Implementación del algoritmo Linear Q-Learning para la convergencia de un Doble Péndulo Invertido a su estado de equilibrio

Preprint · March 2021

CITATIONS
0

READS
721

1 author:

Walter Jesús Felipe Tolentino
Universidad Nacional de Ingeniería (Peru)
4 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Implementación de los algoritmos Deep y Linear Q-Learning para la convergencia de un Doble Péndulo Invertido a su estado de equilibrio*

* Proyecto para el curso Metodología de la Investigación CH033

Walter J. Felipe Tolentino
Escuela Profesional de Matemática
Universidad Nacional de ingeniería
Lima, Perú
wfelipet@uni.pe

***Abstract*—The Linear Q-Learning and Deep Q-Learning algorithms of the Reinforcement Learning have been implemented to converge the motion of an double inverted pendulum to its steady state in a measurable number of iterations. The results indicate that for a uniformly random perturbation of the equilibrium position in a short interval, the algorithms perform between 100 and 300 iterations on average to achieve convergence. This application of Machine Learning to the control of complex and chaotic systems reinforces the approach of learning algorithms in many fields where it is required to achieve the autonomy of the agent or system, compared to the classical controllers usually used.**

***Palabras clave*—Linear Q-Learning, Deep Q-Learning, OpenAI-gym, Doble péndulo invertido.**

I. Resumen

Se ha implementado los algoritmos de aprendizaje por refuerzo Linear Q-Learning y Deep Q-Learning para lograr la convergencia del movimiento de un doble péndulo invertido a su estado de equilibrio en una cantidad mensurable de iteraciones. Los resultados indican que para una perturbación uniformemente aleatoria de la posición de equilibrio en un pequeño intervalo, los algoritmos realizan entre 100 y 300 iteraciones de promedio para lograr la convergencia. Esta aplicación del Aprendizaje Automático al control de sistemas complejos y caóticos refuerzan el enfoque de los algoritmos de aprendizaje en muchos campos donde se requiere lograr la autonomía del agente o sistema, frente a los controladores clásicos usados usualmente.

II. Introducción

Desde la década de los 70 y especialmente en los últimos años, la dinámica y el control de n-péndulos invertidos ha llamado mucho la atención en el campo de estudio de la teoría e ingeniería de control moderna [1], [4].

Los controladores tradicionales, para sistemas dinámicos en general, consisten en controladores de retroalimentación, como el PID, u otros reguladores que requieren modelos matemáticos precisos. Estos métodos se utilizan ampliamente debido a su simplicidad y eficiencia. Sin embargo presentan numerosos problemas cuando los sistemas tiene una alta no linealidad y son inestables en el tiempo, o cuando el modelo dinámico es desconocido o no es preciso [2].

Se ha mostrado que los métodos de aprendizaje por refuerzo son una alternativa prometedora a los métodos clásicos de control, debido a que permite al agente (sistema dinámico) encontrar la mejor acción a partir de sus propias experiencias mediante ensayo y error [1], [5], es decir mediante un aprendizaje completamente no supervisado y centrándose en mejorar el rendimiento in-real-time, el cual implica encontrar un balance entre la exploración (un territorio aún inexplorado) y la explotación (conocimiento actual), muchos proyectos que emplean aprendizaje por refuerzo para controlar sistemas físicos siguen una línea de diseño marcada en la Figura 1.

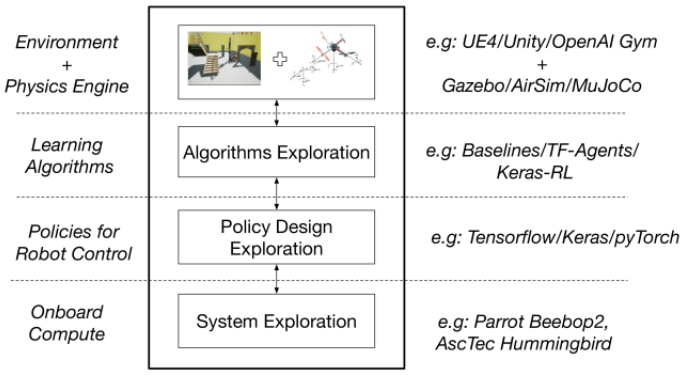

Figura 1. Diseño de un robot aéreo autónomo [14].

En la literatura se puede encontrar muchos métodos de aprendizaje por refuerzo tales como Q-Learning, SARSA (State–action–reward–state–action), métodos de actor-crítico (AC) y gradiente de políticas. Debido que el aprendizaje por refuerzo ha ganado gran atención en los últimos años gracias al éxito del sistema AlphaGo de DeepMind y al enorme desarrollo de librerías para OpenAI, muchos nuevos métodos han aparecido basados en Redes Neuronales Pro-

fundas (DNNs), Redes Neuronales Convolucionales (CNNs) y Redes Neuronales Recurrentes (RNNs) con una extensión de su arquitectura en memoría llamada LSTM (Long Short-Term Memory Network), que permite al agente "recordar" por más tiempo. Abriendo nuevos campos de estudio en el aprendizaje por refuerzo que permiten mejorar la forma de aprender de diversos agentes frente a un entorno asociado. Además gracias a la propiedad de autoaprendizaje del agente, los métodos de aprendizaje por refuerzo han ido aumentando su importancia día a día en muchas áreas, como control de producción, teoría de juegos, finanzas, comunicación, vehículos autónomos, robótica. Las aplicaciones más exitosas consisten en el control de robots móviles y humanoides, incluidos los sensores como cámara, LIDAR (Light Detection and Ranging o Laser Imaging Detection and Ranging), IMU (Inertial Measurement Unit), etc. [1], [6], [7].

Uno de los sistemas dinámicos más usados para demostrar la efectividad de los métodos de control nombrados anteriormente, es el péndulo simple invertido (se entiendo un péndulo cuyo movimiento es un plano) sobre un carro que se traslada es de forma horizontal, es muy usado en laboratorios, puesto que sirve de analogía para muchos sistemas reales [6], [7]. Aumentando la complejidad del control vamos a considerar la dinámica de otro péndulo unido al extremo superior del péndulo inicial, observe la figura 2. Esto hace un problema muy interesante puesto que el sistema es incluso, ahora, caótico [3].

III. El Doble Péndulo Invertido

Un doble péndulo invertido es un sistema no lineal subactuado (son aquellos sistemas que poseen menos entradas de control que grados de libertad), que presenta dos grados de libertad completos (θ_1 y θ_2) y una entrada de control (u) [11], además altamente complejo y caótico, cuya estabilización en el estado invertido y swing-up (caído a levantado), presenta muchos problemas usando los controles tradicionales. Los

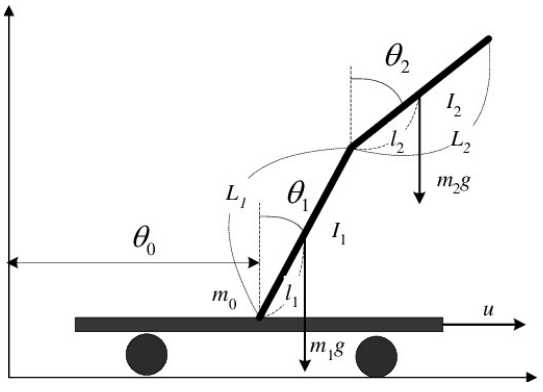

Figura 2. Un doble pendulo invertido sobre un carro

métodos de aprendizaje por refuerzo son un enfoque prometedor para esta tarea. Sin embargo, resolver un problema de control con aprendizaje por refuerzo es un desafío, porque el doble péndulo invertido tiene un espacio de estado continuo, y nosotros trabajamos en uno discreto [2].

Frente a ese desafió, la hipótesis del presente trabajo de investigación es : **Los algoritmos de aprendizaje por refuerzo Deep Q-Learning y Learning Q-Learning permiten que el movimiento de un doble péndulo invertido converja a su estado de equilibrio en una cantidad mensurable de iteraciones**, asimismo vamos a ver que ni la exploración ni la explotación se pueden realizar exclusivamente sin fallar en la tarea, o que pueden tomar un gran cantidad de tiempo el aprender a estabilizarse, dependiendo de las perturbaciones iniciales al sistema.

IV. Modelo matemático (*Ecuaciones del movimiento*)

En la figura 2 se muestra la denominación de algunas constantes, como las masas (m_i), ángulos (θ_i), longitudes (L_i) y los momentos de inercia de cada péndulo (I_i).

Comenzamos el análisis definiendo θ_0 como la traslación horizontal del carro, θ_1 como la desviación vertical del péndulo 1 y θ_2 la desviación vertical del péndulo 2.

Esto nos va permitir determinar de manera unívoca la configuración del doble péndulo invertido, tal que se satisface las condiciones de ligadura. Por lo tanto usamos las ecuaciones de Euler-Lagrange en su forma vectorial para obtener el sistema de ecuaciones diferenciales que modelan el movimiento del doble péndulo invertido.

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) + \frac{\partial \mathcal{L}}{\partial \theta} = \mathcal{Q} \tag{1}$$

donde $\theta = (\theta_0\ \theta_1\ \theta_2)^T$ y $\mathcal{Q} = (u\ 0\ 0)^T$
Cálculamos el Lagrangiano $\mathcal{L} = T - V$, hallando la energía cinética y potencial del sistema por cada componente (carro, péndulos 1 y 2)

$$T = T_0 + T_1 + T_2$$

$$V = V_0 + V_1 + V_2$$

Donde

$$\begin{aligned}
T_0 &= \frac{1}{2} m_0 \dot{\theta_0}^2 \\
T_1 &= \frac{1}{2} m_1 \left[(\dot{\theta_0} + l_1 \dot{\theta_1} \cos\theta_1)^2 + (l_1 \dot{\theta_1} \sin\theta_1)^2 \right] \\
&\quad + \frac{1}{2} I_1 \dot{\theta_1}^2 \\
T_2 &= \frac{1}{2} m_2 (\dot{\theta_0} + L_1 \dot{\theta_1} \cos\theta_1 + l_2 \dot{\theta_2} \cos\theta_2)^2 \\
&\quad + \frac{1}{2} m_2 (L_1 \dot{\theta_1} \sin\theta_1 + l_2 \dot{\theta_2} \sin\theta_2)^2 + \frac{1}{2} I_2 \dot{\theta_2}^2 \\
V_0 &= 0 \\
V_1 &= m_1 g l_1 \cos\theta_1 \\
V_2 &= m_2 g (L_1 \cos\theta_1 + l_2 \cos\theta_2)
\end{aligned}$$

Derivando el Lagrangiano $\mathcal{L} = T_0 + T_1 + T_2 - (V_0 + V_1 + V_2)$ por $\dot{\theta}$ y θ y remplazando en la ec. 1 tenemos el siguiente

sistema no lineal de tres ecuaciones diferenciales de segundo orden

$$\begin{aligned} u &= (m_0 + m_1 + m_2)\ddot{\theta}_{\mathbf{0}} + (m_1 l_1 + m_2 L_1)\cos(\theta_1)\ddot{\theta}_{\mathbf{1}} \\ &\quad + m_2 l_2 \cos(\theta_2)\ddot{\theta}_{\mathbf{2}} - (m_1 l_1 + m_2 L_1)\sin(\theta_1)\dot{\theta}_{\mathbf{1}}^{\mathbf{2}} \\ &\quad - m_2 l_2 \sin(\theta_2)\dot{\theta}_{\mathbf{2}}^{\mathbf{2}} \\ 0 &= (m_1 l_1 + m_2 L_1)\cos(\theta_1)\ddot{\theta}_{\mathbf{0}} + (m_1 l_1^2 + m_2 L_1^2 + I_1)\ddot{\theta}_{\mathbf{1}} \\ &\quad + m_2 L_1 l_2 \cos(\theta_1 - \theta_2)\ddot{\theta}_{\mathbf{2}} + m_2 L_1 l_2 \sin(\theta_1 - \theta_2)\dot{\theta}_{\mathbf{2}}^{\mathbf{2}} \\ &\quad - (m_1 l_1 + m_2 L_1) g \sin(\theta_1) \\ 0 &= m_2 l_2 \cos(\theta_2)\ddot{\theta}_{\mathbf{0}} + m_2 L_1 l_2 \cos(\theta_1 - \theta_2)\ddot{\theta}_{\mathbf{1}} \\ &\quad + (m_2 l_2^2 + I_2)\ddot{\theta}_{\mathbf{2}} - m_2 L_1 l_2 \sin(\theta_1 - \theta_2)\dot{\theta}_{\mathbf{1}}^{\mathbf{2}} \\ &\quad - m_2 l_2 g \sin(\theta_2) \end{aligned} \tag{2}$$

Para el diseño de la ley de control la ec. 2 debe ser reformulada en seis ecuaciones no lineales de primer orden, esto nos va permitir configurar el estado del doble péndulo invertido.

Sea $\mathbf{s} \in \mathbb{R}^6$, $\mathbf{s} = (\theta\ \dot{\theta})^T$, luego en la ec. 2

$$\dot{\mathbf{s}} = \begin{pmatrix} 0 & \mathbf{I} \\ 0 & -\mathbf{D^{-1}C} \end{pmatrix}\mathbf{s} + \begin{pmatrix} 0 \\ -\mathbf{D^{-1}G} \end{pmatrix} + \begin{pmatrix} 0 \\ \mathbf{D^{-1}H} \end{pmatrix}u$$

Las matrices $\mathbf{D}, \mathbf{C}, \mathbf{H}$ y $\mathbf{G}$ son tal que se despejan de la ec. 2 por el cambio de variable, revise [8] para más detalle. Luego el sistema de ecuaciones del doble péndulo invertido puede ser rescrito como

$$\dot{\mathbf{s}} = \mathbf{A}(\mathbf{s})\mathbf{s} + \mathbf{b}(\mathbf{s}) + \mathbf{c}(\mathbf{s})u \tag{3}$$

Donde $\mathbf{A} \in \mathbb{R}^{6\times6}, \mathbf{b} \in \mathbb{R}^6, \mathbf{c} \in \mathbb{R}^6$, $u \in \mathbb{R}$ y la variable de estado

$$\mathbf{s} = (\theta_0\ \theta_1\ \theta_2\ \dot{\theta}_0\ \dot{\theta}_1\ \dot{\theta}_2)^T$$

Si quisieramos usar un método clásico de control como el PID, usando por ejemplo lazo cerrado, con una matriz de retroalimentación, entonces para un mejor análisis la ec. 3 se tendría que linealizar usando la parte lineal de la expansión de Taylor de la función que define la ecuación diferencial, esto solo muestra que podríamos controlar el doble péndulo invertido para ínfimas variaciones de $\mathbf{s}$, sino el estudio se complica dada la no linealidad del sistema [4]. Por eso empleamos los algoritmos de aprendizaje por refuerzo que se detallan a continuación.

V. Métodos y Materiales

A. Linear Q-Learning

1) Proceso de decisión de Markov (MDP): A partir de un conjunto de estados ($\mathbf{s}$), acciones ($\mathbf{A}$), recompensas ($\mathbf{R}$) y probabilidades de transición estado-acción-estado vamos a construir un MDP para el doble péndulo invertido que a partir de ahora llamaremos agente.

El estado del MDP es $\mathbf{s} \in \mathbb{R}^6$ que está presente en la ec. 3, permitiendo configurar unívocamente el agente. Como hay grados completos de libertad y la traslación del carro, el espacio de estados es continuo.

Las acciones $a \in \mathbf{A}$ del agente corresponden a seleccionar qué fuerza u debe ser aplicada en el carro. Como vamos a trabajar en un espacio discreto, entonces u solo puede pertenecer a un conjunto finito de valores. La probabilidad de cada una de las transiciones entre estados por una acción son desconocidas por el agente, pero intrínsecamente dadas por la ec. 3, es decir estudiando como responde el agente cuando realiza cierta acción.

Para especificar las recompensas, consideramos dos tipos diferentes de costos y luego el conjunto de recompensas igualará el costo total negativo. El primer tipo de costo está asociado con cualquier episodio fallido, para el swing-up definimos un episodio fallido un vez que el agente se mueva fuera de los bordes, esto es $|\theta_0|$ es tan grande que supera cierto valor umbral. Para el control del equilibrio una vez ya invertido definimos un episodio fallido una vez que el péndulo se caiga, eso es cuando $|\theta_1| > \frac{\pi}{2}$. El segundo tipo de costo esta asociado con cada estado, penalizando cualquier posición del agente distinta a la del equilibrio (estado objetivo o límite) es decir

$$\theta = (\theta_0\ \theta_1\ \theta_2)^T \neq (0\ n2\pi\ m2\pi)^T$$

Donde $n, m \in \mathbb{Z}$. Finalmente, el factor de descuento a considerar es $\gamma = 0.99$ [3], esto nos va permitir restar las recompensas de las acciones lejanas y futuras, puesto que mientras más lejos esté una acción para llegar a cierto estado objetivo no nos interesa en el estado actual, tan solo las acciones a corto plazo.

Una vez definido lo que usaremos en el MDP vamos a relacionar a cada estado y acción tomada por el agente, un valor llamado **q-value**, estos valores van a formar una matriz Q donde se encuentra el valor máximo $q(\mathbf{s}, a)$ para cada par $\mathbf{s}$,a, esto solo cuando converja (es decir cuando alcancemos un estado límite) o cuando termine la cantidad de episodios definida a priori. En cada episodio debemos ir disminuyendo una función de error la cuál participará la función $q_\pi(\mathbf{s}, a)$ siguiendo una cierta política π (mejor acción para cada estado), se detalla más en las especificaciones del algoritmo Linear Q-Learning.

2) Entorno de simulación: El sistema físico del agente es simulado usando una versión modificada del entorno CartPole-v1 en OpenAI Gym [3], [13], en el lenguaje de programación Python v3.*.

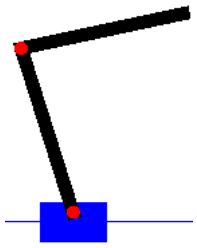

Figura 3. Entorno de simulación usando OpenAI Gym.

En cada episodio del tiempo el agente va seleccionar una acción $a = u$ la cual es aplicado al sistema en su estado actual $\mathbf{s}_t$, el sistema luego es propagado al siguiente paso usando las ecuaciones del agente (ec. 3) cuya solución numérica la

encontramos usando el método de **Runge-Kutta** usando el paquete scipy.integrate.ode de Python. El agente ahora recibe un nuevo estado $\mathbf{s}_{t+1}$ e inmediatamente la recompensa asociada $\mathbf{r}_{t+1}$ tomando la acción elegida a (ya sea por exploración o explotación) por el estado original $\mathbf{s}_t$ y en cada episodio se disminuyendo el costo total a través de una función de error. El estado actual es renderizado en cada episodio, mirar la figura 5, usando el parquete matplotlib.pyplot de Python [3].

3) Algoritmo: El algoritmo Q-Learning esta basado en el método valores por iteración (value iteration), este usa la ecuación de Bellman para actualizar los valores de la tabla Q, lo que significa que aprendemos una función de valor $q_\pi(\mathbf{s}, a)$ y luego usamos la política π implícita basada en esta función de valor [3], [12].

Como el espacio de estados es continuo no podemos implementar un algoritmo Q-Learning regular, por eso usaremos una función lineal de aproximación [3], es decir el algoritmo Linear Q-Learning. Aproximaremos al valor verdadero $q_\pi(\mathbf{s}, a)$, por una combinación de n vectores x_j, es decir

$$q_\pi(\mathbf{s}, a) \approx \sum_{j=1}^{n} w_j x_j(\mathbf{s}, a) = \mathbf{x}(\mathbf{s}, a)^T \mathbf{w} = \hat{q}(\mathbf{s}, a, \mathbf{w}) \quad (4)$$

Donde $\mathbf{x}(\mathbf{s}, a) \in \mathbb{R}^n$ es el vector de características del estado y acción actuales, nuestro objetivo es elegir el parametro $\mathbf{w} \in \mathbb{R}^n$ tal que minimice el error cuadrático medio entre $\hat{q}$ y q_π [3], haciendo uso del método del gradiente descendente, la regla para actualizar es

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(q_\pi(\mathbf{s}, a) - \hat{q}(\mathbf{s}, a, \mathbf{w}))\mathbf{x}(\mathbf{s}, a)$$

Donde $\alpha \in \mathbb{R}$ es el tamaño del paso. Como la función $q_\pi(\mathbf{s}, a)$ es desconocida, la reemplazamos por el la representación de Bellman y obtenemos la siguiente regla para actualizar los valores de $\mathbf{w}$

$$\Delta\mathbf{w} = \alpha(r' + \gamma \max_{a'} \hat{q}(\mathbf{s}', a', \mathbf{w}) - \hat{q}(\mathbf{s}, a, \mathbf{w}))\mathbf{x}(\mathbf{s}, a) \quad (5)$$

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \quad (6)$$

Donde r' es la recompensa inmediata asociada con la acción a desde el estado s a el estado siguiente s'. Notar que el espacio de acciones A en este caso es discreto, así que la operación $\max_{a'}$ en 5 se convierte en una simple comparación [3]. En cada episodio mejoramos las políticas, es decir escogemos $a = \arg\max_{a'} \hat{q}(\mathbf{s}, a', \mathbf{w})$ con una probabilidad $1-\epsilon$ (explotación) y una acción aleatoria de A con probabilidad ϵ (exploración), y aplicamos la regla de actualización anterior 5 y 6. Todo esto forma el algoritmo 1.

Se inicializa el estado del doble péndulo invertido con una ligera variación del primer péndulo, dada una desviación de su ángulo elegido aleatoriamente e uniformemente entre -0.1 y 0.1. El método step resuelve la ec. 3 en el estado $\mathbf{s_t}$ y la acción a_t retornando el estado $\mathbf{s}_{t+1}$, además retorna la recompensa igualando al negativo del costo de llegar al siguiente estado ($\mathbf{s}_{t+1}$) a través de a_t y por último retorna un valor booleano, el cual es verdad si se penaliza con los costos asociados al episodio detallados, en el MDP para luego terminar el episodio y comenzar otro.

Algorithm 1: Linear Q-Learning, modificado de [3].

$\mathbf{w}_0 \leftarrow 0$;
repeat
  empezamos un nuevo episodio;
  $\mathbf{s}_0 \leftarrow (0.0, \mathbf{rand}[\mathbf{U}(-\mathbf{0.1}, \mathbf{0.1})], 0.0, 0.0, 0.0, 0.0)$;
  $t \leftarrow 0$;
  repeat
    $a_t \leftarrow \arg\max_{a'} \hat{q}(\mathbf{s_t}, a', \mathbf{w_t})$;
    $(s_{t+1}, r_{t+1}, \mathit{bool}) \leftarrow \text{step}(a_t)$;
    $a \leftarrow \arg\max_{a'} \hat{q}(\mathbf{s_{t+1}}, a', \mathbf{w_t})$;

    $\Delta\mathbf{w} \leftarrow \alpha(r_{t+1} + \gamma\hat{q}(\mathbf{s}_{t+1}, a', \mathbf{w}_t) - \hat{q}(\mathbf{s}_t, a_t, \mathbf{w}_t))\mathbf{x}(\mathbf{s}_t, a_t)$;
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \Delta\mathbf{w}$;

    if *bool* **then**
      show(Episodio finalizado luego de $t+1$ it.);
      show($\mathbf{w}_{t+1}$);
      break;
    end
    $t \leftarrow t + 1$;
  until *un número máximo de iteraciones*;
until *un número máximo de episodios o hasta que converja*;

B. Deep Q-Learning

En el caso de Linear Q-Learning, para la actualización del parámetro $\mathbf{w}$, tal que minimice el error cuadrático medio entre $\hat{q}$ y q_π, se aproxima q_π con una función $\hat{q}$, que es combinación lineal de vectores que representan características del estado y acción actuales, luego se hace uso del método del gradiente descendente y usando la Ec. de Bellman para q_π se obtiene la ecuación 5. En síntesis este proceso mapea un par estado-acción a un valor q_π, esa terna (s, a, q_π) puede ser representado en una tabla, llamada tabla Q.

Con el aprendizaje profundo encontrar los valores q_π es una tarea para la red neuronal profunda, debido a una de sus aplicaciones de aproximar funciones. Nos brinda a partir de estados de entrada s_t, un par (a, q_π). Una de las cosas interesantes del algoritmo Deep Q-Learning es que el proceso de aprendizaje usa dos redes neuronales, llamadas principal y objetivo, con la misma arquitectura pero diferentes pesos, es decir luego de una cierta cantidad de pasos la red principal se copia en la red de destino, estas redes usan la misma función de activación. El uso de estas dos redes conduce a una mayor estabilidad en el proceso de aprendizaje y ayuda al algoritmo a aprender de manera más efectiva. En nuestra implementación, los pesos de la red principal reemplazan los pesos de la red objetivo cada cien pasos.

El valor q_π de salida, representa el resultado de la elección de una acción partiendo de un estado s_t. Tanto el modelo principal como el modelo objetivo asignan estados de entrada a acciones de salida. Estas acciones de salida en realidad representan el valor q_π predicho del modelo. En este caso, la

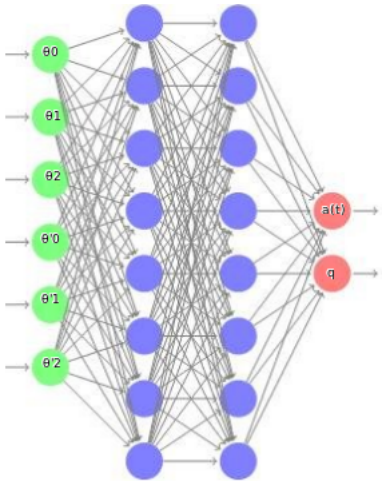

Figura 4. Red neuronal que asigna un estado de entrada a su par correspondiente (a, q_π) [15].

acción que tiene el valor q_π predicho más grande es la acción mejor conocida en ese estado.

Las características más notables son que usamos la inicialización uniforme de He, así como la función de pérdida de Huber para lograr un mejor rendimiento.

Además se emplea la inicialización uniforme de He-et-al, en este método, los pesos se inicializan teniendo en cuenta el tamaño de la capa anterior, lo que ayuda a lograr un mínimo global de la función de costo de manera más rápida y eficiente. Los pesos siguen siendo aleatorios pero difieren en el rango según el tamaño de la capa anterior de neuronas. Esto proporciona una inicialización controlada y, por lo tanto, un descenso de gradiente más rápido y eficiente.

Además se emplea la función de perdida de Huber que es menos sensible a valores atípicos en los datos que la pérdida de error al cuadrado, para lograr un mejor rendimiento.

Al igual que Linear Q-Learning el agente aún necesita actualizar los pesos de nuestro modelo de acuerdo con la Ecuación de Bellman.

1) Entorno de simulación: El mismo que en el anterior algoritmo, con el añadido del paquete de aprendizaje por refuerzo Tensorflow.keras, para modelar las redes neuronales que van calcular y predecir los nuevos valores de q y una acción.

VI. Implementación y Resultados

A. Convergencia al equilibrio

Para cada episodio nuevo se configura el estado inicial, perturbando el estado de equilibrio, tal como sigue:

$$\mathbf{s_0} = [0 \quad \theta_{\mathbf{1_0}} \quad \theta_{\mathbf{2_0}} \quad 0 \quad 0 \quad 0]^T$$

Donde $\theta_{\mathbf{1_0}}, \theta_{\mathbf{2_0}} \sim \mathbf{U}[-\lambda, \lambda]$.

Durante el entrenamiento se trabajo con $\lambda = 0.1$ y 300 el límite máximo de iteraciones. Si este límite es alcanzado el episodio se define como exitoso, aproximándose a la convergencia en menor tiempo en los próximos episodios.

1) Linear Q-Learning: Se considera la probabilidad de exploración $\epsilon = 0.0001$, el espacio de estados $A = \{0, \pm5, \pm10, \pm20, \pm40\}$ y el parámetro w inicializado al vector nulo.

Las características estado-acción usadas para aproximar la función q_π se encuentran en el siguiente vector de características

$$\mathbf{x}(\mathbf{s}, \mathbf{a}) = [f_1, ..., f_{18}]^T \tag{7}$$

Donde $f_1, ..., f_{18}$ son las características, generalmente son combinaciones algebraicas del vector de estados, y están dadas en la tabla I.

TABLA I
Características del algoritmo Linear Q-Learning [3]

$f_1 = \theta_0^2$	$f_2 = \theta_0$
$f_3 = normalAngle(\theta_1)^2$	$f_4 = normalAngle(\theta_1)$
$f_5 = normalAngle(\theta_2)^2$	$f_6 = normalAngle(\theta_2)$
$f_7 = \dot{\theta_0}^2$	$f_8 = \dot{\theta_0}$
$f_9 = \dot{\theta_1}^2$	$f_{10} = \dot{\theta_1}$
$f_{11} = \dot{\theta_2}^2$	$f_{12} = \dot{\theta_2}$
$f_{13} = a\theta_0$	$f_{14} = a\dot{\theta_0}$
$f_{15} = a\theta_1$	$f_{16} = a\dot{\theta_1}$
$f_{17} = a\theta_2$	$f_{18} = a\dot{\theta_2}$

Donde $normalAngle(\phi)$ retorna la diferencia entre ϕ y su multiplo más cercano a 2π.

Con todos estos parámetros, el algoritmo converge en el 70% de episodios de entrenamiento, como se muestra en la curva de aprendizaje de la Figura 5.

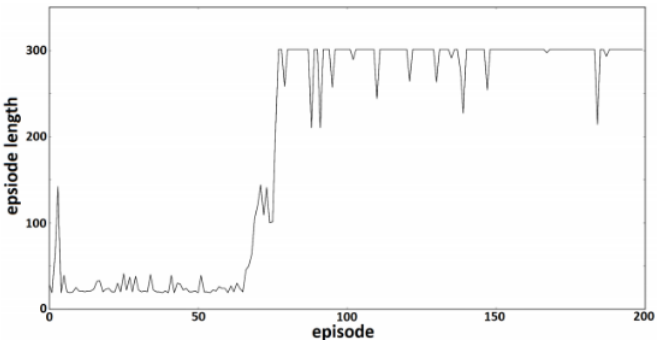

Figura 5. Curva de aprendizaje del algoritmo Linear Q-Learning, relacionado con el trabajo base [3].

Con los valores de $\mathbf{w}$ ya establecidos anteriormente, para probar el rendimiento y la robustez del algoritmo, se modifico la perturbación inicial para valores de $\lambda \in \{0.1, 0.2, 0.3, 0.4\}$, se simulo 100 episodios para cada λ, llegando a los resultados que se muestran en la tabla II.

2) Deep Q-Learning: Inicializamos $\epsilon = 1$, lo que significa que cada elección de la acción es aleatoria al inicio, decayendo a $\epsilon = 0.01$ cuando se requiere más explotar que explorar. La frecuencia de aprendizaje es 0.7 y el factor de descuento 0.618.

Es importante tener la frecuencia de actualización de los parámetros del modelo correcta. Si actualiza los pesos del

TABLA II
PRUEBA DE RENDIMIENTO DEL ALGORITMO LINEAR Q-LEARNING

λ	% de episodios con éxito
0.1	94
0.2	92
0.3	78
0.4	49

modelo con demasiada frecuencia (por ejemplo, después de cada iteración), el algoritmo aprenderá muy lentamente cuando no haya cambiado mucho. En este caso, realizamos actualizaciones de peso del modelo principal cada 4 iteraciones, lo que ayuda a que el algoritmo se ejecute significativamente más rápido [16].

Para nuestros propósitos las redes principal y objetivo son bastante simples y consisten en 3 capas (6 entradas , 2 salidas y una capa oculta), similar a la Figura 5, con pesos iniciales dada por la distribución He (tf.keras.initializers.HeUniform), diseñadas con el paquete tensorflow.keras de Python, con funciones de activación de ReLU (en. Rectified Linear Activation Function),

$$f(x) = \max(0, x)$$

Y cuya aproximación suave del rectificador es la función analítica $f(x) = \ln(1 + e^x)$. Además se trabaja con la función de perdida Huber (tensorflow.keras.losses.Huber) para mínimizar el error.

Se configura una cantidad de 300 episodios para el entrenamiento y 100 episodios de prueba con $\lambda = 0.05$, resultando la convergencia luego de una cantidad máxima de 100 iteraciones para todos los episodios.

VII. CONCLUSIONES

Se ha implementado el algoritmo de aprendizaje por refuerzo Q-Learning relativamente simple para controlar un sistema dinámico complejo y desconocido, como lo es el doble péndulo invertido, se estudio con éxito un controlador para equilibrar usando Linear Q-Learning para pequeñas perturbaciones del estado inicial del doble péndulo invertido, dado que solo usamos aproximadores de funciones lineales. Los problemas de control más desafiantes parecen requerir características inteligentemente diseñadas con una mejor selecciones de funciones aproximadoras. Es por ello que abordamos el control usando Deep Q-Learning encontrando mejores resultados en la convergencia a la posición de equilibrio.

Para trabajos futuros se puede estudiar algunas técnicas de Aprendizaje por refuerzo más avanzadas, como Double DQN Networks y Prioritized Experience Replay [16], diseñando un modelo de control para el doble péndulo invertido, que pueden mejorar aún más el proceso de aprendizaje. Estas técnicas nos darían mejores resultados con un número menor de episodios. Por otro lado existen nuevos modelos de memoria, por ejemplo emplear arquitecturas LSTM para encontrar un mejor balance entre exploración y explotación.

La dinámica del doble péndulo invertido es usada para entender los procesos que implican el mantenimiento del balance, tal como caminar, control de propulsores de cohetes y sistemas mecánicos auto-balanceados (auto-equilibrados) [9]. Y si llegamos a generalizar el trabajo para n-péndulos se vuelve un reto muy bello que lo dejaremos para trabajos futuros.

AGRADECIMIENTOS

A los profesores del curso Metodología de la Investigación CH033.

REFERENCIAS

[1] Özalp R., Varol N.K., Taşci B., Uçar A. "A Review of Deep Reinforcement Learning Algorithms and Comparative Results on Inverted Pendulum System," In: Tsihrintzis G., Jain L. (eds) Machine Learning Paradigms. Learning and Analytics in Intelligent Systems, Springer, Cham, 2020, vol 18, pp 237-256. doi.org/10.1007/978-3-030-49724-8_10

[2] A. Ghio and O. E. Ramos, "Q-learning-based Model-free Swing Up Control of an Inverted Pendulum," 2019 IEEE XXVI International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Lima, Peru, 2019, pp. 1-4. doi: 10.1109/INTERCON.2019.8853619.

[3] F. Gustafsson, "Control of an Inverted Double Pendulum using Reiforcement Learning," Stanford University, 2016.

[4] G. Valandia," Modelamiento, Diseño y Simulación de un sistema de control para un Sistema de Doble Péndulo Invertido," Universidad Autónoma de Colombia, 2007.

[5] K. Ogata, Y. Yang, Modern Control Engineering, vol. 4 London, 2002.

[6] A. S. Diaz, "Aprendizaje por refuerzo para el control de sistemas dinámicos". Universidad Autónoma de Occidente, Facultad de Ingeniería, Departamento de Automática y Electrónica, Cali, Colombia, 2019.

[7] R.S. Sutton, A.G. Barto, "Reinforcement Learning: An Introduction", MIT Press, 2018.

[8] A. Bogdanov, "Optimal Control of a Double Inverted Pendulum on a cart," Technical Report CSE-04-006, 2004.

[9] S. Nagendra, N. Podila, R. Ugarakhod, K. George, "Comparison of Reinforcement Learning Algorithms applied to the Cart-Pole Problem," University Bangalore, India.

[10] RL course by David Silver. https://www.davidsilver.uk/teaching/, 2015.

[11] E. A. Mamani, " Diseño y control en tiempo real del sistema subactuado pendubo," Universidad Nacional de Ingeniería, Ingeniería Mecatrónica, 2012. cybertesis.uni.edu.pe/handle/uni/2291.

[12] Course UC Berkeley CS188 Intro to AI, Project 3: Reinforcement Learning, Version 1.001, 2014, http://ai.berkeley.edu/reinforcement.html#Q8. http://ai.berkeley.edu/lecture_videos.html

[13] Open AI Gym - Cartpole v1, https://gym.openai.com/envs/CartPole-v1/

[14] Srivatsan Krishnan, Behzad Boroujerdian, William Fu, Aleksandra Faust, and Vijay Janapa Reddi, "Air Learning: An AI Research Platform for Algorithm-Hardware Benchmarking of Autonomous Aerial Robots," Harvard University, The University of Texas at Austin, Robotics at Google, 2019.

[15] Sheng Wang, Lieyong Fu, Jianmin Yao, Yun Li, The Application of Deep Learning in Biomedical Informatics, Conference: 2018 International Conference on Robots & Intelligent System (ICRIS), DOI: 10.1109/ICRIS.2018.00104

[16] Mike Wang, Deep Q-Learning - A Practical Guide to Deep Q-Networks. 2020, https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc