Aprendizaje Por Refuerzo: Acrobot

Beatriz Torreiro Mosquera

***Resumen*—El objetivo principal de este proyecto es la construcción de un modelo de aprendizaje por refuerzo que sea capaz de resolver el entorno Acrobot de OpenAI, [1]. A lo largo de este documento se utilizan los algortimos Q-Learning, Double Q-Learning y Dueling Q-Learning, a través de su implementación en Python, para resolver dicho problema.**

I. Introducción

El aprendizaje por refuerzo es uno de los campos del aprendizaje automático. El aprendizaje por refuerzo permite entrenar modelos para que aprendan a reaccionar ante un entorno determinado. Uno de los algoritmos más famosos en el campo del aprendizaje por refuerzo es el Q-learning, que se basa en un concepto de aprendizaje del valor Q que mide cómo de buena es la acción en un estado dado.

En este proyecto, se experimenta con múltiples técnicas de aprendizaje por refuerzo que incluyen Q-learning, Double Deep Q-learning y Dueling Deep Q- learning. Se entrenan y evalúan dichos algoritmos en el entorno Acrobot de OpenAI,[1].

II. Descripción del contexto de trabajo

Acrobot es un péndulo de dos enlaces con solo la segunda articulación activada, Figura 1. Inicialmente, ambos enlaces apuntan hacia abajo. El objetivo es balancear el enlace inferior a una altura de al menos la longitud de un enlace por encima de la base. Ambos enlaces pueden girar libremente y pueden pasar uno al lado del otro, es decir, no chocan cuando tienen el mismo ángulo,[2].

El estado está formado por el seno y el coseno de los ángulos de las dos articulaciones rotacionales y las velocidades angulares articulares:

$$[cos(\theta 1) \quad sin(\theta 1) \quad cos(\theta 2) \quad sin(\theta 2) \quad \omega 1 \quad \omega 2] \qquad (1)$$

Para el primer enlace, un ángulo $\theta 1$ de 0^o corresponde al enlace que apunta hacia abajo. El ángulo del segundo enlace ($\theta 2$) es relativo al ángulo del primer enlace ($\theta 1$). Un ángulo $\theta 2$ de 0^o corresponde a tener el mismo ángulo entre los dos enlaces. Un estado de [1, 0, 1, 0, ..., ...] significa que ambos enlaces apuntan hacia abajo.

La acción es aplicar +1 ($+\tau$), 0 o -1($-\tau$) torque en la unión entre los dos enlaces pendulares.

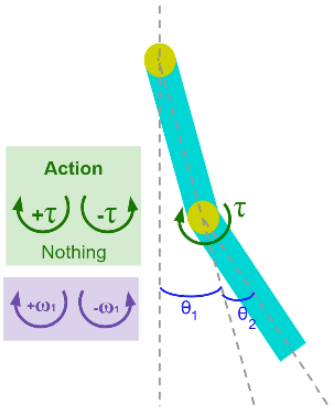

Figura 1: Entorno Acrobot

La siguiente imagen muestra un estado exitoso de Acrobot, Figura 2.

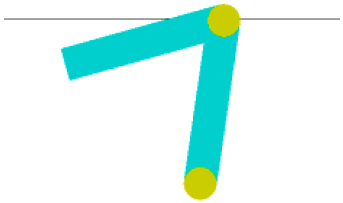

Figura 2: Acrobot alcanzando la meta

III. Aprendizaje por refuerzo

En el aprendizaje por refuerzo existen dos elementos fundamentales, el agente y el entorno. La misión del agente es aprender cómo reaccionar ante los diferentes estados del entorno y así maximizar la recompensa obtenida tras cada acción. Para cada acción realizada por un agente, el entorno responde devolviendo la información del siguiente estado y la recompensa inmediata de la acción realizada. Un entorno está formado por diversos estados, siendo al menos uno de ellos el estado final u objetivo, el estado al que desea llegar el agente a través de acciones. El conjunto de acciones que aprende a ejecutar un agente en función del estado en el que se encuentre se denomina política.

III-A. Algoritmo Q-Learning

El algoritmo Q-Learning es uno de los mayores descubrimientos en el campo del aprendizaje por refuerzo,

siendo uno de los algoritmos más simples pero más potentes. Aplica el concepto *off-policy*, el agente aprende una política o políticas diferentes de la que se está ejecutando. Es decir, el agente explora el entorno a la vez que explota lo que ya ha aprendido del mismo.

En cada iteración, el agente tiene en cuenta el estado futuro y observa la recompensa máxima posible de todas las acciones disponibles en ese estado, escogiendo la acción que le genere una mayor recompensa. Esta información es después utilizada para actualizar su política de acción.

La función Q-learning $Q(s, a)$ es una función que representa el valor de recompensa futura descontada, de tomar una acción a desde el estado s y continuar de manera óptima a partir de ese momento. Representa la puntuación máxima posible tras llegar al estado final del entorno, después de realizar una determinada acción sobre el estado actual.

El algoritmo en detalle es el siguiente:

Algorithm 1: Q-LEARNING

Intialize $Q(s, a)$, $\forall s \in A(s)$, arbitrarily, and $Q(terminal - state, .) = 0$
Repeat (for each episode)
  Initialize $\mathcal{S}$
  Repeat (for each step of episode):
    Choose A from S using policy derived from $Q(e.g, \epsilon - greedy)$
    Take action A, observe R , S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  until S is terminal

En la implementación anterior, α representa la tasa de aprendizaje $(0 < \alpha < 1)$ y γ representa el factor de descuento, si $\gamma = 0$ significa que sólo se tienen en cuenta las recompensas más inmediatas, mientras que si $\gamma = 1$, el agente considera con mayor fuerza las recompensas futuras.

III-B. Algoritmo Double Q-Learning

Q-Learning tiene un rendimiento muy bajo en algunos entornos debido a su gran sobreestimación de la recompensa causada por el uso de $max(Q(s', a))$. Para resolver este problema, Hado Van Hasselt propuso el método Double Q-Learning.

La solución propuesta consiste en mantener dos matrices de recompensas QA y QB, cada una recibe una actualización de la otra para el siguiente estado. La actualización consiste en encontrar la acción$a*$ que maximiza QA en el siguiente estado $(Q(s', a*) = MaxQ(s', a))$, y luego se usa $a*$ para obtener el valor de $QB(s', a*)$ para actualizar $QA(s, a)$. El algoritmo actualiza QA y QB de manera equiprobable.

El algoritmo en detalle es el siguiente:

Algorithm 2: DOUBLE Q-LEARNING

Intialize Q^A, Q^B, s
Repeat (for each episode)
  Choose a, based on $Q^A(s, .)$ and $Q^B(s, .)$, observe r, s'
  Choose either UPDATE(A) or UPDATE(B)
  if UPDATE(A) then
    Define $a* = arg \quad max_a Q^A(s', a)$
    $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a)(r + \gamma Q^B(s', a*) - Q^A(s, a))$
  else if UPDATE(A) then
    Define $b* = arg \quad max_a Q^B(s', a)$
    $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a)(r + \gamma Q^A(s', a*) - Q^B(s, a))$
  end if
  $s \leftarrow s'$
until end

En la implementación anterior, α representa la tasa de aprendizaje $(0 < \alpha < 1)$ y γ representa el factor de descuento, si $\gamma = 0$ significa que sólo se tienen en cuenta las recompensas más inmediatas, mientras que si $\gamma = 1$, el agente considera con mayor fuerza las recompensas futuras.

III-C. Algoritmo Dueling Deep Q-Learning

El algoritmo Dueling Deep Q-Learning presenta un cambio en la arquitectura del algortimo Deep Q-Learning. La nueva arquitectura tiene el siguiente aspecto, Figura 3 [3].

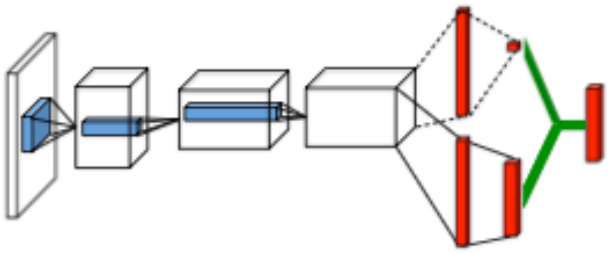

Figura 3: Arquitectura Dueling Deep Q-Learning

Dueling DQN usa un Dueling Q Head especializado para separar la matriz de recompensas Q en un vector A denominado ventaja y un vector V denominado valor. Agregar este tipo de estructura en la red permite que esta diferencie mejor las acciones entre sí y mejora significativamente el aprendizaje.

El cálculo de la matriz de recompensa viene determinado por la siguiente ecuación, [3]:

$$Q(s, a, \theta, \alpha, \beta) = V(s, \theta, \beta) + (A(s, a, \theta, \alpha) - max_{a' \in |A|} A(s, a', \theta, \alpha)) \quad (2)$$

En la ecuación anterior, θ representa los parámetros de configuración de la red neuronal mientras que α y β representan parámetros de los dos vectores V y A que está interconectados.

IV. Implementación

A la hora de implementar cualquiera de los algoritmos expuestos, ha sido necesario hacer mano de una función que determine la altura a la que se encuentra en enlace inferior de Acrobot, ya que no es un dato que venga dado por el estado. La fórmula empleada para el cálculo de la altura ha sido la siguiente:

$$altura = -cos(\theta 1) - (cos(\theta 1)cos(\theta 2) - sin(\theta 1)sin(\theta 2)) \tag{3}$$

Además, la altura se ha utilizado como medida de la recompensa.

IV-A. Algoritmo Q-Learn

La naturaleza del entorno hace muy compleja la implementación del algoritmo Q-Learn directamente, debido al gran número de estados que presenta. Por ese motivo, se ha implementado una variación de dicho algoritmo a través de Deep Q-Networks (DQN).

En vez de calcular directamente el valor de la función $Q(s, a)$, se calcula una función aproximada $Q(s, a, \theta)$. La red neuronal tiene que aprender una jerarquía no lineal de características que proporcionan estimaciones precisas del valor Q. Esta red tiene una salida separada para cada acción posible, que proporciona el valor estimado de Q para esa acción dado el estado de entrada. Además, la red neuronal se entrena utilizando actualizaciones de gradiente estocástico en mini lotes y reproducción de experiencia, y utiliza la política *epsilon-greedy* para seleccionar las acciones.

Los parámetros empleados en la implementación son los siguientes:

- $\epsilon = 0{,}3$: Probabilidad inicial de que se escoja una acción aleatoria en vez de la determinada por la política
- $\epsilon_{decay} = 0{,}995$: Factor por el que se multiplica a ϵ cada vez que el entorno es resuelto
- $\gamma = 0{,}99$: Factor de descuento que tiene muy en cuenta recompensas futuras
- $\alpha = 0{,}01$: Tasa de aprendizaje
- $steps = 300$: Número de acciones que se pueden ejecutar antes de dar por finalizado un episodio
- $episodes = 1000$: Número de veces que se intenta resolver el entorno
- La red neuronal utilizada para estimar la matriz de recompensas está compuesta por dos capas lineales, ya que el entorno no es excesivamente complejo

Tras la ejecución del modelo se obtienen los siguientes resultados:

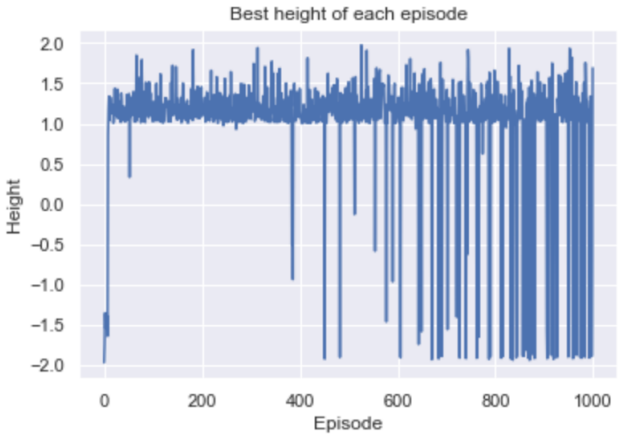

Figura 4: Mejor altura conseguida en cada iteración, Q-Learning

Figura 5: Recompensas obtenidas en cada iteración, Q-Learning

En la Figura 4, se puede observar como el modelo es capaz de conseguir el objetivo, es decir, alcanzar una altura igual o superior a uno, en tan sólo unas pocas iteraciones. Sin embargo, dicho gráfico es bastante ruidoso, lo que implica que una vez aprendida la política, no es capaz de alcanzar la meta de forma consistente. Este ruido también se puede observar en la Figura 5, que muestra el cúmulo de recompensas obtenido en cada episodio.

Con el objetivo de tener una visión más certera del rendimiento de este modelo, se ejecuta el modelo 100 veces y se comparan los resultados obtenidos, Figura 6.

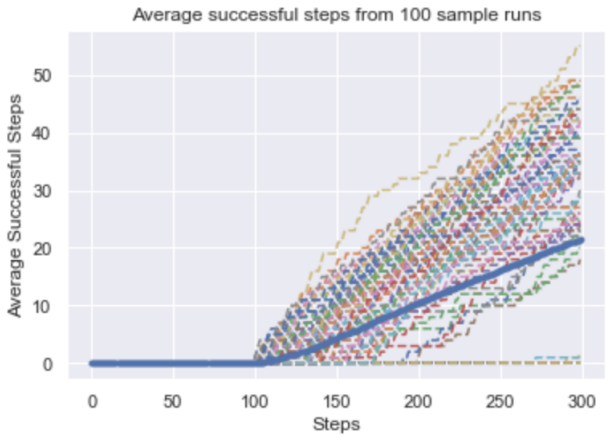

Figura 6: Rendimiento modelo Q-Learn

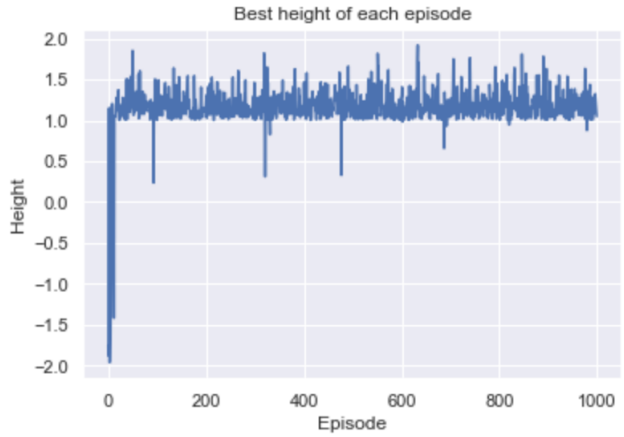

Figura 7: Mejor altura conseguida en cada iteración, Double Q-Learning

La imagen anterior, Figura 6, representa el número de pasos satisfactorios medio (recta azul), entendidos como aquellos momentos en los que el enlace inferior se encuentra a una altura de 1 o superior, a lo largo de cada episodio de 300 pasos. En este caso, de media, se dan $21{,}47$ pasos exitosos, que representan el $7{,}16\,\%$ de los pasos totales y el primer paso satisfactorio medio es el 100.

IV-B. Algoritmo Double Q-Learning

En este caso, debido a la naturaleza del entorno, se ha utilizado una red neuronal para estimar los valores de las matrices de recompensa Q^A y Q^B.

Los parámetros empleados en la implementación son los siguientes:

- $\epsilon = 0{,}3$: Probabilidad inicial de que se escoja una acción aleatoria en vez de la determinada por la política
- $\epsilon_{decay} = 0{,}995$: Factor por el que se multiplica a ϵ cada vez que el entorno es resuelto
- $\gamma = 0{,}99$: Factor de descuento que tiene muy en cuenta recompensas futuras
- $\alpha = 0{,}01$: Tasa de aprendizaje
- $steps = 300$: Número de acciones que se pueden ejecutar antes de dar por finalizado un episodio
- $episodes = 1000$: Número de veces que se intenta resolver el entorno
- $batch\_size = 128$: Tamaño de batch
- La red neuronal utilizada para estimar la matriz de recompensas está compuesta por una capa lineal.

En la Figura 7, se puede observar como el modelo es capaz de conseguir el objetivo en tan sólo unas pocas iteraciones. Además, una vez alcanzado el primer episodio exitoso, la gran mayoría de los siguientes episodios son exitosos, es decir, es mejor modelo que el obtenido a través de Q-Learning si comparamos la Figura 7 con la Figura 4, al ser menos ruidoso. Esta disminución de ruido también se puede apreciar en las recompensas totales obtenidas en cada episodio, Figura 8.

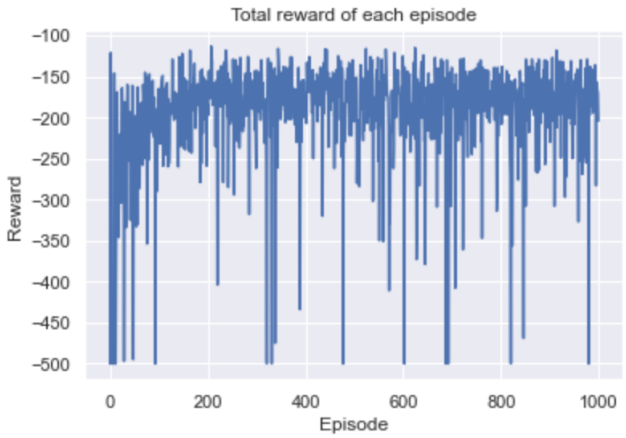

Figura 8: Recompensas obtenidas en cada iteración, Double Q-Learning

Con el objetivo de tener una visión más acertada del rendimiento de este modelo, se ejecuta 100 veces y se comparan los resultados obtenidos, Figura 9.

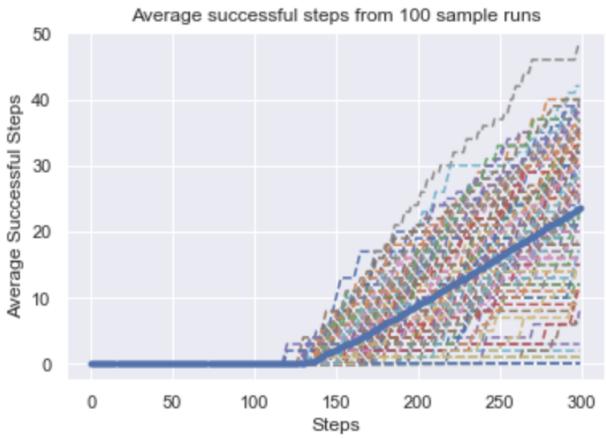

Figura 9: Rendimiento del modelo Double Q-Learning

La imagen anterior, Figura 9, representa el número de pasos satisfactorios medio a lo largo de cada episodio de 300 pasos. En este caso, de media, se dan 23,65 pasos exitosos, que representan el 7,88 % de los pasos totales y el primer paso satisfactorio medio es el 118.

IV-C. Algoritmo Dueling Deep Q-Learning

Los parámetros empleados en la implementación de este algoritmo son los siguientes:

- $\gamma = 0{,}99$: Factor de descuento que tiene muy en cuenta las recompensas futuras
- $\epsilon = 1{,}0$: Probabilidad inicial de que se escoja una acción aleatoria en vez de la determinada por la política
- $\epsilon_{decay} = 0{,}995$: Factor por el que se multiplica a ϵ cada vez que el entorno es resuelto
- $\alpha = 0{,}01$: Factor de aprendizaje del optimizador de la red neuronal
- $episodes = 1000$: Número de veces que se intenta resolver el entorno
- $batch\_size = 128$: Tamaño de batch
- Los vectores V y A son estimados utilizando una red neuronal compuesta por dos capas lineales.

En la Figura 10, se puede observar como el modelo es capaz de alcanzar el objetivo en tan solo unos pocos episodios. El problema, es que en las iteraciones posteriores, no siempre alcanza la meta, presentando el gráfico de alturas máximas, Figura 10, mucho ruido. Sin embargo, el gráfico de recompensas presenta otra realidad, Figura 11. Una vez conseguida la altura deseada por primera vez, el gráfico de recompensas muestra un cúmulo de recompensas cercano a -100 para todos los episodios consecutivos con muy poco ruido, pese algunas excepciones.

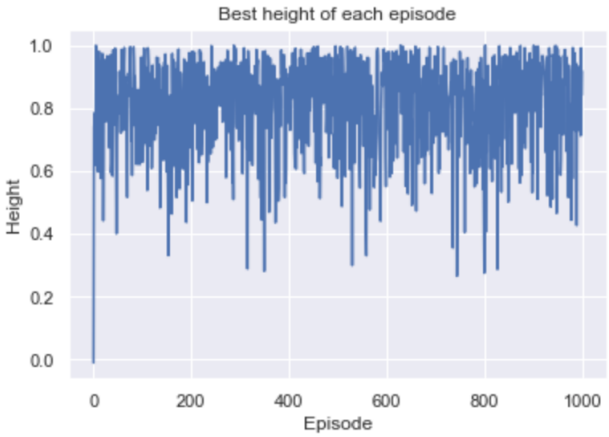

Figura 10: Mejor altura conseguida en cada iteración, Dueling Deep Q-Learning

Figura 11: Recompensas obtenidas en cada iteración, Dueling Deep Q-Learning

Con el objetivo de tener una visión más acertada del rendimiento de este modelo, se ejecuta 100 veces y se comparan los resultados obtenidos, Figura 12.

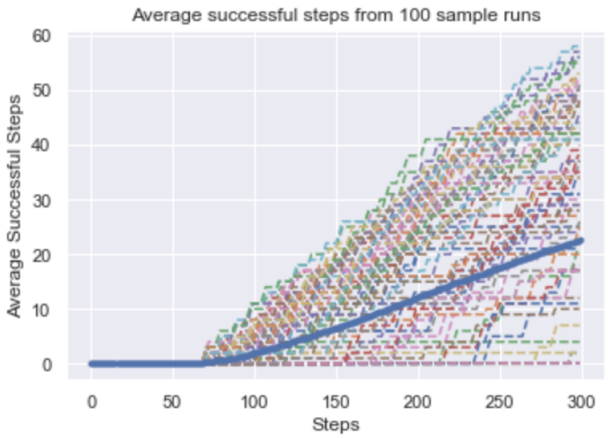

Figura 12: Rendimiento del modelo Dueling Deep Q-Learning

La imagen anterior, Figura 12, representa el número de pasos satisfactorios medio a lo largo de cada episodio de 300 pasos. En este caso, de media se dan 22,63 pasos exitosos, que representan el 7,54 % de los pasos totales y el primer paso satisfactorio medio es el 69.

V. Conclusiones y Recomendaciones

La Tabla I y la Figura 13 muestra los rendimientos de los modelos entrenados anteriormente

Modelo	Número medio de pasos exitosos	Primer paso exitoso medio
Q-Learning	21,47	100
Double Q-Learning	23,65	118
Dueling Deep Q-Learning	22,63	69

Tabla I: Comparativa de rendimiento

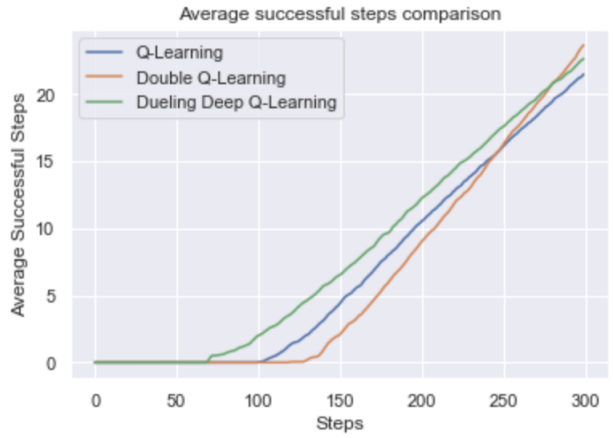

Figura 13: Comparativa de los rendimientos de todos los modelos

En la tabla, Tabla I, se puede observar que el mejor modelo en términos de número medio de pasos exitosos es el Double Q-Learning y que el mejor modelo en términos de primer paso exitoso medio es el Dueling Deep Q-Learning. Dado que la diferencia el número de pasos exitosos es muy pequeña entre estos dos modelos yo escogería el Dueling Deep Q-Learning como mejor modelo. Este modelo aprende es capaz de alcanzar la altura deseada muy rápido, pero después tiene problemas para mantenerse por encima de dicha altura. Si quisiésemos un modelo que consiguiese mantener el enlace inicialmente inferior por encima de la altura indicada durante más tiempo, habría que modificar la función de recompensa para indicarle al modelo que una vez alcanzada la meta queremos que el enlace se quede en esa posición, eso sería un problema distinto al que se ha resuelto en este proyecto.

REFERENCIAS

[1] OpenAI, “Acrobot-v1,” https://gym.openai.com/envs/Acrobot-v1/.
[2] Christoph Dann, “Acrobot,” https://github.com/openai/gym/blob/master/gym/envs/classic_control/acrobot.py.
[3] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, Nando de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning.”