Universidad EAFIT

Maestría en Ciencia de Datos y Analítica

Aprendizaje Automático

S2261-0136

Aprendizaje por Reforzamiento: Control de Sistemas Dinámicos

Autores:

Alejandro Barrientos-Osorio

Luis Miguel Caicedo-Jimenez

Omar Alejandro Henao-Zapata

26 de abril de 2022

I. INTRODUCCIÓN

Recientemente, el interés por resolver problemas complejos utilizando inteligencia artificial ha venido incrementando gracias a los avances que se han realizado en términos de capacidad de cómputo. El aprendizaje reforzado consiste en una serie de métodos de solución de problemas, en los cuales la base es el aprendizaje a partir de la interacción con el entorno, con el fin de maximizar una recompensa [1]. Teniendo esto en cuenta, este tipo de algoritmos se pueden emplear en problemas en los cuales se tengan salidas que ocasionen cambios observables en el entorno.

Uno de los problemas más grandes en ingeniería es el control de procesos dinámico, en el cual se tienen una serie de sensores en tiempo real, obteniendo grandes volúmenes de información por unidad de tiempo, los cuales deben ser procesados para tomar acciones en torno a un objetivo claro [2]. Un ejemplo actual de este tipo de problemas son los sistemas de control de los autos autónomos, los cuales perciben su entorno y toman decisiones con base en él, aunque también existen aplicaciones en el modelado de procesos productivos y el control de máquinas para la consecución de ciertos objetivos.

En este trabajo, se empleará la metodología de aprendizaje reforzado para el diseño de un modelo capaz de controlar un sistema dinámico conocido como el péndulo invertido, para el cual se tienen unos conjuntos de acciones y estados no finitos. Se definirá el modelo de decisión Markoviano, y cómo este puede ser utilizado para la maximización de la recompensa, la cual será otorgada cuando la barra de metal se encuentre en posición vertical.

A. Objetivo General

- Aplicar el método DDPG de aprendizaje reforzado a un problema de control dinámico, para espacios de estado-acción continuos.

B. Objetivos Específicos

- Implementar en Python la simulación del entorno del péndulo invertido, con el fin de definir una política de aprendizaje para tomar acciones basadas en el estado del sistema.

- Aproximar funciones de valor utilizando redes neuronales y políticas deterministas basadas en memoria temporal de observaciones.
- Realizar análisis de sensibilidad de los resultados, para evaluar las variaciones del desempeño del modelo con respecto a los hiperparámetros de entrenamiento.

II. METODOLOGÍA

La metodología empleada fue CRISP-DM, la cual es empleada en diversas aplicaciones de analítica. Cabe resaltar que no se realiza la etapa de implementación, dado que el alcance del trabajo es el estudio de los algoritmos de aprendizaje por reforzamiento.

A. Entendimiento del Negocio

Para poder ejecutar un algoritmo de aprendizaje reforzado, es fundamental comprender en primer lugar el problema a solucionar. En este caso, se tiene a disposición la simulación del problema del péndulo invertido, disponible en la librería *gym* de Python. El problema consiste en lograr mantener una barra en posición vertical, sobre el eje en el que gira, al imprimir un torque bien sea positivo o negativo que produzca un momento angular en la barra. En la Figura 1, se observa la representación gráfica del problema generado con gym en Python.

Esta configuración representa un problema de control continuo y dinámico, en el cual las variables cambian con el tiempo, y no se llega a un estado estacionario de manera espontánea. Es por este motivo que es necesario implementar un agente que pueda procesar las variables de entrada para producir una acción sobre el entorno, que le permita conseguir el objetivo deseado.

Generalmente, este tipo de configuraciones son solucionadas con controladores Proporcionales Integrales y Derivativos (PID), los cuales son costosos de implementar y requieren de una modelación del entorno bastante acertada con el fin de obtener resultados satisfactorios. Por esta razón, y con el fin de simplificar el problema de modelado, es posible emplear un método de aprendizaje automático que no requiera un modelo explícito del entorno. Es aquí donde el aprendizaje reforzado toma relevancia en el problema.

Figura 1: Entorno del péndulo invertido generado con gym en Python

Otra característica del problema que permite el uso de métodos de aprendizaje reforzado, es la dependencia de un estado con el inmediatamente anterior: dado el estado de la barra, definido por su posición y velocidad angular, y teniendo en cuenta que es un fenómeno físico, es posible predecir el estado siguiente teniendo en cuenta la acción que se tome. Debido a esto, es posible afirmar que el sistema tiene la *propiedad de Markov*, la cual establece que la respuesta del entorno en un tiempo $t+1$, depende únicamente del estado y la acción tomada en t. De esta manera, las dinámicas del entorno pueden definirse en su totalidad como [1]:

$$p(s^*, r|s, a) = P\{R_{t+1} = r, S_{t+1} = s^*|S_t, A_t\}$$

la cual establece que la probabilidad de obtener una recompensa r en el estado s, solo depende del estado actual (S_t) y la acción tomada (A_t). Así, será posible establecer políticas y

funciones de valor que puedan ser utilizadas con la ecuación de Bellman para la maximización de la recompensa.

Dentro de los métodos de aprendizaje automático, encontramos el Deep Deterministic Policy Gradient (DDPG) [3], el cual consiste en encontrar una política $\pi(a|s)$ determinista con la cual, a diferencia de otras aproximaciones, se obtiene un valor continuo representando la acción a tomar, y no una serie de probabilidades de tomar diversas acciones dado un estado. Esto posibilita que los resultados sean consistentes y que se tengan las acciones correctas en cada uno de los estados disponibles. Para establecer esta política, se hace uso de redes neuronales, las cuales permiten transformar estados en acciones sin necesidad de tener un modelo del entorno, y además tener resultados en variables continuos.

Adicional a esto, el DDPG utiliza redes neuronales para calcular las funciones de valor del problema. En este caso, se estiman valores $Q(s, a)$, correspondientes a la función de valor estado-acción, que permite evaluar la calidad de la acción tomada en determinado estado. Esta función se aproxima a través de redes neuronales profundas, y también tienen un comportamiento determinista, en el sentido de que para el mismo par ordenado estado-acción, se obtiene el mismo valor Q.

De acuerdo con la propiedad de Markov del problema, y la posibilidad que ofrecen los métodos de aprendizaje reforzado profundo, se decide emplear el método DDPG para la solución del problema. En la siguiente sección, se describen las características de los datos a utilizar, con el fin de determinar la estructura del modelo.

B. Entendimiento de los datos

Con el fin de poder implementar el método DDPG, es necesario entender cuáles es la información que se tiene del entorno para representar el estado, así como las acciones que pueden tomarse sobre él y el efecto obtenido.

1. Ingeniería de Características

De acuerdo con la documentación de la librería *gym*, el estado del sistema está definido por tres variables:

1. Posición en x del extremo libre: [-1.0, 1.0]

2. Posición en y del extremo libre: [-1.0, 1.0]

3. Velocidad angular del péndulo: [-8.0, 8.0]

Las 3 variables están representadas en un medio continuo, con lo cual el uso de las redes neuronales dará una gran facilidad en su manejo. Estas variables se tomarán para el entrenamiento del modelo, sin realizar modificaciones adicionales. Cabe resaltar que otras formas de representar el estado pueden realizarse analizando la representación de la imagen y realizando convoluciones para tener una simplificación de la información mostrada. Sin embargo, el alcance de este trabajo es ejemplificar el método DDPG, que puede ser extendido a cualquier número de variables de entrada.

Adicional a las variables representando el estado, el problema posee un sistema de episodios, en el cual se llevan a cabo una serie de pasos de tiempo, en los cuales se aplica una acción determinada. De acuerdo con la documentación, el número máximo de pasos que pueden tomarse en el sistema son 200, punto en el cual deberá reiniciarse el escenario.

2. Acciones Disponibles

En el problema estudiado, solo se dispone de una acción. Sin embargo, al tratarse de una variable continua, no es posible realizar aproximaciones tradicionales, empleando tablas de estado-acción. En este caso, la acción a tomar representa un torque aplicado en el extremo libre del péndulo, lo que ocasiona un cambio en su aceleración angular, y por ende en su velocidad y posición.

Los valores de salida disponibles para la acción corresponden al intervalo continuo [-2.0, 2.0]. Sin embargo, luego de realizar simulaciones previas sobre el entorno, se concluyó que con el fin de obtener mejores resultados, tanto en la exploración como en la explotación, el torque se restringió a [-1.0, 1.0].

Cabe aclarar que la implementación del modelo permite definir cuantas entradas se requieran, así como acciones a tomar en el entorno, lo cual aumenta su flexibilidad para ser aplicado en otro tipo de problemas, o para incluso ser utilizado en métodos de transfer learning.

3. Función de Recompensa

La recompensa obtenida en cada uno de los estados está definida con base en el ángulo formado con respecto a la vertical, y el torque que se debió aplicar para lograr para llegar a dicha posición. Así, el agente recibe una recompensa de 0 cuando el ángulo es cero y el torque aplicado es mínimo. La mínima recompensa que se puede obtener por estado es -16.27 de acuerdo con la documentación, y no se obtienen valores positivos durante el desarrollo. De esta manera, se hace importante resaltar que los valores que se obtendrán de recompensa serán muy bajos en etapas tempranas del entrenamiento, y deben tender a 0 conforme aumentan los episodios.

C. Preparación de los datos

Con el fin de poder implementar el algoritmo, es necesario realizar una serie de preparaciones, que consisten en la definición del tipo de datos a utilizar, así como las estructuras de datos necesarias para almacenar las características y modelos que serán aplicados al problema.

- Conversión en Tensores: Para facilitar la implementación de las redes neuronales, se utilizará la librería Keras del paquete Tensorflow. Dicha librería trabaja con una estructura de datos conocida como tensores. Por esta razón, la representación de los estados, las acciones y las recompensas deben transformarse de arreglos a tensores.

- Ruido Aleatorio (N): Este será utilizado para promocionar la exploración del agente en el escenario establecido. Para esto, se define una estructura de datos a través de una clase, con la cual se almacenará el ruido anterior y se generarán ruidos cada vez que se llame la instancia de la clase.

- Memoria de Observaciones (R): Consiste en una estructura de datos capaz de almacenar 4 arreglos de tensores, representando los estados actuales, las acciones tomadas en dicho estado, la recompensa observada al ejecutar dichas acciones, y la representación del estado siguiente, obtenido al aplicar la acción.

- Creación de Redes Neuronales: Para la creación de las redes neuronales, se hace uso de un modelo de Keras, el cual recibe tensores como entrada en cada una de las capas

de las redes. Además, esto permite definir gradientes personalizados, que puedan ser ajustables a diferentes tipos de implementaciones.

Cabe resaltar que en la preparación de los datos, también se incluye la inicialización del entorno de entrenamiento, utilizando la librería gym, la cual permitirá obtener las variables anteriormente mencionadas, para luego ser almacenadas como tensores en la memoria de observaciones.

D. Modelado

Para el modelado y solución del problema, se hace uso del método DDPG, el cual consiste en implementar redes neuronales para la estimación de las acciones *continuas* dado un estado, representando la *política* (μ), así como para calcular la función de valor acción-estado (Q), que da el nombre a Q-Learning, conocido como un método de aprendizaje profundo. Además, utiliza lo que se conoce como el el Gradiente de Política Determinista para la aproximación de la política óptima. Cabe resaltar que el modelo es determinista porque para cada par estado-acción (s_t, a_t), se obtiene siempre el mismo resultado al aplicar $\mu(s_t)$ y $Q(s_{t,t})$.

1. Política

En este caso, se utiliza una política determinista para determinar la acción a_t a tomar dado un estado s_t. Esto implica que para el estado s_i, se corresponde siempre la misma acción a_i. En términos de la probabilidad que determina el **Proceso de Decisión Markoviano**, se tiene que:

$$P\{A_t = a_t | S_t = s_t\} = 1$$

Esto asegura la propiedad de Markov, haciendo que cada una de las transiciones dependan únicamente del estado inmediatamente anterior. La política está representada por una red neuronal, que recibe la representación continua del estado, que puede estar representado por cualquier cantidad de variables. Su salida será también un vector de valores continuos, representando el valor de cada una de las acciones a tomar. Esta red, denotada por $\mu(s_t)$, recibe el nombre de **Actor**. La tasa de aprendizaje para esta red se define como 0.001.

Su arquitectura está definida de la siguiente manera:

- **Entradas:** Capa de tamaño igual al número de variables que representa el estado. Para este caso, se tienen 4 variables.
- **Capas ocultas:** 2 capas ocultas con un determinado número de unidades cada una. Los parámetros de ambas capas se inicializan con una distribución uniforme $U(-\frac{1}{\sqrt{f_i}}, \frac{1}{\sqrt{f_i}})$, donde f_i es el número de unidades. La función de activación de ambas es *ReLu*. En este caso, se definen 256 unidades para cada capa oculta.
- **Capa de salida:** Capa con número de unidades igual al número de acciones posibles que se pueden tomar en el entorno. Su inicialización se realiza con una distribución uniforme $U(-0{,}003, 0{,}003)$ con el fin de obtener salidas cercanas a cero en las primeras etapas del entrenamiento. Su función de activación es *tanh*, de tal manera que se puedan obtener acciones positivas y negativas. En este caso, la salida consiste de solo una acción, por lo tanto es una sola unidad.

2. Estrategia de Exploración-Explotación

Dado que las acciones son continuas, no es recomendable utilizar un método como el ϵ-greedy para el problema de exploración y explotación, dado que sería muy exhaustivo encontrar la acción que da el máximo valor Q (esto implicaría evaluar el espacio infinito de acciones). Por esta razón, se utilizan 2 estrategias para el entrenamiento de la red:

1. Se utiliza un ruido aleatorio N, el cual afecta las acciones seleccionadas en una determinada cantidad que cambia en cada estado, con el fin de promover la exploración del agente.
2. Se utiliza una memoria R para almacenar los estados y recompensas observados, y las redes neuronales se entrenan tomando muestras aleatorias de dicha memoria. Esto facilita la explotación de las experiencias pasadas que otorgan una alta función de valor.

3. Función de Valor

En el algoritmo propuesto, se utiliza una función de valor **estado-acción**, lo cual es la base del Q-Learning. Dado que se tienen entradas continuas, se emplea una red neuronal que permita el cálculo del valor de la acción tomada en un determinado estado, Esta red

se denota por $Q(s_t, a_t)$ y es denominada **Crítica**, dado que actúa como la evaluadora de la acción tomada. La tasa de aprendizaje de esta red se establece en 0.002.

Su arquitectura es la siguiente:

- **Entradas del Estado:** Capa que recibe las variables que representan el estado, en este caso, 4 unidades.
- **Capas Ocultas:** 2 capas ocultas con un número determinado de unidades cada una. Los parámetros de ambas capas se inicializan con una distribución uniforme $U(-\frac{1}{\sqrt{(f_i)}}, \frac{1}{\sqrt{(f_i)}})$, donde f_i es el número de unidades. La función de activación de ambas es *ReLu*. En este caso, se utilizan 16 y 32 unidades respectivamente.
- **Entradas de la Acción:** Luego de procesar el estado con las capas anteriores, se agrega la entrada de las acciones, las cuales se determinan usando la red Actor. Estas acciones se pasan por una capa de 32 unidades, a las cuales se les aplica una función de activación *tanh*.
- **Capa de Concatenación:** Concatena la salida de la segunda capa oculta y la entrada de las acciones, para un total de 64 unidades representando el par estado-acción.
- **Capa de salida:** Capa con 1 sola salida, representando el valor Q para la combinación estado-acción alimentada. Su función de activación es *ReLu*, con lo cual la función de valor es siempre positiva.

4. Ajuste de los Parámetros

En los métodos tradicionales de aprendizaje profundo, se utiliza la misma red neuronal de la política y la función de valor para el cálculo del error, tomando la diferencia entre resultados anteriores y los resultados actuales. Sin embargo, con el fin de obtener consistencia en los resultados y evitar mínimos locales, se utilizan dos redes adicionales para el cálculo de las funciones de pérdida y optimización de los parámetros. Estas redes se definen como redes **Objetivo**, y se denotan como $\mu^*(s_t)$ para la política, y $Q^*(s_t, a_t)$ para la función de valor estado-acción.

Dado que se cumple la propiedad de Markov, es posible escribir la *Ecuación de Bellman* para la función de valor, como la recompensa esperada al aplicar la acción a_t en el estado s_t,

y seguir la política en el estado inmediatamente posterior. Así, se tiene la siguiente expresión para la recompensa futura:

$$Q^{\mu}(s_t, a_t) = E\big[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1})\big]$$

que representa el valor esperado, donde r es la recompensa obtenida al aplicar la acción a_t en el estado s_t, y γ el parámetro de descuento para las recompensas futuras, el cual se define como 0.99. Se busca entonces minimizar la pérdida representada por la función de valor evaluada en el estado actual, con respecto a la recompensa esperada en el siguiente estado, al variar los parámetros de la red. Sin embargo, la recompensa esperada es estimada con las redes objetivo con el fin de obtener consistencia.

Cabe resaltar que los estados y acciones que son alimentados a las redes para obtener estos resultados, son obtenidos de muestrear aleatoriamente la memoria de observaciones. En este caso, se tomaron lotes con 64 observaciones para el entrenamiento.

Finalmente, los parámetros de las redes Objetivos son encogidos suavemente hacia los parámetros de las redes Actor y Crítica, con un parámetro τ cercano a cero, en este caso, de 0,01.

5. Algoritmo

El algoritmo de aprendizaje es como sigue, de acuerdo con el artículo en el que se describe [3]:

1. Definir redes neuronales Actor $\mu(s|\theta^{\mu})$ y Crítica $Q(a, s|\theta^{Q})$ con parámetros aleatorios θ^{μ} y θ^{Q}
2. Inicializar redes objetivo μ^* y Q^* con parámetros $\theta^{\mu^*} \leftarrow \theta^{\mu^*}$ y $\theta^{Q^*} \leftarrow \theta^{Q}$
3. Crear memoria de observaciones R Definir un número de episodios. En cada episodio, realizar los siguientes pasos:
    1. Inicializar el ruido aleatorio N para la exploración
    2. Obtener el estado inicial s_0 Definir un número de iteraciones t y ejecutar los siguientes pasos hasta la iteración T:

1. Seleccionar una acción $a_t = \mu(s_t|\theta^\mu) + N_t$ de acuerdo con la política actual y el ruido de exploración
2. Ejecutar la acción a_t y obtener la recompensa r_t y el nuevo estado s_{t+1}
3. Guardar la observación (s_t, a_t, r_t, s_{t+1}) en R
4. Seleccionar un lote de n observaciones (s_i, a_i, r_i, s_{i+1}) de R. Se puede definir tomar observaciones solo cuando el número de datos en la memoria superan al tamaño del lote.
5. Calcular $y_i = r_i + \gamma Q^*(\mu^*(s_{t+1}|\theta^{\mu^*}), s_{t+1}|\theta^{Q^*})$ para cada elemento obtenido de la memoria.
6. Actualizar la red *crítica* minimizando la pérdida $L = \frac{1}{N}\sum_i (y_i - Q(a_i, s_i|\theta^Q))^2$
7. Actualizar la red de la política *actor* utilizando el gradiente de política:

$$\nabla_{\theta^\mu} = \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)\nabla_{\theta^\mu}\mu(s|\theta^\mu)$$

8. Actualizar las redes objetivo de acuerdo con:

$$\theta^{\mu^*} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu^*}$$

$$\theta^{Q^*} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q*}$$

E. Evaluación y Análisis del Modelo

Para la evaluación del modelo, se utilizaron 100 episodios de 200 iteraciones cada uno. En la Figura 2 se encuentra la recompensa promedio de los últimos episodios. Se pueden observar variaciones en los valores de la recompensa, indicando que la exploración del agente está siendo efectiva. Además, se tiene para la mayoría de episodios una recompensa promedio de -1200. En diversos episodios se tienen recompensas menores, que representan la capacidad de exploración del algoritmo con el ruido aleatorio.

A partir de la gráfica, se puede observar que el algoritmo requiere de más episodios de entrenamiento, con el fin de obtener mejores resultados en el problema. Esto se explica debido a que las tasas de aprendizaje son muy bajas, así como el parámetro de actualización τ, debido a que el algoritmo es un método de aprendizaje lento. Para evaluar el rendimiento

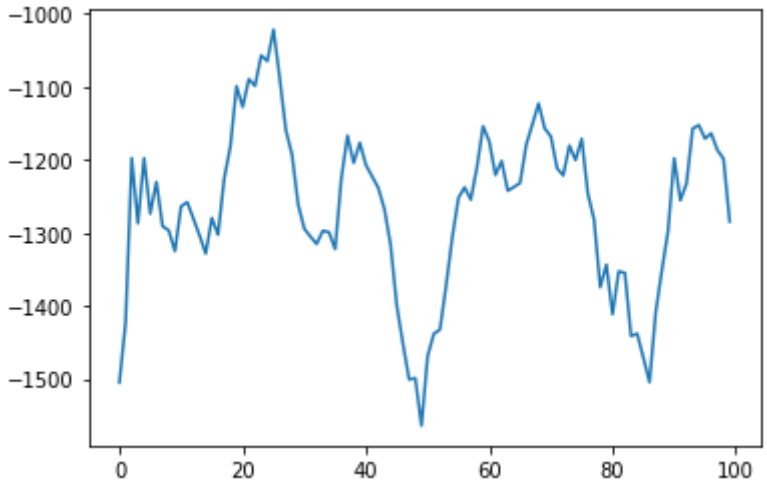

Figura 2: Recompensa promedio de los últimos episodios

del algoritmo sin ruido aleatorio, se llevó a cabo una simulación de 200 iteraciones, para la cual se obtuvo una recompensa total de -1311.80. Esto quiere decir que el algoritmo debe ser o bien entrenado una mayor cantidad de tiempo con el fin de mejorar el desempeño obtenido en el problema, o realizar una optimización de parámetros que permita obtener una mejor recompensa en cada uno de los episodios. Para esto, se realizó un análisis de sensibilidad, para el cual los resultados se muestran a continuación.

F. Análisis de Sensibilidad

1. Tasa de Aprendizaje

En primer lugar, se realizó una variación de las tasas de aprendizaje del modelo. En este caso, se definieron como 0.1 y 0.2 para las redes Actor y Crítica respectivamente. En la Figura 3, se puede apreciar la evolución de la recompensa promedio de los últimos episodios. Se puede apreciar que la recompensa promedio incrementa en tendencia, lo cual muestra que para este problema, es preferible una tasa de aprendizaje mayor. Sin embargo, se debe tener cuidado de no incrementar mucho su valor, dado que puede ocasionar divergencia en la aproximación de la función de valor a su óptimo.

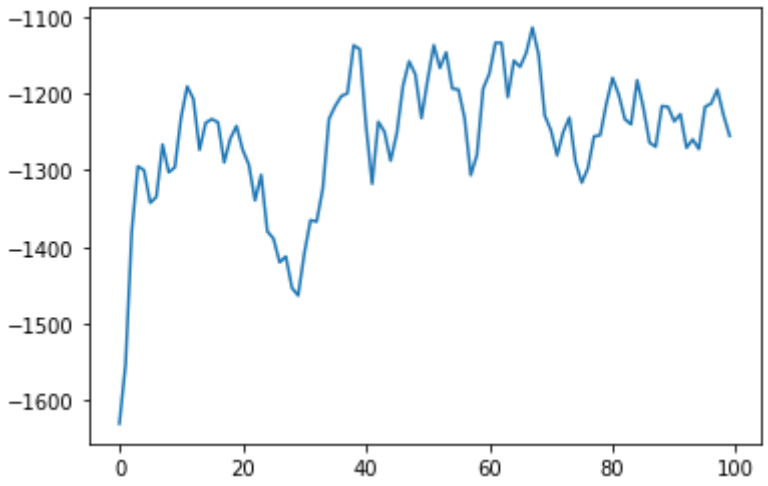

Figura 3: Recompensa promedio de los últimos episodios, con tasa de aprendizaje mayor

2. Tasa de Descuento de la Recompensa

Para este análisis, se retornan las tasas de aprendizaje originales, y se modifica el parámetro γ, que representa el descuento de las recompensas futuras. En este caso, se le asigna un valor de 0.6, y los resultados pueden observarse en la Figura 4. Se puede apreciar que la recompensa por episodio tiene mayores variaciones, mostrando así que darle un menor peso a las recompensas inmediatas afecta negativamente el desempeño del modelo. Por ende, se debe mantener el descuento tal alto como sea posible.

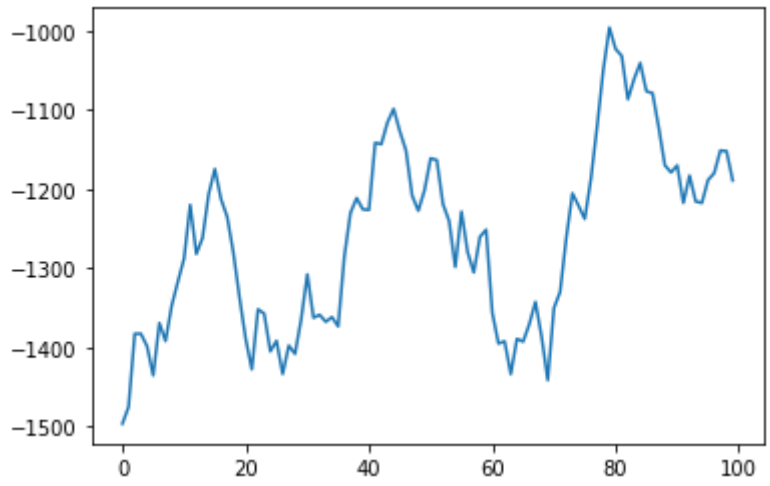

Figura 4: Recompensa promedio de los últimos episodios, con mayor tasa de descuento de recompensa

3. Tasa de Actualización de las Redes Objetivo

El último parámetro a modificar consiste en la tasa de actualización de las redes objetivo. Este es el parámetro más crítico, pues es el que más fácilmente puede introducir inconsistencias y mínimos locales en el ajuste del modelo. Los resultados se pueden apreciar en la Figura 5, en los que se observa gran variabilidad en los resultados, especialmente en las etapas tempranas de entrenamiento. Esto demuestra que la exploración del agente se ve afectada por la estimación errónea de funciones de valor, con lo cual los parámetros de las redes neuronales tienden a cambiar muy rápidamente. Por esta razón, se determina que es mejor tener un valor bajo de este parámetro.

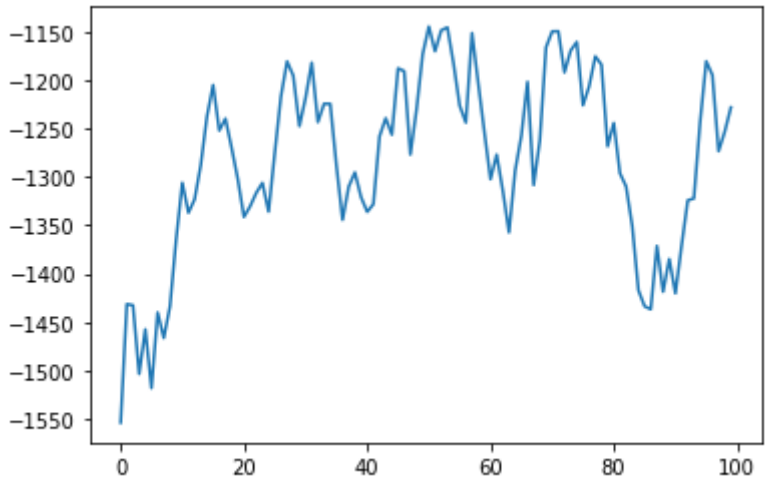

Figura 5: Recompensa promedio de los últimos episodios, con mayor tasa de actualización de las redes objetivo

III. CONCLUSIONES

- Se aplicó el método de aprendizaje reforzado DDPG para el sistema de péndulo invertido, con el fin de implementar un sistema de control dinámico, para el cual no se necesitó una representación matemática del entorno, sino la aplicación de redes neuronales para simular su comportamiento.
- Se determinó que el algoritmo funciona mejor cuando se tienen parámetros de aprendizaje lento, debido a las grandes variaciones que se obtienen en las funciones de valor.

Por ende, es preferible mantener la consistencia de los resultados, aunque el entrenamiento del modelo tome más tiempo.

- El modelo aproxima una política determinista con entradas y salidas continuas, y presenta gran flexibilidad dado que el número de variables se puede modificar sin necesidad de mayores cambios a la implementación. Esto implica además que pueden tenerse entornos representados por una gran cantidad de variables, así como aquellos en los que se tengan gran cantidad de acciones posibles, que pueden o no ejecutarse en simultáneo.

- Otras aproximaciones y variaciones del modelo pueden incluir la modificación de la arquitectura de la red (número de capas, funciones de activación, etc.), el tamaño de la memoria, la representación del estado (utilizar observaciones anteriores como parte de las variables) y las funciones del gradiente.

IV. REFERENCIAS

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (The MIT Press, 2018), 2nd ed.

[2] C. D. Jayaweera and M. Narayana, Chemical Engineering Journal **426**, 130797 (2021), ISSN 1385-8947.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2015).