Q-learning para un mundo estocástico de dos dimensiones

Luis Carlos Mantilla Calderón

June 8, 2021

1 Resumen

En el ámbito de procesos de decisión de Markov (MPDs) se busca encontrar una política que maximice la recompensa recibida a lo largo de todo un proceso. Hay distintos métodos derivados de programación dinámica que permiten hallar soluciones a problemas de este tipo, entro ellos iteración por valor, iteración por política, inducción hacia atrás, Q-learning, entre otros. Este proyecto busca estudiar el algoritmo de *Q-learning*, y una modificación de este llamada *SARSA*, para encontrar una política óptima que un dron debe seguir para llegar a un objetivo dentro de un entorno estocástico. En la primera sección se introducen conceptos presentes en el campo de aprendizaje por refuerzo. Posteriormente, en la siguiente sección se introducen los algoritmos de q-learning y SARSA y se discute la convergencia de q-learning a una política óptima. Finalmente, el la última sección se realiza una implementación de ambos algoritmos para resolver el problema del taller 2 que se estudió en el curso, en donde un dron busca llegar a una zona sin entrar a otra prohibida, en el menor tiempo posible y dentro de un ambiente estocástico el cual simula una corriente de viento.

2 Aprendizaje por refuerzo

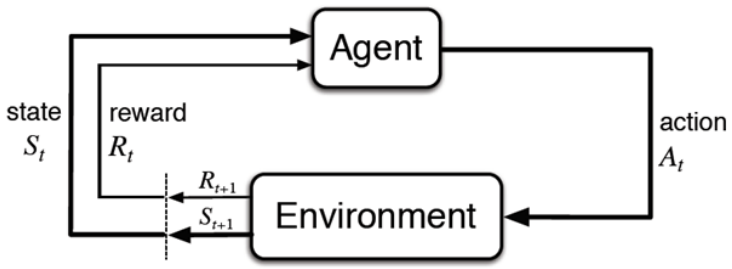

Figure 1: Interacción entre el ambiente y el agente. Tomado de [1].

El aprendizaje por refuerzo es un área del aprendizaje automático que involucra a un agente que puede estar en un conjunto de estados S y puede tomar algunas acciones A por estado [1]. El agente interactúa repetidamente con un entorno cuando realiza alguna acción y recibe una recompensa R_t como se muestra en la Figura 1, por lo que enfrenta un problema de toma de decisiones secuenciales (un problema multietapa). Para reducir la complejidad de estos problemas, en la práctica, se supone que el estado del agente solo depende de la observación. Matemáticamente, esto significa que las probabilidades de transición satisfacen la propiedad de Markov:

$$\mathbb{P}\left(r, s \mid S_t, A_t\right) = \mathbb{P}\left(r, s \mid \mathcal{H}_t, A_t\right).$$

Con esta condición, se pueden utilizar técnicas de programación dinámica para solucionar tales tareas. Resolver un problema de este tipo correspondería a obtener una política π tal que dé una distribución óptima de acciones a tomar para maximizar la recompensa

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$$

ganada por el agente. En comparación con otros paradigmas de aprendizaje automático, el aprendizaje por refuerzo considera escenarios en los que la retroalimentación de ciertas acciones puede retrasarse, ya que las decisiones tempranas pueden influir en las interacciones lejanas en el tiempo con el entorno.

Resolver estos problemas de procesos de decisión de Markov se puede hacer de varias maneras, por ejemplo, usando métodos de monte-carlo, usando temporal difference (TD) learning, o algoritmos de programación dinámica que hagan uso de un modelo del entorno (Figura 2). Generalmente en la práctica, cuando tratamos de usar aprendizaje por refuerzo para lograr una tarea en el mundo real, no se tiene acceso a un modelo explícito que nos diga las probabilidades de transición entre estados, lo cual limita usar métodos como iteración por valor y iteración por política. Hay varios métodos que no requieren un modelo del MDP que hacen parte de TD-learning, tales como Q-learning o SARSA.

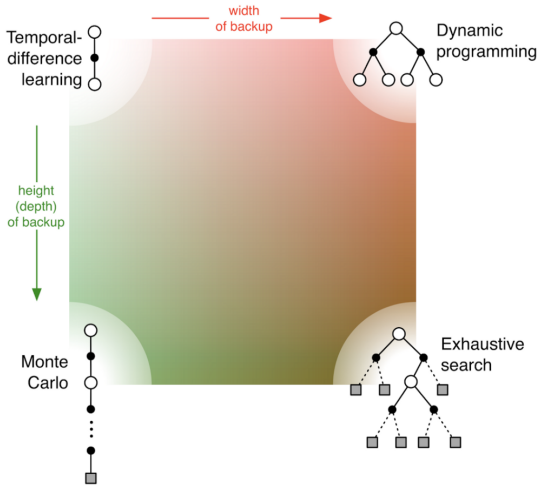

Figure 2: Una sección de los métodos de aprendizaje por refuerzo. Tomado de [1].

La esencia de los algoritmos de TD-learing está en mezclar la idea de hacer bootstraping como se hace en los métodos de programación dinámica y la idea de sampleo como se hace en métodos montecarlo. Un ejemplo sencillo de un método TD vendria siendo una actualización de la función de valor de la siguiente manera:

$$V(s) \leftarrow V(s) + \alpha(\overbrace{r + \gamma V(s')}^{\text{TD target}} - V(s)).$$

El sampleo viene de tomar una acción y un nuevo estado para actualizar la función V, y el bootstrap viene de usar la misma función $V(s_{t+1})$ (la cual es un estimador de $V_\pi(s_{t+1})$) para actualizar a V. Estos algoritmos son estimadores sesgados, contrario a los métodos usados por monte-carlo. Aún así, logran reproducir resultados significativamente buenos como lo veremos en la última sección.

3 Q-learning

Q-learning es un algoritmo de tipo TD que aborda el problema de encontrar las acciones óptimas que un agente debería realizar en entornos Markovianos para maximizar su recompensa sin tener a priori

un modelo explícito del entorno (es decir, sin saber el kernel de transición). El algoritmo encuentra la función de "calidad" $Q : S \times A \to \mathbb{R}$ de cada posible par estado-acción (s, a) dada por:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, \quad A_t = a] = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r_t\right]. \tag{1}$$

Si seguimos la política óptima $\pi^\star$ tendremos la función Q óptima, denotada por $Q^\star(s,a) = Q_{\pi^\star}(s,a)$. El objetivo de Q-learning es encontrar la función $Q^\star$. Una vez hallada, esta función nos permite elegir una política óptima pues, como es la recompensa esperada para un par estado-acción tenemos que $V^\star(s) = \max_a Q^\star(s,a)$, y así podemos reconstruir la política óptima $\pi^\star = \operatorname{argmax}_a Q^\star(s,a)$.

El algoritmo de **Q-learning** comienza inicializando la función Q de manera aleatoria y luego actualizan sus valores cada episodio t con la siguiente regla:

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha_t \left[r_{t+1} + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right]. \tag{2}$$

Aquí, $\alpha_t \in [0,1]$ se conoce como la *tasa de aprendizaje* y esta controla qué tan rápido cambia la función Q por paso. Adicionalmente, $\gamma \in [0,1]$ es el *factor de descuento* y controla qué tan rápido el agente da pasos importantes para maximizar la recompensa. El factor de descuento también se puede interpretar en el sentido de que el agente prioriza obtener una recompensa r en el tiempo t_0 en vez de obtener la misma recompensa r en un futuro $t_f > t_0$.

En la práctica, una forma de implementar este algoritmo es tomando α constante, y la forma como se elige el estado s_{t+1} a partir del estado s_t es usando una política ϵ-greedy derivada de Q_t. Estas políticas funcionan de la siguiente manera: generamos un numero aleatorio $p \in [0,1]$, si $p > \epsilon$ tomamos $a_t = \operatorname{argmax}_a Q(s_t, a)$, si no, tomamos a_t de manera aleatoria. Hay muchas otras maneras de implementar el algoritmo, como por ejemplo, tomar en cuenta los valores de $\{Q(s_t,a)\}_a$, y usarlos para obtener una distribución con la cual se elige a_t. En ciertos casos esto podría servir más usar otra política, pero en general las funciones ϵ-greedy sirven para actualizar la función Q. Es importante aclarar que si ϵ es constante, no vamos a tener una política óptima, pues siempre tendremos una probabilidad de explorar, y si quisiéramos llegar a una política óptima deberíamos hacer tender ϵ a 0.

Watkins mostró [2] que bajo ciertas condiciones, el algoritmo de q-learning (ecuación 2) converge a la función óptima $Q^\star$:

Teorema 1. *Dado que las recompensas estén acotadas $|r_t| \leq M$, las tasas de aprendizaje $\alpha_t \in [0,1)$, y además, para todo par de estado-acción (s,a)*

$$\sum_{i=1}^{\infty} \alpha_{n(i,s,a)} = \infty, \quad \sum_{i=1}^{\infty} (\alpha_{n(i,s,a)})^2 \leq \infty$$

con $n(i,s,a)$ el tiempo t donde el agente tomó por i-ésima vez la acción a en el estado s, entonces se tiene que con probabilidad 1

$$Q_n(X,a) \longrightarrow Q^\star(s,a), \\ n \longrightarrow \infty$$

donde $Q_n(s,a)$ es el valor de Q que se obtiene en la n-ésima iteración.

Adicionalmente, existe un algoritmo conocido como SARSA [1] que es una variación sutil del algoritmo de Q-learning. La diferencia es que SARSA usa la misma política de evolución para ajustar el valor de la función Q, contrario al algoritmo mencionado anteriormente, que usa la política dictada por $a_{t+1} = \operatorname{argmax}_a Q\left(s_{t+1}, a\right)$. El algoritmo **SARSA** viene dado por

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha_t \left[r_{t+1} + \gamma Q\left(s_{t+1}, a_{t+1}\right) - Q\left(s_t, a_t\right)\right], \tag{3}$$

donde a_{t+1} es una acción que se toma con la misma política usada para evolucionar al estado s_t. SARSA se conoce como un algoritmo *on-policy*, mientras que q-learning es *off-policy*. En algunos casos, SARSA puede ser mejor que el algoritmo de Q-learning como lo veremos en la sección de implementación.

4 Implementación

El problema que se estudia es aquel de un agente en un entorno estocástico dos-dimensional cuyo objetivo es llegar a una zona deseada evitando unas zonas prohibidas (como en el taller 2 del curso). La interpretación física del modelo se puede entender como un dron que busca hacer una entrega en un lado evitando obstáculos bajo la presencia de viento. En particular, se tomó un mapa de tamaño 15×15, recompensas de 0, 100, y -50 para los estados normales, la zona deseada y la zona prohibida, respectivamente. El kernel de transición viene dado por

$$Q\left((z,w) \mid ((x,y),u)\right) = \begin{cases} 0.3 & \text{si } z = x, w = y+1 \\ 0.4 & \text{si } z = x, w = y+2 \\ 0.2 & \text{si } z = x-1, w = y+2 \\ 0.1 & \text{si } z = x-1, w = y+1 \end{cases}$$

$$Q\left((z,w) \mid ((x,y),d)\right) = \begin{cases} 0.3 & \text{si } z = x, w = y \\ 0.3 & \text{si } z = x, w = y-1 \\ 0.2 & \text{si } z = x-1, w = y \\ 0.2 & \text{si } z = x-1, w = y-1 \end{cases}$$

$$Q\left((z,w) \mid ((x,y),l)\right) = \begin{cases} 0.3 & \text{si } z = x-1, w = y+1 \\ 0.2 & \text{si } z = x-1, w = y \\ 0.3 & \text{si } z = x-2, w = y \\ 0.2 & \text{si } z = x-2, w = y+1 \end{cases}$$

$$Q\left((z,w) \mid ((x,y),r)\right) = \begin{cases} 0.3 & \text{si } z = x+1, w = y \\ 0.4 & \text{si } z = x+1, w = y+1 \\ 0.2 & \text{si } z = x, w = y \\ 0.1 & \text{si } z = x, w = y+1, \end{cases}$$

y con la condición de frontera que si el agente hace una acción inválida se queda en el estado en el que estaba. El kernel de transición actúa como un viento que empuja al agente hacia la esquina derecha superior del mapa, así que para que el agente logre llegar a la zona de mayor recompensa, este debe explorar el mapa lo suficiente para que, a pesar de que el viento lo aleja de la zona de mayor recompensa, el la encuentre y actualice su función Q con el fin de llevarlo a esta en futuros episodios. Para implementar este problema se usó la librería de OpenAI [3]. El código que se hizo para este proyecto se puede encontrar en este link.

4.1 Sin barrera

Primero, se estudia el algoritmo de q-learning en un entorno donde no hay zonas prohibidas como se muestra en la Figura 3. Para poder cuantificar que tan preciso es este algoritmo, se resuelve el problema de MDP usando iteración por valor, un método de programación dinámica que usa el kernel de transición para hallar la política óptima, y se compara con la política que se obtiene cada episodio con la siguiente métrica:

$$d(\pi_1, \pi_2) = \sum_s \delta(\pi_1(s) - \pi_2(s)),$$

para dos políticas π_1, π_2 y δ el delta de dirac (básicamente, contar los lugares donde ambas políticas sean distintas).

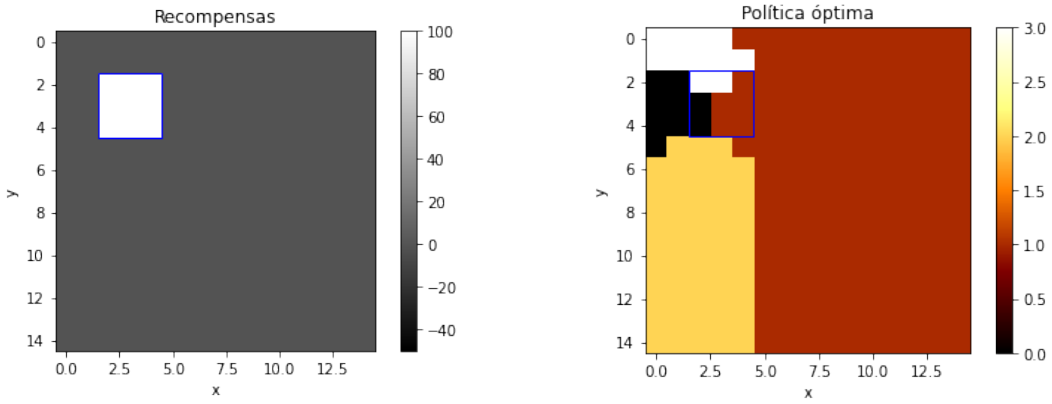

Figure 3: Recompensas de cada estado (x, y) en el que puede estar el agente y política óptima encontrada usando iteración por valor. Para el mapa de la política óptima, el color rojo indica moverse a la izquierda, el negro hacia la derecha, el amarillo hacia arriba, y el blanco hacia abajo.

Se implementa el algoritmo de q-learning (2) con una política ϵ-greedy derivada de Q para varios valores de ϵ iterando por $1,000,000$ episodios, usando una taza de aprendizaje $\alpha = 0.3$, y llegando a un estado terminal después de 30 pasos. Al comparar la política óptima mostrada en la Figura 3 con las políticas obtenidas usando este método se obtienen los resultados mostrados en la Figura 4. Podemos ver que los algoritmos con ϵ mayores logran reducir el error mucho más rápido. Hay un trade-off entre velocidad de convergencia dictada por ϵ y estabilidad a largo plazo, pero en este caso en donde solo hay una zona de recompensa y el mapa es sencillo, no es algo que se note en las curvas de error.

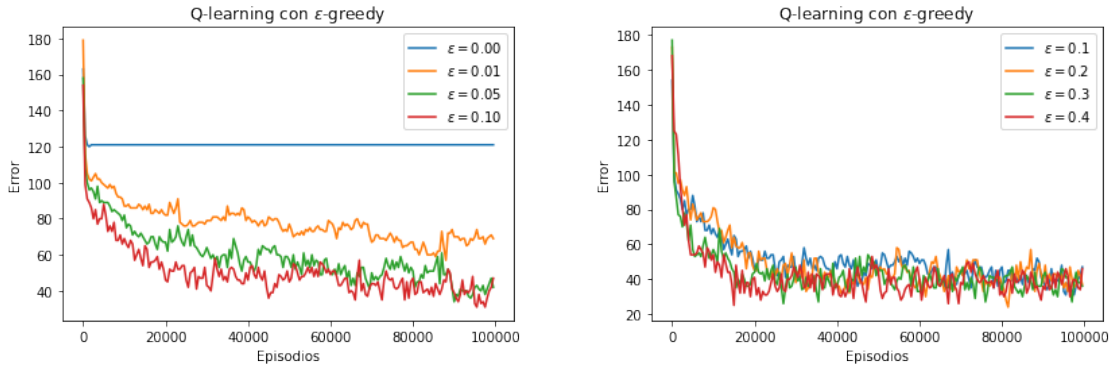

Figure 4: Error de las políticas encontradas usando q-learning con ϵ-greedy para varios valores de ϵ con respecto a la política óptima hallada usando iteración por valor para los primeros $100,000$ episodios del algoritmo.

También se implementa el algoritmo con la variación SARSA con las mismas políticas ϵ-greedy usadas para el algoritmo de q-learning inicial. El error que se obtiene para los mismos valores de épsilon se ve en la Figura 5. En este caso, las curvas de error se comportan de manera muy similar al algoritmo anterior. Comparando el error para ambos algoritmos (Figura 6) vemos que el algoritmo de SARSA es un poco mejor al inicio, en los primeros $10,000$ pasos, pero después de los $30,000$ pasos, la política obtenida usando ambos algoritmos es muy similar.

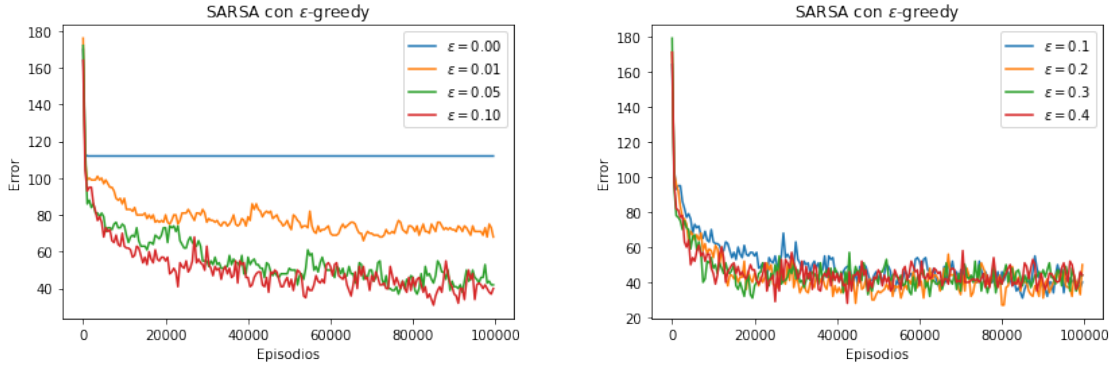

Figure 5: Error de las políticas encontradas usando SARSA con ϵ-greedy para varios valores de ϵ con respecto a la política óptima hallada usando iteración por valor para los primeros $100,000$ episodios del algoritmo.

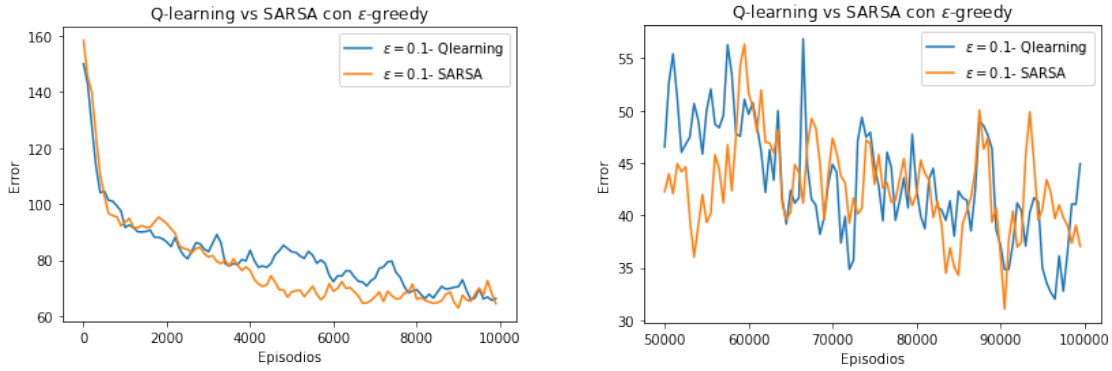

Figure 6: Comparación del error para el algoritmo SARSA con ϵ-greedy y Q-learning con ϵ-greedy con $\epsilon = 0.1$.

4.2 Con barrera

Con el ejemplo anterior en mano, podemos estudiar el mismo problema pero colocando una zona prohibida, como lo muestra la Figura 7, la cual el agente deberá evitar. Esta tarea es un poco más complicada que la anterior ya que el agente deberá evitar la zona prohibida, debe explorar lo suficiente para que aprenda a rodear la barrera (yendo en contra del viento), y debe alejarse de la barrera para que el viento no lo empuje hacia ella. En este tipo de situaciones es cuando el aprendizaje por refuerzo sobresale, ya que las decisiones tomadas por el dron al inicio tienen consecuencias importantes en la recompensa total, particularmente, es preferible que se demore en llegar a la recompensa con el fin de evitar la barrera prohibida, y así la recompensa esperada será mayor. Nuevamente se resuelve el MPD con iteración por valor para obtener la política óptima para compararla con las políticas obtenidas por q-learning y SARSA después de $1,000,000$ de iteraciones y un valor de $\alpha = 0.3$.

Las curvas de error que se obtienen usando q-learning con política ϵ-greedy que se muestran en la Figura 8 tienen un comportamiento distinto en este mapa en comparación con el anterior. Vemos que mayor ϵ no implica menor error al inicio, pues esto lleva al agente a explorar mucho y vuelve inestable la actualización de la función Q. También se ve que, después de varias iteraciones, para al rededor de 80 estados se tiene una política errónea, mientras que en el mapa anterior, era al rededor de la mitad. Esto muestra que los resultados obtenidos para $1,000,000$ de iteraciones con este algoritmo y para casi

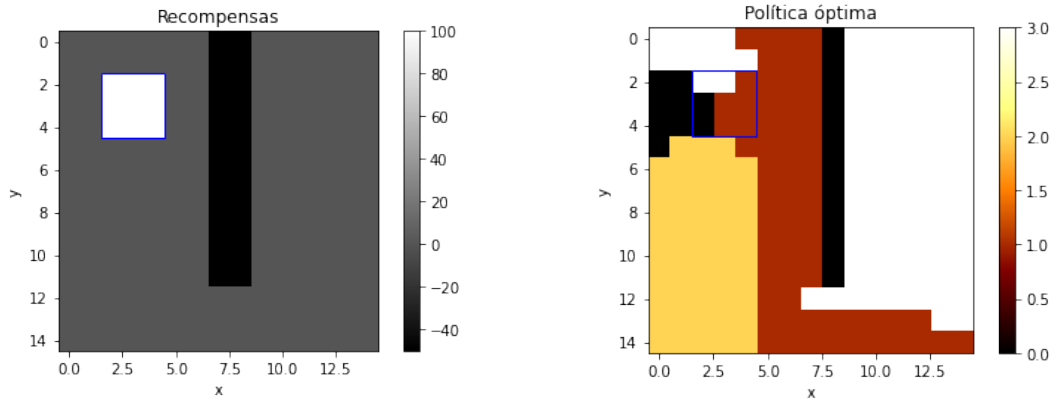

Figure 7: Recompensas de cada estado (x, y) en el que puede estar el agente y política óptima encontrada usando iteración por valor.

todos los valores de ϵ usados, se alejan bastante de la política óptima.

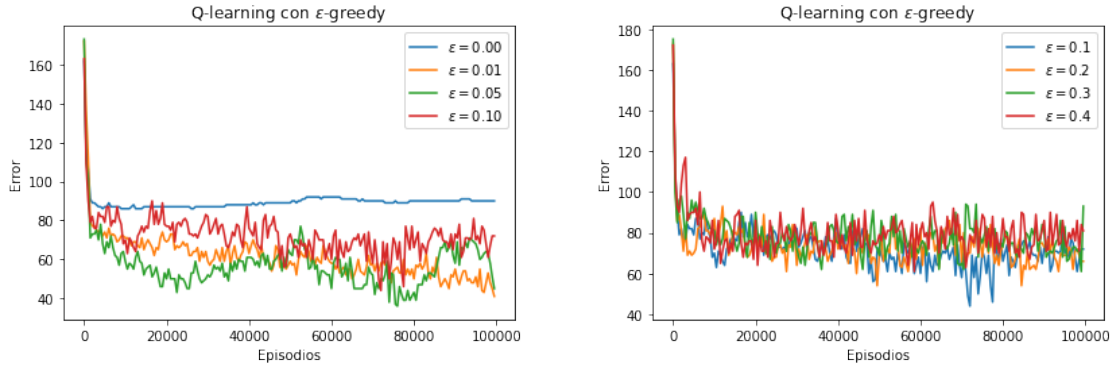

Figure 8: Error de las políticas encontradas usando q-learning con ϵ-greedy para varios valores de ϵ con respecto a la política óptima hallada usando iteración por valor para los primeros $100,000$ episodios del algoritmo.

Ahora bien, usando el algoritmo de SARSA se obtienen mejores resultados como lo muestra la Figura 9. Aquí podemos ver que para ciertos valores de épsilon, el error viene siendo grande y después de cierto número de episodios, el error baja drásticamente. Esto se puede entender como el agente aprendiendo que la mejor opción para aumentar la recompensa esperada es yendo alrededor del obstáculo en vez de pasar por el. Este comportamiento se ve para valores de $\epsilon = 0.1$ y $\epsilon = 0.2$, pero no ocurre para $\epsilon = 0.3$ y $\epsilon = 0.4$. Esto ya que al tener un ϵ tan alto, como mencionamos anteriormente, el agente explora mucho el mapa volviendo inestable el método.

Si comparamos ambos métodos con un valor de $\epsilon = 0.1$, vemos que SARSA inicia con un error muy grande en comparación con aquel de q-learning, pero al rededor de los $30,000$ pasos logra reducir su error de manera significativa (Figura 10). Esto ocurre ya que con SARSA el agente aprende a rodear el obstáculo, y la mayoría del error venia de tomar la acción incorrecta cuando se estaba cerca de la barrera. Viendo la política que se obtiene en la última iteración para q-learning y SARSA (Figura 11), vemos que q-learning no logra identificar que lo óptimo es rodear la barrera, mientras que SARSA si lo hace.

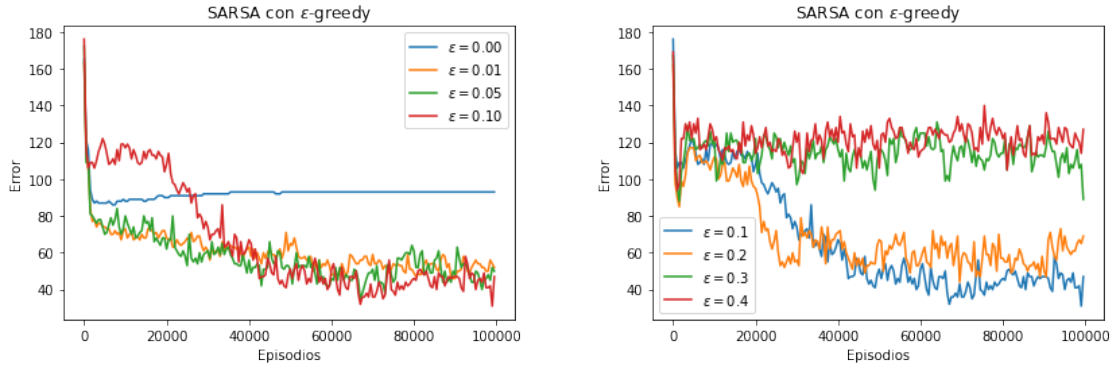

Figure 9: Error de las políticas encontradas usando SARSA con ϵ-greedy para varios valores de ϵ con respecto a la política óptima hallada usando iteración por valor para los primeros $100,000$ episodios del algoritmo.

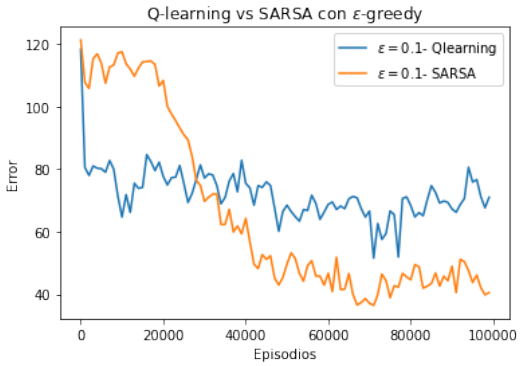

Figure 10: Comparación del error entre el algoritmo SARSA y Q-learning para el mapa de la Figura 7.

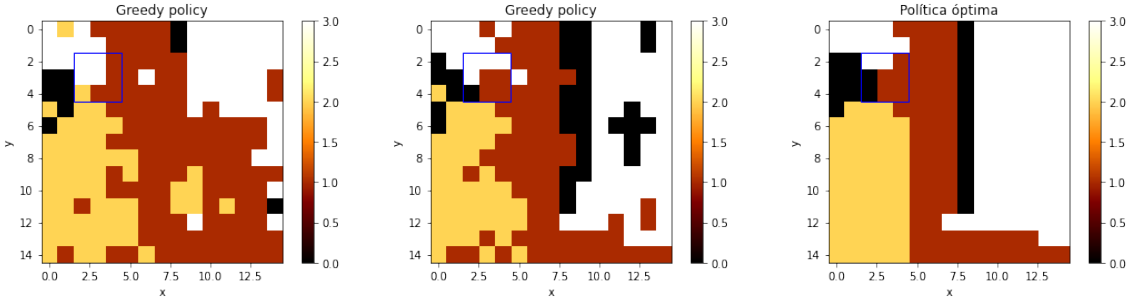

Figure 11: Políticas obtenidas usando Q-learning ($\epsilon = 0.1$), SARSA ($\epsilon = 0.1$), e iteración por valor.

5 Conclusiones

En este proyecto se estudió la aplicación dos algoritmos de diferencia temporal, q-learning y SARSA, con el fin de resolver un problema de decisiones secuenciales en un entorno Markoviano de dos dimensiones el cual busca modelar el comportamiento de un dron bajo la influencia de una corriente de viento que lo arrastra de manera aleatoria hacia una dirección especifica. Se comparan, para dos escenarios de recompensas distintos, las políticas obtenidas usando estos dos algoritmos, que no tienen conocimiento del kernel de transición, con la política óptima hallada usando iteración por valor (la cual si hace uso de dicho kernel). Primero, se estudia un mapa en el cual hay únicamente una región con una recompensa alta ubicada contracorriente a donde el agente debería ir. Se encuentra que ambos métodos se comportan de manera similar y logran que el agente cumpla con esta meta. Luego, se estudió un mapa con la misma región anterior, pero ahora con una región con recompensa negativa que está bloqueando parcialmente el paso a la zona de recompensa alta. En este caso, se halla que el algoritmo SARSA (on-policy) logra un desempeño significativamente mejor en comparación al algoritmo de Q-learning (off-policy).

References

[1] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction", en, 352.

[2] C. J. Watkins and P. Dayan, "Q-learning", Machine learning **8**, 279–292 (1992).

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym", arXiv preprint arXiv:1606.01540 (2016).

[4] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming* (John Wiley & Sons, 2014).

[5] J. N. Tsitsiklis, "Asynchronous stochastic approximation and q-learning", Machine learning **16**, 185–202 (1994).

[6] C. L. Beck and R. Srikant, "Error bounds for constant step-size q-learning", Systems & control letters **61**, 1203–1208 (2012).