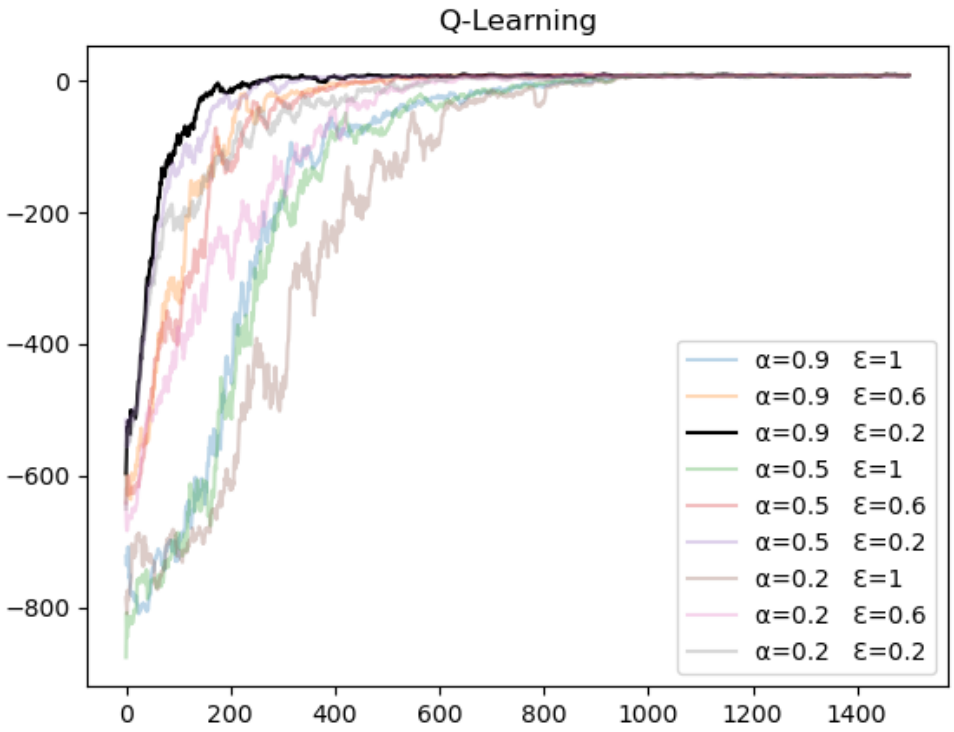

Figura 15. Alfa-épsilon. Q-Learning en el entorno Taxi-v3.

En la Figura 15 se puede observar como parece converger a 0, pero realmente converge a un valor cercano a 2, que es la recompensa obtenida, tal como se puede observar en la figura 16, que es la misma gráfica pero ampliada.

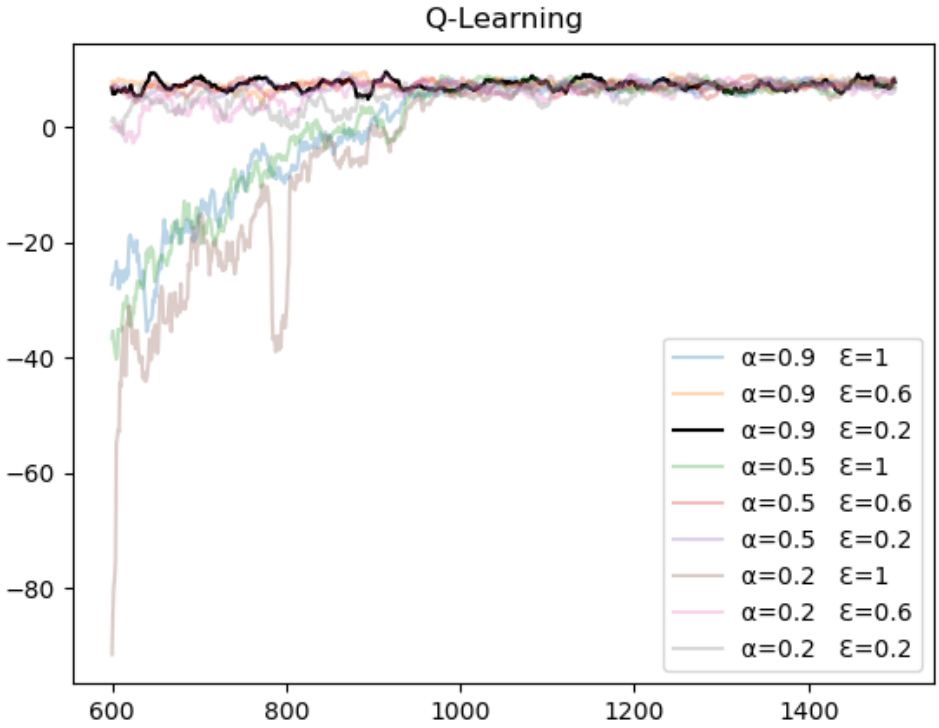

Figura 16. Figura 15 ampliada.

Double Q-Learning. Para este algoritmo hay que tener en cuenta dos cosas. Tal como se observa en la figura 17 hay remarcadas dos líneas, la primera con un color gris oscuro, que es la correspondiente a los valores de alfa = 0.9 y épsilon = 0.2. La segunda es la coloreada de negro, correspondiente a los valores de **alfa = 0.5** y **épsilon = 1**. Como se observa en la figura 17, la primera línea descrita anteriormente (color gris oscuro) empieza a obtener mejor recompensa de forma más temprana, sin embargo, llega un momento en que apenas mejora, es por ello se ha optado por tomar los valores de la segunda línea (color negro) ya que es la que maximiza la recompensa.

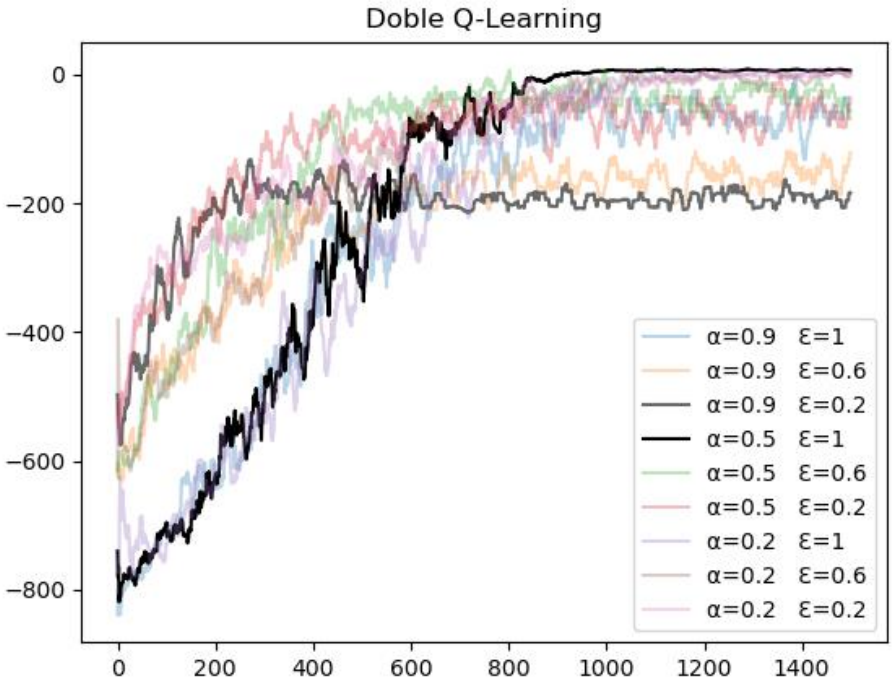

Figura 17. Alfa-épsilon. Doble Q-Learning en el entorno Taxi-v3.

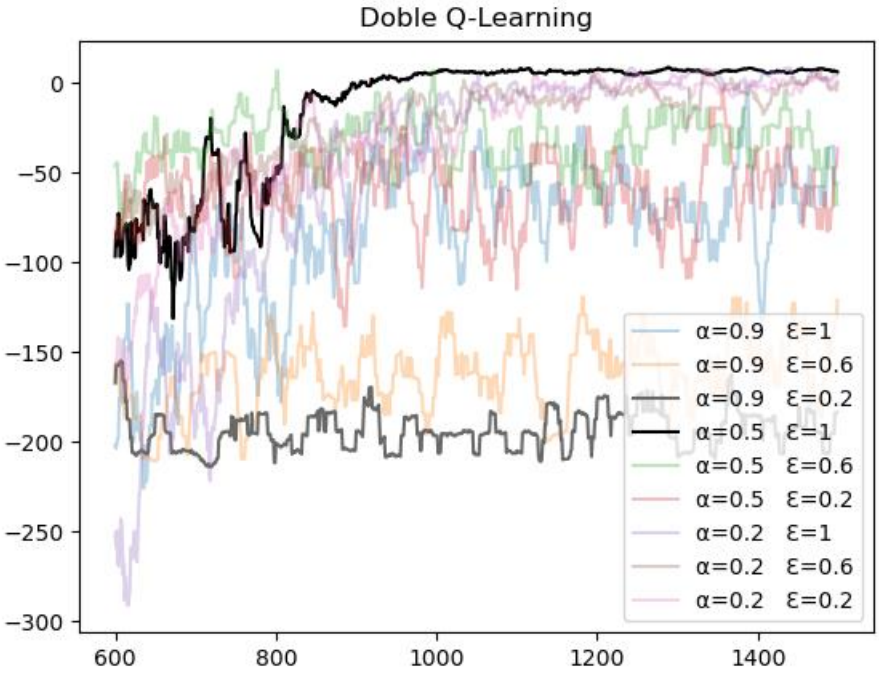

Figura 18. Figura 17 ampliada.

SARSA. En este algoritmo se puede observar en la figura 19 como los valores de **alfa = 0.5** y **épsilon = 0.2** son los que mejores resultados da.

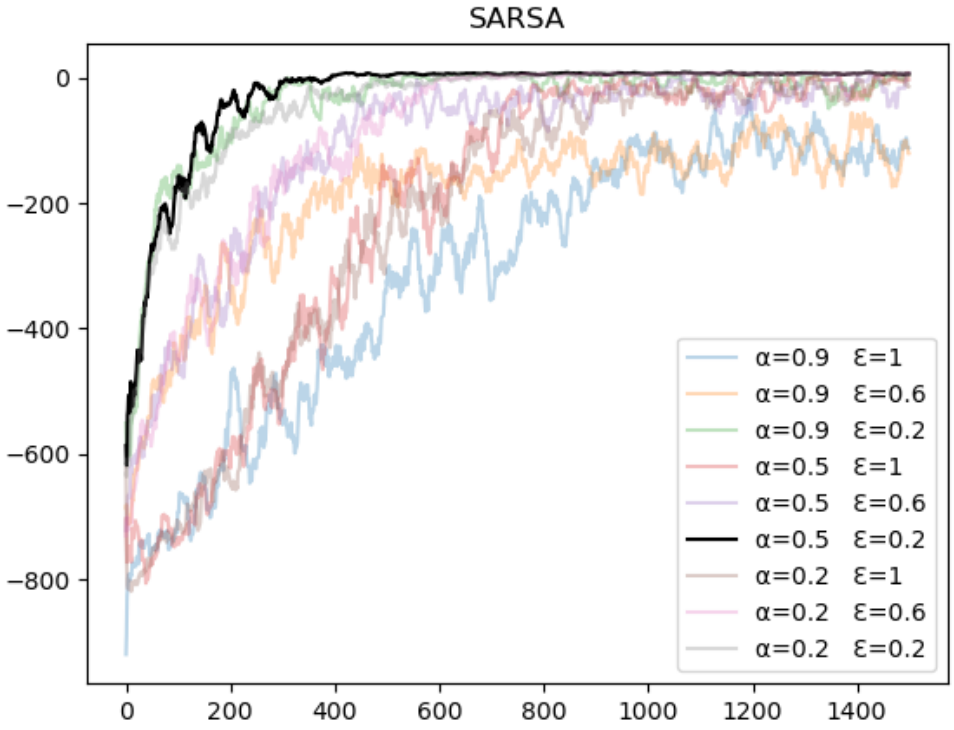

Figura 19. Alfa-épsilon. SARSA en el entorno Taxi-v3.

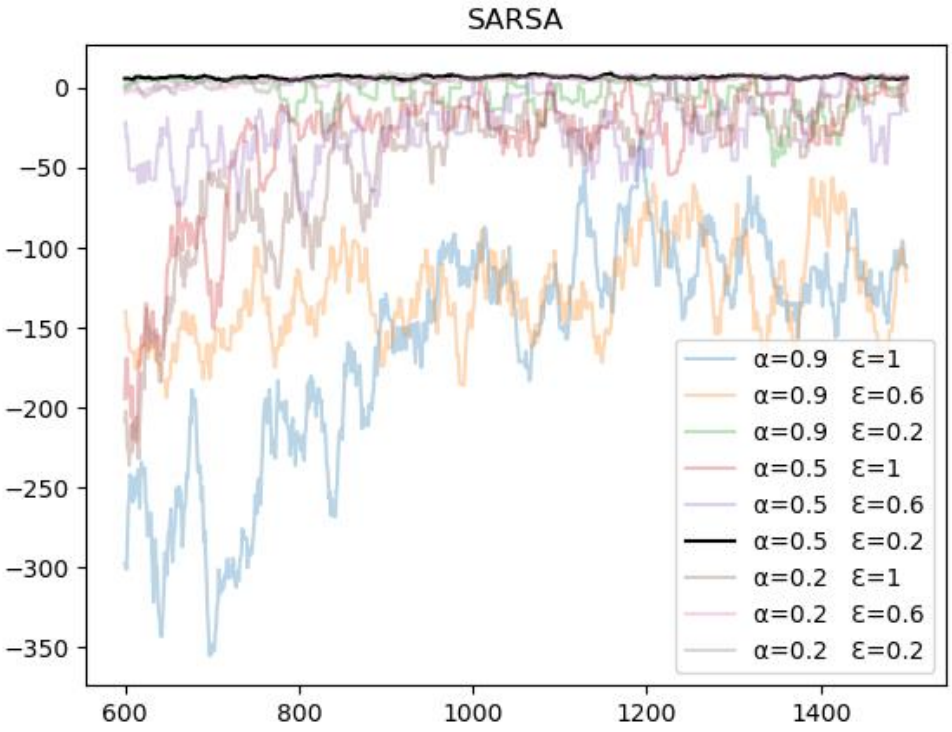

Figura 20. Figura 19 ampliada.

Expected SARSA. Para este algoritmo, tal como se puede observar en la figura 21, los valores de **alfa = 0.9** y **épsilon = 0.2** son los que mejores resultados da.

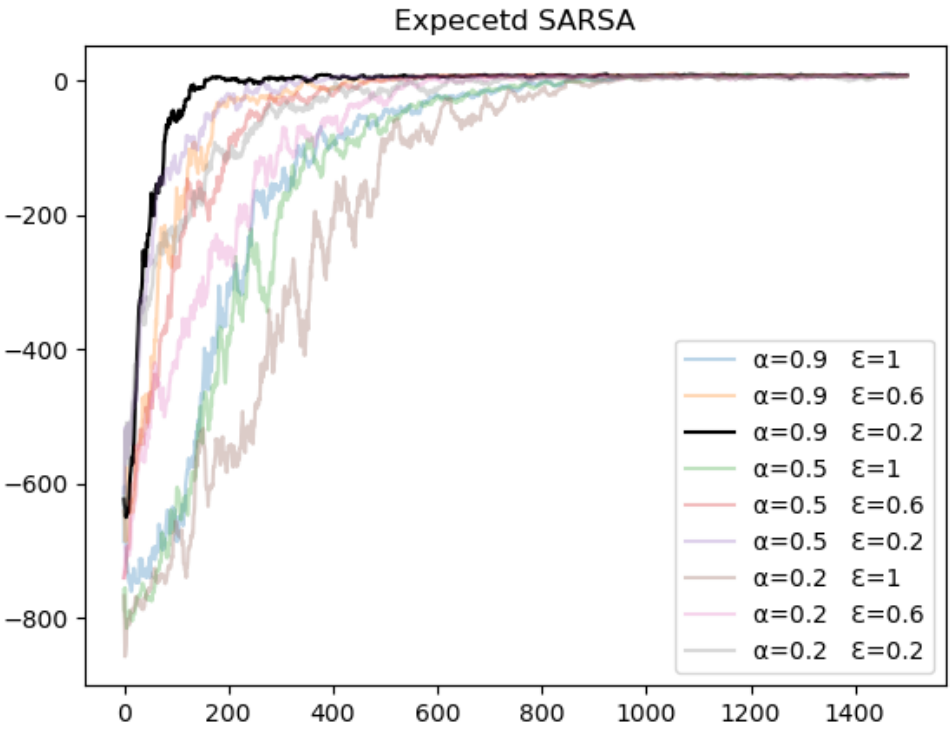

Figura 21. Alfa-épsilon. Expected SARSA en el entorno Taxi-v3.

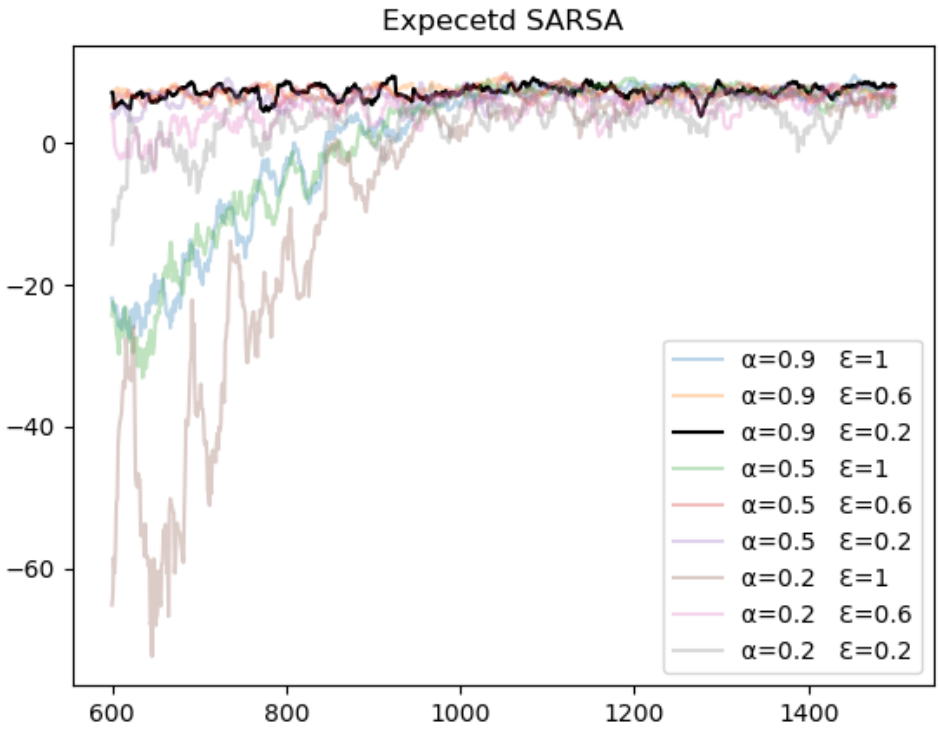

Figura 22. Figura 21 ampliada

Una vez se han tomado los valores óptimos de alfa y épsilon para cada algoritmo, se han de comparar entre ellos, eso es lo que muestra la figura 23.

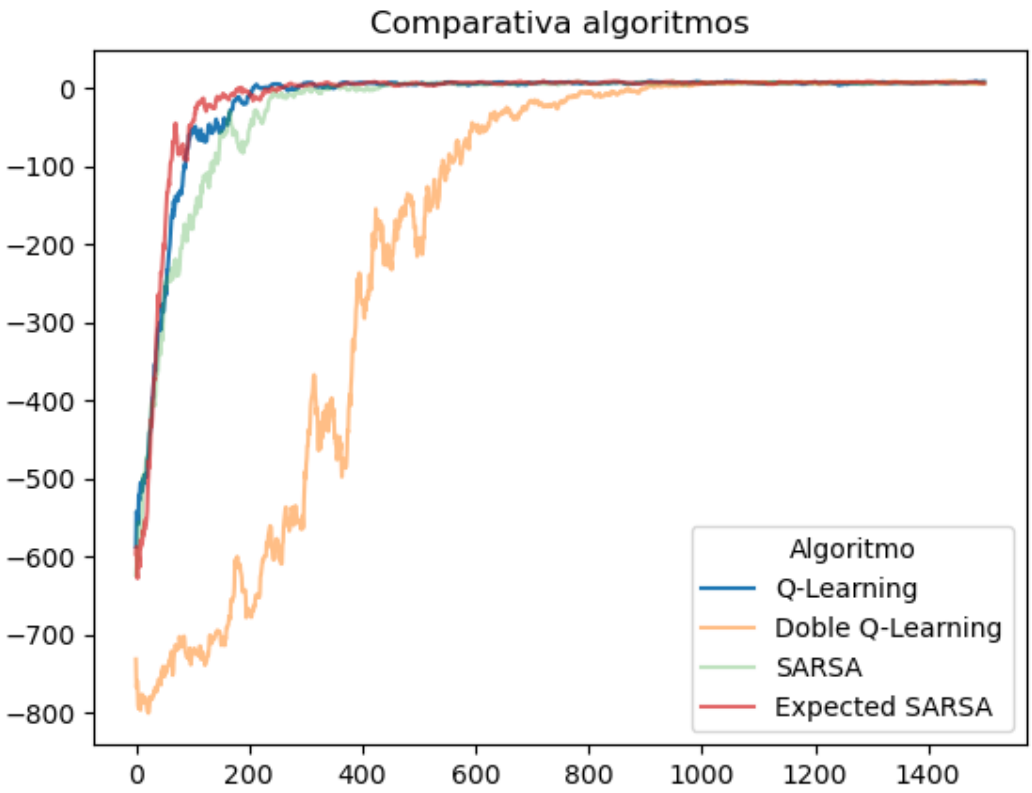

Figura 23. Comparativa de los 4 algoritmos en el entorno Taxi-v3.

En la figura anterior se puede observar como los algoritmos que mejor parecen funcionar son tanto Q-Learning como Expected SARSA, pero esto es en cuanto a resolver el problema se refiere, es decir, estos algoritmos pueden llegar a la solución de una forma más rápida, pero eso no significa que obtengan la puntuación más alta, puede ocurrir que a pesar de converger rápidamente, el valor al que converge sea menor que otro algoritmo. Esto no es el caso, y es que como se puede observar en la tabla 5, en la que se han plasmado las dos métricas descritas anteriormente en este capítulo, el algoritmo que obtiene resuelve de media el problema en un menor número de pasos y por ende obtiene una mayo recompensa es Expected SARSA, que a su vez es el que converge más rápido.

Algoritmo	Episodios hasta la convergencia	Recompensa
Q-Learning	349.5	2.17
Doble Q-Learning	2549.5	2.74
SARSA	1333.5	2.33
Expected SARSA	239	1.92

Tabla 5. Resultados de las dos métricas en el entorno Taxi-v3.

Tanto en el caso de Q-Learning como en el de Expected SARSA, ambos consiguen resolver el problema en muy pocos episodios. Siendo Expected SARSA el que obtiene mejores resultados, ya que consigue resolver el problema en una media de 239 episodios. Por otra parte, el algoritmo Expected SARSA no solo logra ser el que más rápido aprende, sino que además, es el que obtiene la recompensa más alta. Por lo que para este entorno, Expected SARSA es el algoritmo que mejores resultados consigue.

5.2 CartPole-v1

Para este entorno, igual que para el anterior, se ha utilizado las mismas métricas descritas en la sección 5.1 a fin de poder comparar los diferentes algoritmos.

1. Número de episodios que son necesarios para que el valor se estabilice.

2. Recompensa obtenida cuando se ha acabado el aprendizaje.

Además, este entorno difiere del anterior en que un estado está compuesto por variables continuas que pueden tomar los valores descritos en la tabla de la sección 4.2. Es decir, el entorno en cada acción, nos devuelve una observación compuesta por variables (posición del carro, velocidad del carro, ángulo del poste y velocidad del poste en la punta) que toman valores continuos, a diferencia del entorno anterior en el que cada estado estaba bien representado por un valor discreto, tablero, posición del taxi, posición del pasajero y destino, todas estas son variables discretas.

Por lo tanto en este entorno al ser los estados continuos, o lo que es lo mismo, infinitos, implica que la Tabla estados-acción, sea a su vez infinita, y dado que un ordenador no dispone de recursos infinitos, esto es inabarcable, por lo que para poder afrontar este problema se han discretizado los estados, es decir, para cada variable de la observación, dado su valor continuo, se discretiza para así poder formar la Tabla estado-acción y poder aplicar los algoritmos.

De igual manera que en el entorno anterior, antes de comparar los diferentes algoritmos entre sí, es necesario ver que parámetros alfa y épsilon se ajustan mejor a cada algoritmo.

Q-Learning. Tal y como se muestra en la figura 24, el agente obtiene mejores resultados durante el entrenamiento para los valores de **alpha = 0.9** y **épsilon = 0.2**

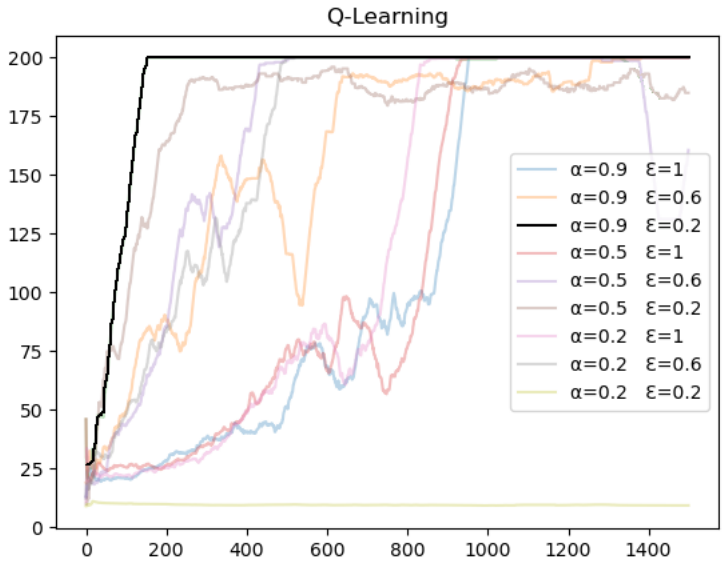

Figura 24. Alfa-épsilon. Q-Learning en el entorno CartPole-v0.

Doble Q-Learning. Para este algoritmo se puede observar de forma clara en la figura 25 como cuando **alfa = 0.5** y **épsilon = 0.6** el agente logra aprender de forma rápida convergiendo y maximizando la recompensa obtenida.

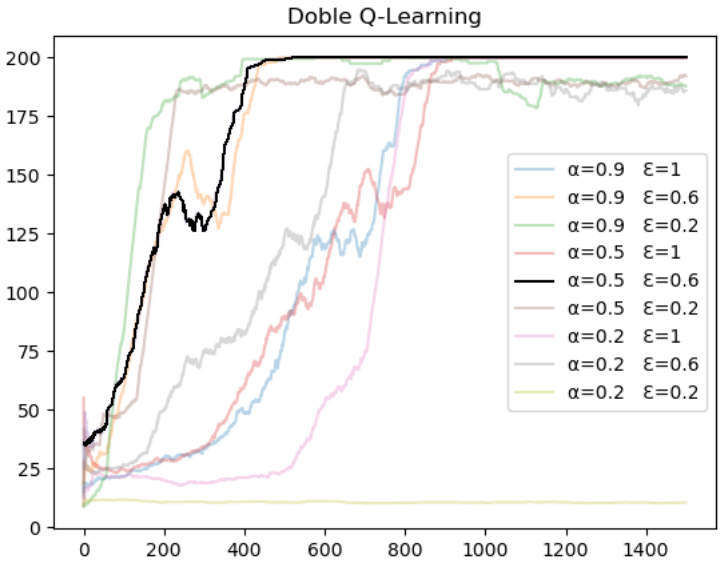

Figura 25. Alfa-épsilon. Doble Q-Learning en el entorno CartPole-v0.

SARSA. Para este algoritmo lo que mejor resultados da es estableciendo **alfa = 0.5** y **épsilon 0.6**, este comportamiento se puede observar en la figura 26.

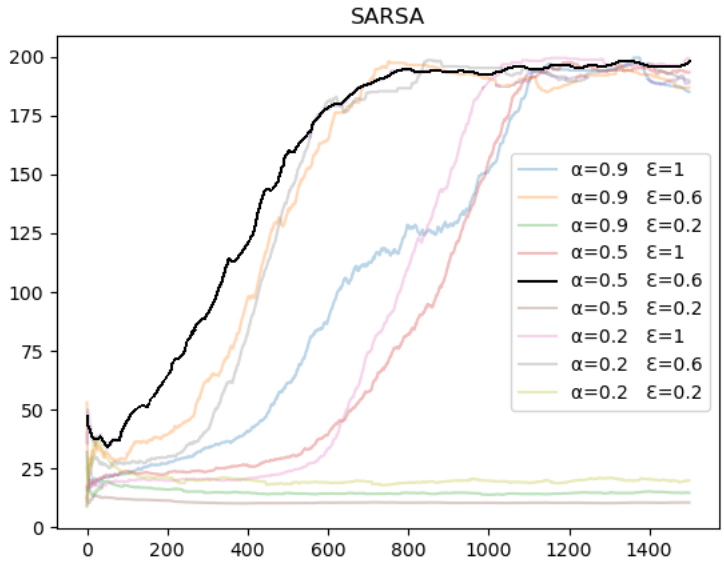

Figura 26. Alfa-épsilon. SARSA en el entorno CartPole-v0.

Expected SARSA. En este algoritmo se puede observar en la figura 27 como para los pares de valores alfa = 0.5, épsilon = 0.6 y alfa = 0.9, épsilon 0.6 el agente logra aprender de una forma más rápida, sin embargo lo que mejor resultado da es **alfa = 0.9** y **épsilon 0.9**, ya que converge obteniendo la recompensa máxima.

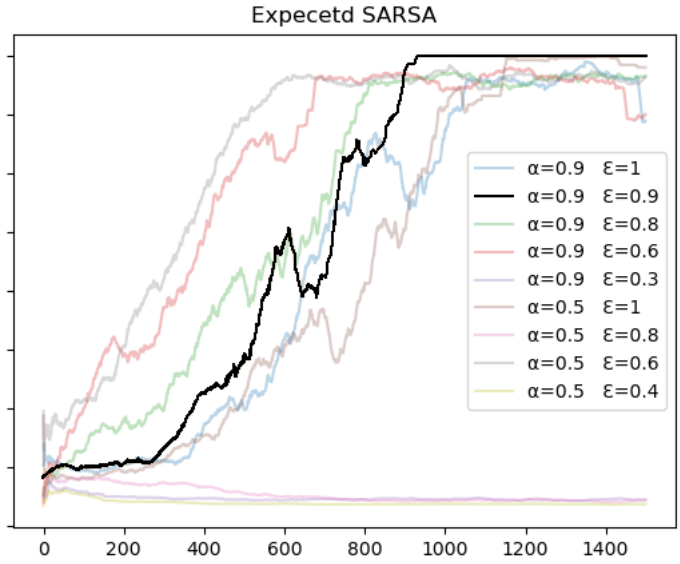

Figura 27. Alfa-épsilon. Expected SARSA en el entorno CartPole-v0.

Una vez tomado los valores correctos para alfa y épsilon en cada algoritmo, se ha procedido a hacer una comparación entre estos. Los resultados obtenidos se pueden observar en la figura 28.

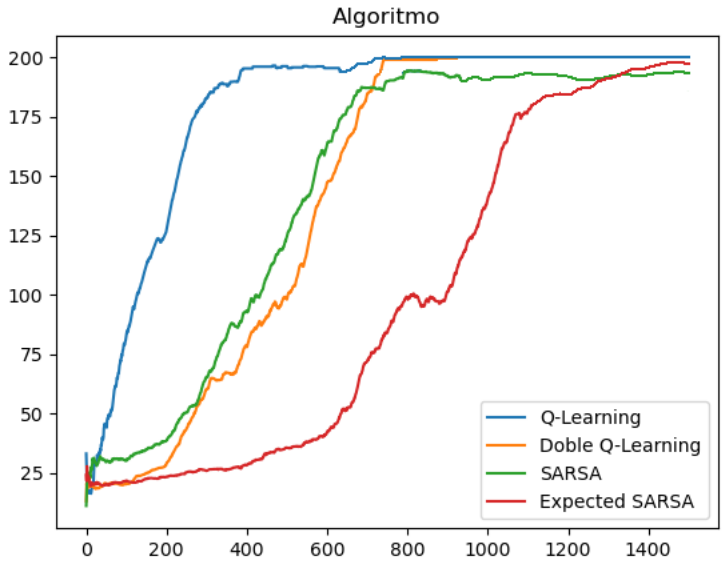

Figura 28. Comparativa de los 4 algoritmos en el entorno CartPole-v0.

En la figura anterior, se puede observar claramente como el algoritmo Q-Learning converge de una manera mucho más rápido que los demás, el algoritmo Doble Q-Learning también logra maximizar la recompensa, sin embargo tanto SARSA como expected SARSA, a pesar de obtener buenos resultados, se puede observar como no logran conseguir de forma constante la máxima recompensa.

En la siguiente tabla se muestra los resultados obtenidos de las dos métricas descritas al comienzo de la sección.

Algoritmo	Episodios hasta la convergencia	Recompensa
Q-Learning	411	199.97
Doble Q-Learning	580	199.86
SARSA	1012	185.32
Expected SARSA	1237	183.33

Tabla 6. Resultados de las dos métricas en el entorno CartPole-v0.

En la tabla 6 se puede observar lo descrito anteriormente, que tanto el algoritmo Q-Learning como Doble Q-Learning, logran converger a la máxima recompensa posible, ambos están muy cerca de 200. Sin embargo tanto SARSA como Expected SARSA a pesar de tener una recompensa alta, no están tan cerca de la máxima recompensa como Q-Learning y Doble Q-Learning. Además, dado que el algoritmo Q-learning consigue converger en un menor número de episodios con una diferencia notable, es por todo esto que este algoritmo es considerado el mejor para este entorno.

5.3 MountainCar-v0

Para poder medir la eficiencia de los algoritmos en este entorno, igual que para los anteriores, se utilizarán las métricas descritas ya anteriormente:

1. Número de episodios que son necesarios para que el valor se estabilice.

2. Recompensa obtenida cuando se ha acabado el aprendizaje.

De igual manera que con el entorno CartPole, este entorno nos provee de una observación o estado compuesta por variables continuas, es decir, debemos discretizarlas antes de poder comenzar el entrenamiento.

También es necesario establecer cuando se da por resuelto el problema, y es que puede ocurrir que el agente llegue a la meta en un episodio aleatorio, y no lo haga en los siguientes, por lo que estos resultados no serían muy fiables. Por ello, se ha considerado resuelto el problema cuando el agente ha obtenido una recompensa superior a -300 en los últimos 100 episodios, o lo que es lo mismo cuando el agente haya llegado a la meta en menos de 300 pasos en los últimos 100 episodios.

Una cosa a destacar en este entorno y es que como bien se ha descrito en la sección 4.3, el único estímulo que recibe el agente es una recompensa de -1 hasta que llega a la meta, esto puede dificultar el aprendizaje, ya que si el agente no logra llegar a la meta, todas las acciones reciben la misma recompensa y el agente puede no ser capaz de optimizar la Tabla estados-acción, es por esto que este entorno puede resultar más difícil para el agente.

Q-Learning. Como se puede observar en la figura 29, los valores que consiguen una mejor recompensa es **alfa = 0.2** y **épsilon = 0.2**.

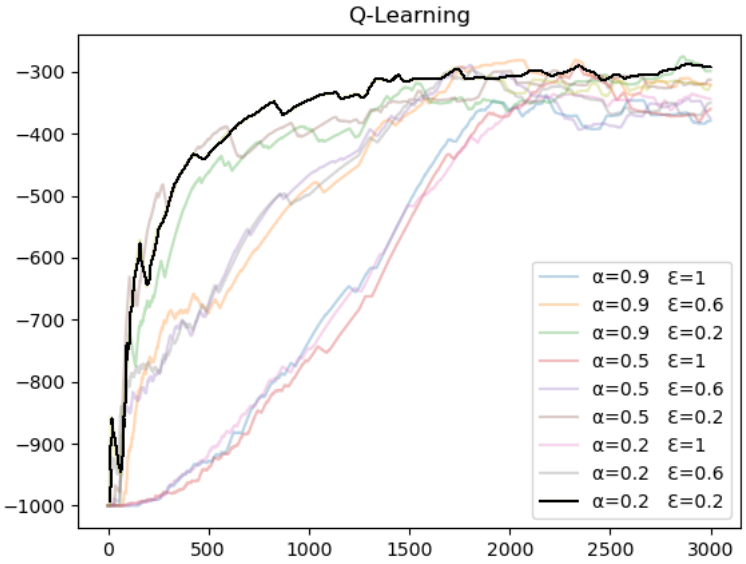

Figura 29. Alfa-épsilon. Q-Learning en el entorno MountainCar-v0.

Doble Q-Learning. El par de valores que hace que nuestro agente obtenga la recompensa más alta es para **alfa = 0.2** y **épsilon = 0.2**, esto se puede observar en la figura 30.

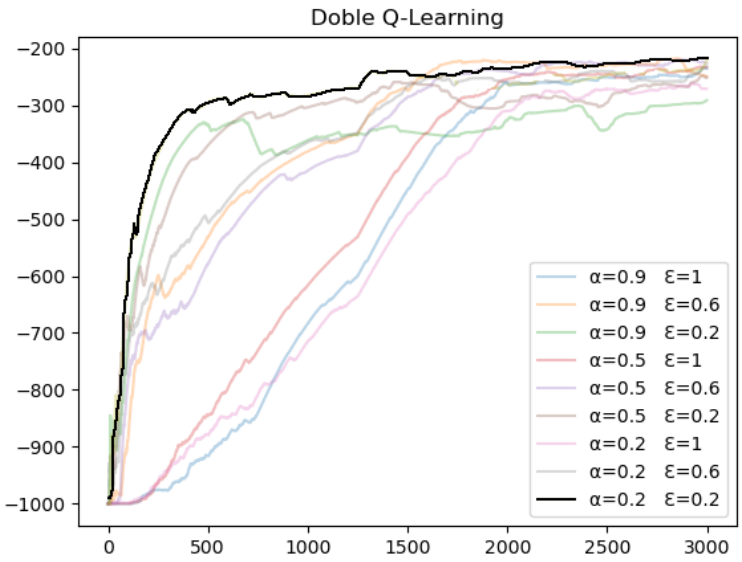

Figura 30. Alfa-épsilon. Doble Q-Learning en el entorno MountainCar-v0.

SARSA. Para este algoritmo el agente consigue los mejores resultados para **alfa = 0.5** y **épsilon = 0.5**.

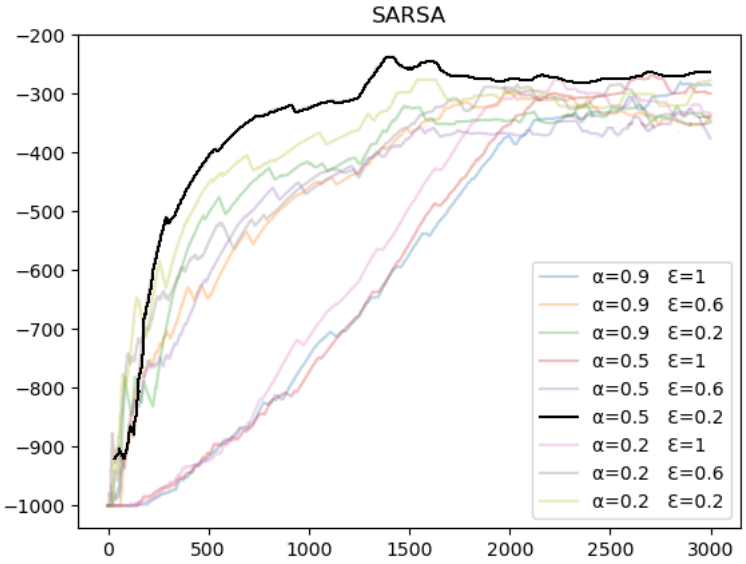

Figura 31. Alfa-épsilon. SARSA en el entorno MountainCar-v0.

Expected SARSA. El par de valores que hace que el agente obtenga mejores resultados es el par **alfa = 0.5** y **épsilon = 0.1** como se puede observar en la figura 32.

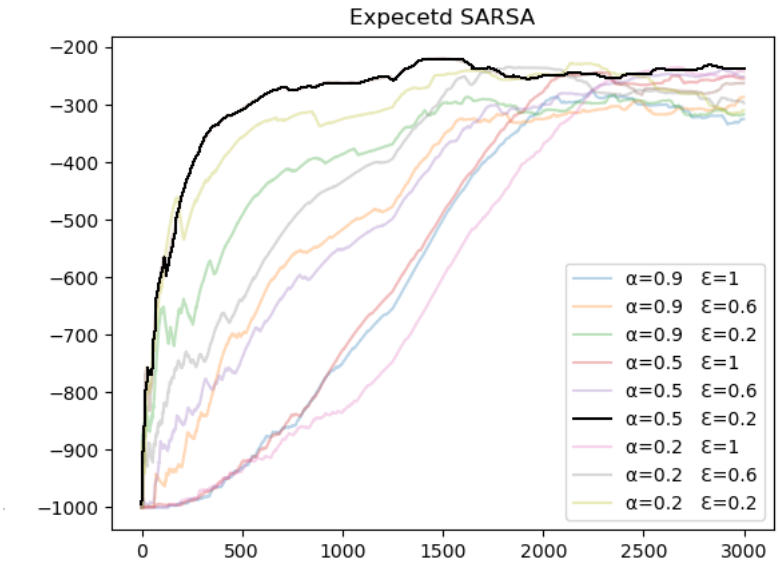

Figura 32. Alfa-épsilon. Expected SARSA en el entorno MountainCar-v0.

Una vez se han optado por los valores óptimos de alfa y épsilon para cada algoritmo. Es necesario hacer una comparativa de los 4 algoritmos usando estos valores de alfa y épsilon y comparar las 2 métricas para poder sacar conclusiones claras.

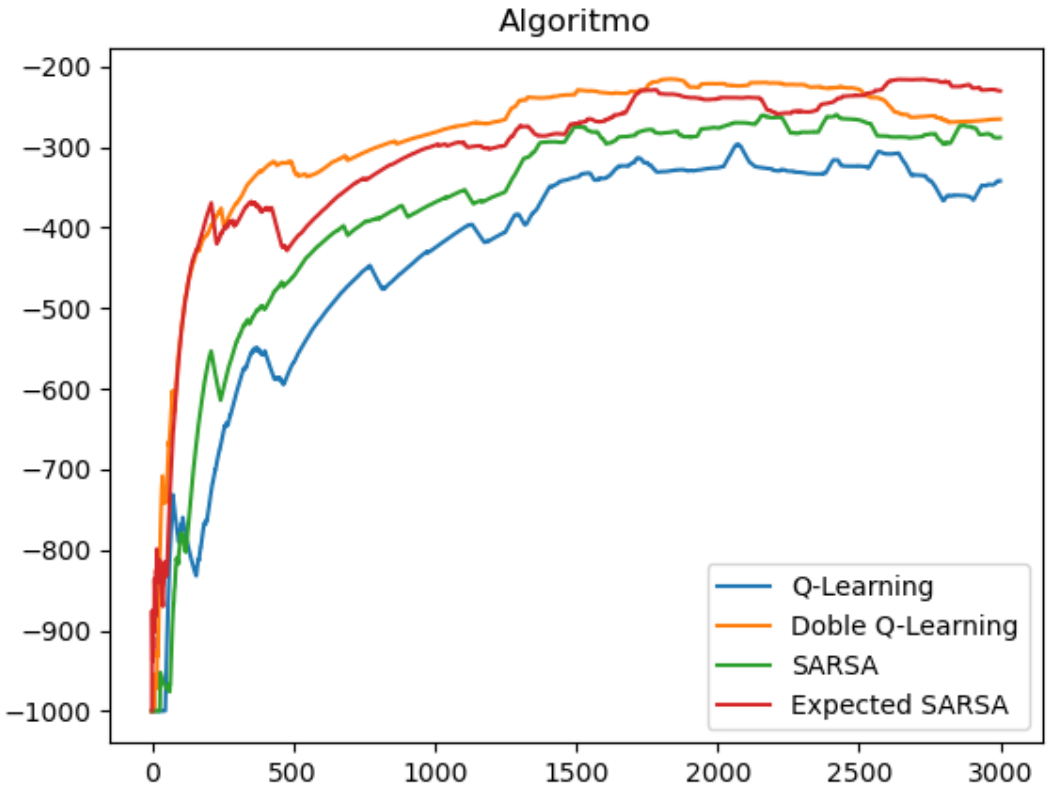

Figura 33. Comparativa de los 4 algoritmos en el entorno MountainCar-v0.

En la figura 33 se puede observar el rendimiento de los 4 algoritmos, siendo así el algoritmo Expected SARSA el que consigue mejores resultados, ya que es el que consigue una recompensa más alta, a su vez, también es el que logra aprender de una forma más rápida. Estos resultados se pueden observar en la siguiente tabla.

Algoritmo	Episodios hasta la convergencia	Recompensa
Q-Learning	1289	-276.67
Doble Q-Learning	633	-188.45
SARSA	956	-223.78
Expected SARSA	545	-158.12

Tabla 7. Resultados de las dos métricas en el entorno MountainCar-v0.

En la tabla 7 se observa los resultados obtenidos, y se ve como el algoritmo que consigue resolver el problema en un menor número de episodios es el algoritmo Expected SARSA, a su vez es el algoritmo que consigue resolver en el menor número de pasos y por ende, conseguir una recompensa más alta. Por lo que para este problema sin ninguna duda el mejor algoritmo es Expected SARSA.

5.4 Resultados finales

En este apartado se hará un resumen de las conclusiones sacadas tras la finalización de todas las pruebas presentes en este documento. Para ello, tabla se puede observar los resultados obtenidos en cada prueba, tanto el algoritmo que mejor resultado ha dado en cada entorno, como los valores Alfa-Épsilon empleados para cada algoritmo.

Entorno	Mejor Algoritmo	Alfa	Epsilon
Taxi-v3	Expected SARSA	0.9	0.2
CartPole-v1	Q-Learning	0.9	0.2
MountainCar-v0	Expected SARSA	0.5	0.2

Tabla 8. Resumen de resultados.

De la tabla anterior quizás se podría concluir que el mejor algoritmo es Expected SARSA, ya que ha resultado ser “ganador” en dos de los tres entornos, pero bajo mi criterio, estas pruebas no son totalmente concluyentes del todo y es que por ejemplo en el entorno Taxi-v3, a pesar de que el mejor algoritmo ha sido Expected SARSA, la diferencia con el segundo algoritmo (Q-Learning) no es demasiado grande, tal como se puede observar en la figura 23, ambos consiguen resultados muy similares, con esto quiero aclarar que no hay un algoritmo que se puede considerar el claro vencedor. Con los valores de Alfa y Épsilon ocurre lo mismo, como se puede observar en las gráficas donde se ha ido variando el valor de estos parámetros, se puede observar como la diferencia en los resultados a veces es mínima. Es por esto que considero que no hay un criterio claro a la hora de elegir cual es el mejor algoritmo.

6

Conclusiones

El objetivo principal del presente trabajo era el de comparar los 3 algoritmos descritos en el apartado 3 ajustando tanto el valor del parámetro Alfa y el valor del parámetro Épsilon para conseguir que el algoritmo sea lo más eficiente posible, y tras ello poder sacar conclusiones claras acerca del algoritmo que más se adecua a cada tipo de entorno. Para ello, en un primer momento se ha tenido que hacer un estudio previo de los diferentes algoritmos y conceptos relacionados con el aprendizaje por refuerzo, para poder tener la base y entender el problema con los conocimientos asentados. También se ha tenido que profundizar en el framework OpenAI Gym para poder abordar los diferentes entornos correctamente y no tener problema a la hora de programar los diferentes algoritmos. En base a los resultados obtenidos, se puede decir que la configuración de los parámetros es muy importante ya que según qué valores se escojan puede dar lugar a que el algoritmo no sea capaz de converger a buenos resultados.

Sin embargo, tal como se ha comentado anteriormente en el apartado 5.4, no es posible llegar a conclusiones totalmente claras acerca del mejor algoritmo, ya que en ocasiones los resultado han sido muy similares para los diferentes algoritmos aplicados. Por lo que para poder sacar estas conclusiones sería bueno extender el número de entornos y aplicar dichos algoritmos, quizás en futuras líneas de trabajo.

La realización de este trabajo de fin de grado ha sido de gran utilidad para complementar mis estudios. Aunque a lo largo de la carrera se dan asignaturas de inteligencia artificial, quizás no se profundiza demasiado, y con la realización de dicho trabajo he podido profundizar un poco más en este campo de la informática tan importante hoy día. Es de hecho la realización de este TFG lo que ha hecho que me decida a seguir mis estudios con un master en Data Science, campo que está estrechamente relacionado con la inteligencia artificial.

Referencias

[1] OpenAI, «OpenAI gym,» [En línea]. Available: https://gym.openai.com/.

[2] «Python,» [En línea]. Available: https://www.python.org/.

[3] «https://www.jetbrains.com/es-es/pycharm/,» [En línea].

[4] «https://es.wikipedia.org/wiki/Mind,» [En línea].

[5] Stanford University, «Artificial intelligence index,» 2019.

[6] H. v. Hasselt, «Double Q-learning,» 2010.

[7] R. S. S. y. A. G. Barto, Reinforcement Learning: An Introduction, 1998.