



INGENIERÍA EN INFORMÁTICA

21 de Diciembre 2020

**Esquema Actor-Crítico en aprendizaje por  
refuerzo con espacios continuos**  
Proyecto Final

Francisco Delgado  
57101

---

# Índice

<b>1. Resumen</b>	<b>3</b>
<b>2. Introducción</b>	<b>4</b>
<b>3. Fundamentos</b>	<b>6</b>
3.1. Procesos de decisión de Markov . . . . .	6
3.2. La interfaz Agente - Ambiente . . . . .	6
3.3. Función de valor . . . . .	7
3.4. Ecuaciones de Bellman . . . . .	8
3.5. Políticas . . . . .	8
3.6. Espacio de estados . . . . .	8
3.7. Función de Ventaja . . . . .	9
<b>4. Q-Learning tabular</b>	<b>10</b>
4.1. Q-Learning . . . . .	10
4.2. Caso de estudio: Péndulo invertido . . . . .	11
4.3. Discretización del espacio de estados . . . . .	12
4.4. Resultados . . . . .	12
<b>5. Espacio de estados continuos de alta dimensionalidad</b>	<b>15</b>
5.1. Métodos de gradientes de política . . . . .	15
5.2. Teorema del gradiente de la política . . . . .	16
5.3. Algoritmo REINFORCE . . . . .	17
5.4. Inicialización de los pesos de la red . . . . .	18
5.5. Componentes de la red . . . . .	18
5.5.1. Función softmax . . . . .	18
5.5.2. Capa softmax . . . . .	19
5.5.3. El gradiente de la función softmax . . . . .	19
5.5.4. Derivada respecto a los pesos y sesgos . . . . .	21
5.6. Optimizadores . . . . .	21
5.6.1. Momentum . . . . .	22
5.6.2. RMSProp . . . . .	22
5.6.3. ADAM . . . . .	23
<b>6. Métodos de Actor Crítico</b>	<b>24</b>
6.1. REINFORCE con valor base . . . . .	24
6.1.1. Introduciendo el valor base . . . . .	25
6.2. Actor-Crítico de un paso . . . . .	26
6.3. Proximal Policy Optimization . . . . .	26
6.3.1. Métodos de gradiente de política . . . . .	27
6.3.2. Métodos de región de confianza . . . . .	27
6.3.3. Objetivo Recortado . . . . .	28
6.3.4. Algoritmo completo . . . . .	28
6.4. Funciones de pérdida y gradientes . . . . .	28

6.4.1.	Función de pérdida y gradiente del Actor . . . . .	28
6.5.	Crítico . . . . .	29
<b>7.</b>	<b>Resultados Experimentales</b>	<b>30</b>
7.1.	Optimizadores y su impacto en el aprendizaje . . . . .	30
7.2.	REINFORCE vs. PPO . . . . .	33
7.2.1.	Curvas de aprendizaje . . . . .	33
7.2.2.	Comparación de la entropía . . . . .	33
7.3.	Aplicando PPO a un problema complejo . . . . .	37
7.3.1.	Agente . . . . .	37
7.3.2.	Representación del estado . . . . .	38
7.3.3.	Función de Refuerzo . . . . .	38
7.3.4.	Progreso del entrenamiento . . . . .	39
7.3.5.	Análisis de la entropía . . . . .	39
7.3.6.	Explotando el agente . . . . .	40
<b>8.</b>	<b>Conclusiones</b>	<b>42</b>
	<b>Referencias</b>	<b>43</b>

# 1. Resumen

El siguiente informe es el resultado del trabajo de relevamiento, estudio y desarrollo de métodos de aprendizaje por refuerzo principalmente enfocados al esquema Actor-Crítico. A lo largo del texto, se introducen los conceptos base de la teoría y se plasman los fundamentos y teoría no solo de el esquema Actor-Crítico, sino también su predecesor REINFORCE, donde se introduce el concepto de optimización de políticas. El esquema Actor-Crítico es instanciado mediante la implementación de Proximal Policy Optimization (PPO) el cual utiliza redes neuronales artificiales (RNA) como estimador de función no lineal. Para el ajuste de los pesos de las RNAs se evaluaron tres variantes del gradiente estocástico las cuales incluyen Momentum, RMSProp y ADAM. Se diseñaron una serie de experimentos con el propósito de comparar el esquema Actor-Crítico con REINFORCE y sus resultados permiten establecer las diferencias de eficiencia entre ambos. Estos experimentos se hacen en base al problema del péndulo invertido en un ambiente dinámico. Finalmente, para demostrar la robustez y flexibilidad de los métodos de Actor-Crítico toma como caso de estudio un problema de control complejo en donde un agente debe aprender a caminar. Dicho agente es una criatura artificial semejante a una hormiga de cuatro patas, con dos rotores por pata que aplican torque sobre las mismas. La evaluación del desempeño del agente se realiza midiendo la longitud de la trayectoria recorrida hacia un objetivo en una cantidad de tiempo.

## 2. Introducción

El aprendizaje automático es el campo de estudio que trata con algoritmos que mejoran su desempeño a partir de la experiencia. Este se subdivide tradicionalmente en tres categorías.

- **Aprendizaje supervisado** El algoritmo debe aprender a asociar entradas con salidas deseadas, ambas especificadas en un conjunto de entrenamiento. El objetivo del método consiste en, después de entrenado, poder predecir salidas correctas ante entradas que no fueron observadas anteriormente. Esto se conoce como generalización.
- **Aprendizaje no supervisado** A diferencia del aprendizaje supervisado este no posee en su entrenamiento las salidas correctas por lo cual debe aprender por su cuenta la estructura de las entradas. Este tipo de aprendizaje es muy utilizado en técnicas de descubrimiento buscando patrones escondidos en los datos, como también para extraer características de los mismos. Esto se conoce como aprendizaje de características.
- **Aprendizaje por refuerzo** Estos algoritmos se pueden interpretar como agentes que aprenden a tomar secuencias de acciones buscando maximizar una recompensa a lo largo del tiempo. El aprendizaje, en estos algoritmos se basa en la exploración, en donde este toma acciones en busca de nuevo conocimiento, y explotación, en donde el agente busca utilizar el conocimiento que obtuvo hasta el momento para tomar la acción que estima le dará la mayor recompensa a futuro.

En este trabajo se aplicará el aprendizaje por refuerzo. El ambiente dinámico es implementado con el OpenAI Gym[6], el simulador de física PyBullet[8] y la integración entre PyBullet y OpenAI Gym, PyBullet Gymperium[7]. Llamaremos Gym de ahora en adelante a la colección de estos ambientes y simuladores.

Estos ambientes, como simulaciones dinámicas nos proveen una buena aproximación a la realidad de manera tal que los resultados obtenidos a partir del aprendizaje por refuerzo sean una aproximación para una implementación útil de robots concretos. Concretamente Gym nos provee una Interfaz (API) de manera tal que podemos concentrarnos en la implementación del agente. Gym nos permite obtener el efecto de las acciones dentro del sistema físico y el valor de la función de refuerzo ajustada para el ambiente de manera tal que el agente(robot) aprenda la tarea deseada.

Cuando un agente aprende a realizar una tarea lo que está aprendiendo es una política, esto es, saber qué acción debe tomar para cada estado del ambiente. El conocimiento adquirido por el agente durante su aprendizaje puede ser almacenado en forma de tabla o usando un aproximador. Por ejemplo, un aproximador puede ser usado para aproximar la política aprendida hasta el momento y a medida que mejore la política aprendida, ajuste

los parámetros del aproximador para memorizarla. Particularmente los métodos de aproximación de políticas serán implementados con el uso de redes neuronales artificiales (RNA) como aproximadores de funciones no lineales. A lo largo de este proyecto, todas las RNAs utilizadas fueron integralmente codificadas en Python y solamente se usaron librerías para multiplicación y manipulación de matrices. Resulta especialmente interesante analizar como los distintos optimizadores y estructuras de red impactan en el aprendizaje del agente mas allá del algoritmo de aprendizaje por refuerzo que se utiliza.

Este informe se encuentra estructurado de la siguiente forma. En la sección 3 se encuentran los fundamentos del aprendizaje por refuerzo. En la sección 4, se encuentra la introducción a Q-Learning y es utilizado para entrenar un agente que debe controlar un péndulo invertido. Aquí se detalla el problema que se busca resolver y posteriormente se analizan los resultados experimentales del aprendizaje con Q-Learning. La sección 5 introduce el aprendizaje por refuerzo en espacios de estados continuos. Para esto se ven los métodos de aproximación de políticas y el método REINFORCE. En la sección 6 se encuentra la teoría de los agentes Actor - Crítico junto con el método que se utilizará concretamente, Proximal Policy Optimization (PPO). La sección 7 muestra los resultados experimentales que se obtuvieron para dos agentes diferentes que deben resolver problemas y ambientes distintos. Se analiza aquí el impacto que tienen las distintas implementaciones de las RNA en el aprendizaje de los agentes. Finalmente tendremos las conclusiones del trabajo en la sección 8, donde se resumen los resultados principales y consideran posibles trabajos futuros.

### 3. Fundamentos

Para entender el problema del aprendizaje por refuerzo debemos entender la teoría fundamental que lo compone. En esta sección veremos todas las definiciones y teoría necesaria para entender el trabajo realizado.

#### 3.1. Procesos de decisión de Markov

El proceso de decisión de Markov (PDM) es una formalización matemática del proceso de decisión secuencial, en donde se tiene un agente que realiza acciones dentro de un ambiente. En base a sus acciones el ambiente cambia y el agente percibe estos cambios mediante cambios en el estado y la recompensa que recibe. Las acciones no solo influyen las recompensas inmediatas sino que también influyen los estados subsiguientes y a través de estos las recompensas futuras.

#### 3.2. La interfaz Agente - Ambiente

Los PDM proveen un marco simple al problema de aprender de la interacción para lograr un objetivo. El ente que aprende y toma decisiones se le llama el agente. Mientras que el medio con el que interactúa, que involucra todo lo que está por fuera del agente, se le llama el ambiente.

El agente y el ambiente interactúan constantemente. El agente toma acciones mientras que el ambiente responde a estas presentándole nuevas situaciones al agente. El ambiente además le provee al agente recompensas por sus acciones. Estas recompensas son valores numéricos que el agente tratará de maximizar en el tiempo mediante su elección de acciones en cada estado.

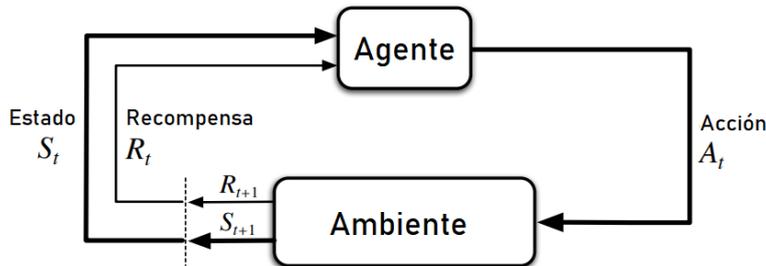


Figura 1: Ciclo de interacción entre el agente y el ambiente. Extraído de [5].

Más específicamente, el agente y el ambiente interactúan en una secuencia de pasos temporales discretos  $t = 0, 1, 2, 3, \dots$ . En cada paso temporal el agente tiene una representación del estado  $S_t \in \mathcal{S}$  donde  $\mathcal{S}$  representa el espacio de estados posibles para el ambiente. En base a este, selecciona una acción  $A_t \in \mathcal{A}(S_t)$  donde  $\mathcal{A}(S_t)$  es el espacio de acciones dado el estado  $S_t$ . En el paso siguiente el agente recibe la recompensa  $R_{t+1} \in \mathcal{R} \subset \mathbf{R}$  donde  $\mathcal{R}$  es el espacio de recompensas y este está contenido por los reales. Finalmente el

agente se encuentra en un nuevo estado  $S_{t+1}$ . Entonces el agente y el ambiente realizan una secuencia de estado acción recompensa llamada trayectoria y se denota por la letra  $\tau$ :

$$\tau = S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \dots \quad (3.1)$$

Anteriormente se dijo que el agente busca maximizar las recompensas tanto presentes como futuras. Esto se encapsula en el concepto de retorno, el cual es la suma descontada de recompensas que recibe un agente a lo largo de toda la trayectoria de un episodio:

$$G_t = \sum_{i=1}^n \gamma^i R_{i+1}(S_i, a_i), \quad (3.2)$$

donde  $\gamma$  es el factor de descuento de manera tal que las recompensas más lejanas pesan menos que las recompensas mas cercanas dado que  $0 \leq \gamma < 1$ .

### 3.3. Función de valor

Las funciones de valor son tales que tienen como entrada estados o pares acción-estado y estiman cuan bueno es para un agente estar en un estado determinado (o que tan bueno es realizar una acción en un estado dado). La noción de bondad del estado esta definida en términos de la recompensa futura que recibirá el agente, lo cual es precisamente el retorno.

Más formalmente, la función de valor de un estado  $s$  bajo una política  $\pi$ , definido como  $V_\pi(s)$ , es el retorno esperado cuando se arranca en  $s$  y se sigue la política  $\pi$  de ese estado en adelante. Para un PDM se puede definir formalmente como:

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s], \text{ para todo } s \in \mathcal{S}, \quad (3.3)$$

donde  $\mathbb{E}_\pi[\cdot]$  denota el valor esperado de una variable aleatoria dado que el agente sigue la política  $\pi$  y  $t$  es un paso dado. A esta función se la denomina *función de valor-estado bajo  $\pi$* .

Similarmente podemos definir el valor de tomar la acción  $a$  en el estado  $s$  y luego continuar con la política  $\pi$ , denotado como  $Q_\pi(s, a)$ :

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a], \text{ para todo } s \in \mathcal{S}, a \in \mathcal{A}. \quad (3.4)$$

A esta función se la denomina *función de valor estado-acción bajo  $\pi$* .

Teniendo estas dos definiciones, podemos definir  $V_\pi(s)$  en base a  $Q_\pi(s, a)$  como:

$$V_\pi(s) = \mathbb{E}_{a \sim \pi(s)} [Q_\pi(s, a)]. \quad (3.5)$$

### 3.4. Ecuaciones de Bellman

Bellman demostró que un problema de optimización dinámica en tiempo discreto puede ser descrito de manera recursiva en una forma conocida como inducción invertida[2]. Se hace escribiendo la función de valor en un período dependiendo solamente del período posterior. En el caso de aprendizaje por refuerzo sabemos que el valor de un estado  $s$  es la suma del refuerzo recibido en  $s$  por ejecutar una acción  $a$  más el valor del estado que sigue  $s'$ :

$$V_{\pi}(s) = \mathbb{E}_{\substack{a \sim \pi(s) \\ s', r \sim p(s, a)}} [r + \gamma V_{\pi}(s')] \quad (3.6)$$

donde  $a$  es la acción que se toma siguiendo la política  $\pi$ ,  $r$  es la recompensa que se recibe por realizar dicha acción dictada por la función de transición  $p(s', r | s, a)$ .  $\gamma$  es el factor de descuento.

De manera análoga se define para  $Q$

$$Q_{\pi}(s, a) = \mathbb{E}_{s', r \sim p(s, a)} \left[ r + \gamma \mathbb{E}_{a' \sim \pi(s')} [Q_{\pi}(s', a')] \right]. \quad (3.7)$$

### 3.5. Políticas

La política determina como el agente elige una acción dado que se encuentra en un determinado estado. Estas pueden ser tanto deterministas como estocásticas. Cuando éstas son deterministas se encuentran representadas por la letra  $\mu$

$$a = \mu(s), \quad (3.8)$$

y cuando son estocásticas se representan por la letra  $\pi$ :

$$a = \pi(s, a) \quad (3.9)$$

Esta asocia estados a la probabilidad de tomar cada acción posible. Si el agente sigue la política  $\pi$  en un tiempo  $t$  entonces  $\pi(s, a)$  es la probabilidad que  $A_t = a$  si  $S_t = s$ .

### 3.6. Espacio de estados

Los estados se encuentran restringidos a un determinado espacio definido por el ambiente. Este espacio puede ser tanto finito como infinito, sin embargo cuando los espacios de estados son infinitos estos no podrán ser abordados por los métodos tabulares ya que significaría una tabla infinita. En este caso y como veremos en el trabajo realizado se debe discretizar el espacio de estados de tal manera que quepa razonablemente en la tabla del agente.

### 3.7. Función de Ventaja

La ventaja es un concepto que permite evaluar el desempeño de una acción relativo al estado en el cual se esta tomando la acción. Esta da una medida de cuan mejor o peor es la acción respecto a las acciones disponibles en ese estado. Esto se cuantifica en la función de ventaja  $A_\pi : (\mathcal{S}, \mathcal{A}) \rightarrow \mathbb{R}$ .

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s). \quad (3.10)$$

## 4. Q-Learning tabular

El primer método de aprendizaje por refuerzo que se implementó fue Q-Learning. Consiste en aproximar la función de estado valor  $Q(s, a)$  mediante el uso de una tabla. A continuación se presenta en detalle como se implementó Q-Learning y un ejemplo puntual del algoritmo.

### 4.1. Q-Learning

Este es un método tabular de aprendizaje por refuerzo que consiste en mantener una tabla Q la cual contiene una estimación del valor de  $Q(s, a)$  para cada posible valor de estado y acción. Al tener que estimar  $Q(s, a)$  se lo conoce como un método basado en funciones de valor. Utilizado convencionalmente se ve inmediatamente que tanto el espacio de estados como el espacio de acciones debe ser discreto, ya que todos sus posibles valores deben entrar en una tabla.

La derivación fundamental de Q-Learning parte de la base de que el valor de  $Q(s, a)$  es el valor esperado del retorno  $G$ . Esto es por definición de la función de valor Q. Esto significa que puede ser estimado como el promedio de los retornos de las trayectorias luego de tomar la acción  $a$  en el estado  $s$ :

$$Q_{n+1}(s, a) = \frac{1}{n} \sum_{i=1}^n G_i(s, a), \quad (4.1)$$

donde  $G_i$  es el retorno de tomar la acción  $a$  en el estado  $s$  en la trayectoria  $i$ .

Ahora bien, vemos que el valor de  $Q_{n+1}$  puede ser determinado a partir de su paso anterior,  $Q_n$  y  $G_n$ . Esto resulta conveniente ya que presenta un algoritmo que no necesita tener conocimiento de los pasos anteriores.

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n G_i \\ &= \frac{1}{n} \left( G_n + \sum_{i=1}^{n-1} G_i \right) \\ &= \frac{1}{n} \left( G_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} G_i \right) \\ &= \frac{1}{n} (G_n + (n-1)Q_n) \\ &= \frac{1}{n} (G_n + nQ_n - Q_n) \\ &= Q_n + \frac{1}{n} (G_n - Q_n). \end{aligned} \quad (4.2)$$

Observando detenidamente la ecuación tiene la estructura básica de:

$$NuevaEst. = ViejaEst. + Paso[Objetivo - ViejaEst.], \quad (4.3)$$

Se interpreta entonces a  $[Objetivo - ViejaEst.]$  como el error que se cometió en la estimación, este error es aplicado tomando un paso de tamaño *paso* hacia el *Objetivo*. El *Objetivo* se presume una dirección deseable en la cual se debe mover el agente, aunque puede ser ruidoso.

Prestando atención a la expresión derivada en (4.2) queda claro como se están valorando de la misma manera las recompensas de pasos temporales lejanos como cercanos. En la práctica los objetivos pueden no ser estacionarios, por lo que optamos por una recompensa descontada en donde el peso que se le da a las recompensas lejanas no es el mismo que se le da a las cercanas del paso temporal en cuestión. Esto se logra mediante un tamaño de paso constante en vez de usar uno dependiente de  $n$ , como lo es  $1/n$ . Cuando usamos un paso constante obtenemos el efecto anteriormente mencionado como se ve en el siguiente desarrollo:

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha[G_n - Q_n] \\ &= \alpha G_n + (1 - \alpha)Q_n \\ &= \alpha G_n + (1 - \alpha)[\alpha G_{n-1} + (1 - \alpha)Q_{n-1}] \\ &= \alpha G_n + (1 - \alpha)\alpha G_{n-1} + (1 - \alpha)^2 Q_{n-1} \\ &= \alpha G_n + (1 - \alpha)\alpha G_{n-1} + (1 - \alpha)^2 \alpha G_{n-2} + \\ &\quad \dots + (1 - \alpha)^{n-1} \alpha G_1 + (1 - \alpha)^n Q_1 \\ &= (1 - \alpha)^n Q_1 + \alpha \sum_{i=1}^n (1 - \alpha)^{n-i} G_i, \end{aligned} \quad (4.4)$$

quedando la recompensa descontada por  $(1 - \alpha)^{n-i}$ .

Con esto se instancia finalmente la regla de actualización de Q-Learning:

$$Q_{n+1}(s, a) = Q_n(s, a) + \alpha(r + \gamma \max_{a'} Q_n(s', a') - Q_n(s, a)) \quad (4.5)$$

en donde el agente transiciona del estado  $s$  al estado  $s'$  mediante la acción  $a$  y obtiene la recompensa  $r$ .

## 4.2. Caso de estudio: Péndulo invertido

El primer acercamiento al problema de aprendizaje por refuerzo se realizó sobre el problema del péndulo invertido. Este consiste en un ambiente dinámico en donde se debe balancear un palo por encima de un carro como se puede ver en la figura 2.

Tanto el palo como el carro poseen las propiedades de la física dinámica como lo son el rozamiento, la masa y la inercia. El agente posee únicamente

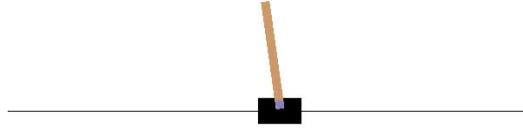


Figura 2: Péndulo invertido

dos acciones. Aplicar fuerza al carro hacia la derecha o la izquierda, y el estado se encuentra representado por un vector de cuatro dimensiones representando: la posición del carro, la velocidad del carro, el ángulo del palo respecto a la vertical y la velocidad angular del palo.

El problema se considera resuelto cuando el agente mantiene el péndulo balanceado por más de doscientos pasos. Si el péndulo excede los  $15^\circ$  de inclinación de la vertical, o el carro se mueve más de 2.4 unidades del centro se considera terminado el episodio.

La función de refuerzo consta en una recompensa de +1 por cada paso del episodio que se ha mantenido sin caerse (ángulo del palo  $< 15^\circ$ ) y sin alejarse a más de 2,4 unidades del centro. Si bien resulta contra intuitiva al principio, entendemos que cuanto más tiempo logre el agente más recompensa obtiene. Por lo tanto los estados que lleven al agente a permanecer por más tiempo equilibrado el palo tendrán un valor mayor que aquellos que lleven el agente a terminar prematuramente el episodio.

### 4.3. Discretización del espacio de estados

Debido a que tenemos dimensiones que constituyen un intervalo de  $\mathbf{R}$  contamos con un espacio de estados infinito, esto nos imposibilita la implementación de un agente tabular y por esto debemos discretizar el ambiente. La velocidad angular del palo y el ángulo se dividirán en 6 y 12 intervalos de igual tamaño respectivamente. El ángulo del palo se tendrá en el intervalo  $[-24, 24]$  grados. La velocidad angular se tendrá en el intervalo  $[-50, 50]$ .

### 4.4. Resultados

Se utilizó un  $\gamma = 1$  y  $\alpha = 0,1$  como hiper parámetros del método. Para entender la evolución del agente a lo largo del aprendizaje, se usó como medida de desempeño el promedio y la desviación estándar de la cantidad de pasos realizados en cada episodio (medido sobre 5 experimentos). A los efectos de suavizar la curva, sobre todo al comienzo del aprendizaje, se utilizó para la gráfica, una ventana deslizante de 30 episodios. Cuantos más pasos

temporales por episodio realiza el agente mejor es su desempeño, recordando que la función de refuerzo es de 1 por cada paso temporal.

La figura 3 muestra la curva de aprendizaje de un agente Q-Learning. Se muestra el promedio y la desviación estándar en el mismo gráfico. En este caso el agente presenta una tendencia creciente del desempeño durante toda el aprendizaje. Se ve que consigue además en el último tramo de su entrenamiento reducir la desviación estándar de tal manera que obtiene consistentemente 200 pasos temporales por episodio, el cual es el máximo por diseño del experimento.

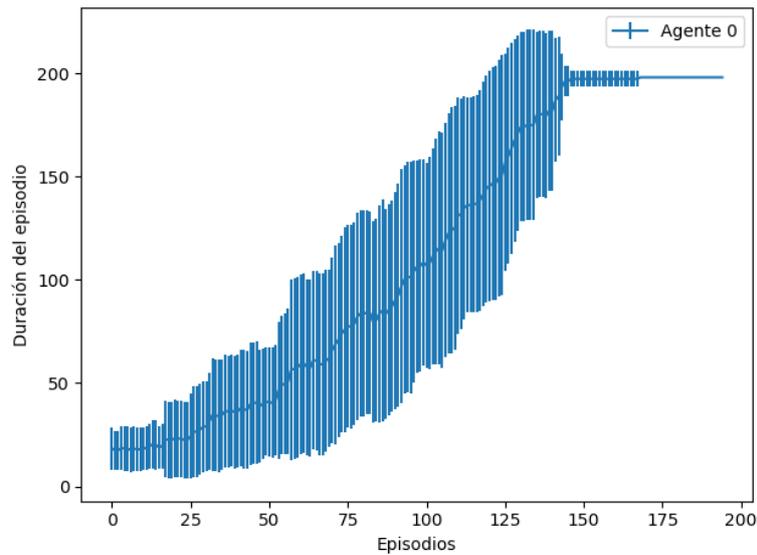


Figura 3: Curva de aprendizaje de un agente Q-Learning con su desviación estándar.

Para verificar el modelo se hicieron cinco corridas distintas del mismo agente Q-Learning tal y como se enunció anteriormente. Viendo la figura 4 se ve claramente como las cinco corridas logran resolver el problema llegando a doscientos pasos temporales por episodio en menos de doscientos episodios.

Q-Learning entonces es capaz de resolver el problema del péndulo invertido consistentemente en menos de doscientos episodios. La discretización del ambiente resultó adecuada también como la elección de los valores de los hiperparámetros.

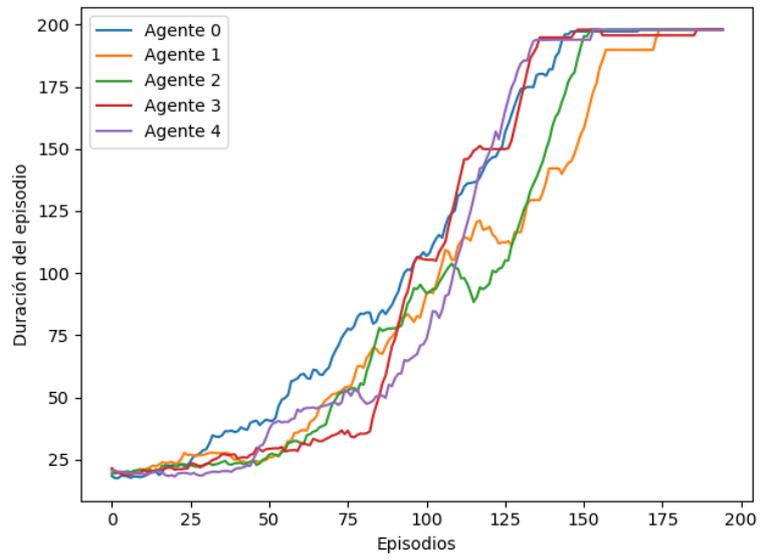


Figura 4: Curvas de aprendizaje para cinco agentes Q-Learning

## 5. Espacio de estados continuos de alta dimensionalidad

Hasta ahora observamos como un agente tabular puede aprender por refuerzo. En el caso estudiado del péndulo invertido, hemos usado la técnica de binning la cual divide el rango completo de valores de cada variable del espacio de estados en intervalos y asigna a cada uno de ellos un valor. Sin embargo, La complejidad de la discretización dependerá del problema, ya que puede resultar muy dificultoso establecer cuales son los anchos de los intervalos adecuados. Un aspecto a tener especialmente en cuenta es la dimensionalidad ya que el agente tabular tendrá  $n^d * a$  entradas donde  $d$  es la dimensión del espacio de estados,  $n$  es la cantidad de intervalos que se toman para cada dimensión y  $a$  es la cantidad de acciones posibles que puede tomar el agente.

Esto es, se tiene una cota espacial la cual es de orden exponencial respecto a la dimensión. En dimensionalidades altas y/o con una alta cantidad de intervalos puede resultar en el fenómeno conocido como explosión combinatoria. Por ejemplo si tenemos dos dimensiones con 60 intervalos tendremos  $60^2 * a = 3600 * a$  entradas. Pero si tenemos 8 dimensiones en cambio, tendremos  $60^8 * a = 1,67 \times 10^{14}$  entradas.

Por esto resulta necesaria la introducción de nuevos métodos distintos a los tabulares capaces de abordar estos problemas más complejos. Un enfoque alternativo al tabular, es el uso de aproximadores parametrizados. Dichos aproximadores pueden ser lineales o no lineales pero en todos los casos, dependen un conjunto de parámetros, notados usualmente con  $\theta$ .

### 5.1. Métodos de gradientes de política

En los métodos del gradiente de política, esta última es aproximada por una función  $F(S) = A$  en donde el dominio de  $F$  es el espacio de estados y cada punto de la imagen  $A$  representa las probabilidades de elegir cada acción dado  $s \in S$ . Esto es,  $F(s) = \mathbf{a}$  donde  $\mathbf{a} \in [0, 1]^{|A|}$ ,  $\mathbf{a} = (\pi(\mathbf{a}_1|s), \pi(\mathbf{a}_2|s), \dots, \pi(\mathbf{a}_{|A|}|s))$  y  $A$  es el conjunto finito de acciones. Como  $F$  esta parametrizada por  $\theta$  entonces, podemos escribir.

$$F(s, \theta) = (\pi(a_1|s), \pi(a_2|s), \dots, \pi(a_{|A|}|s)). \quad (5.1)$$

Los métodos de gradientes de políticas deben aprender una política parametrizada. Estos tienen como espacio de parámetros  $\theta \in R^{d'}$  donde  $d'$  es la dimensión del espacio de parámetros. La probabilidad que una acción  $a$  se tome en un tiempo  $t$  entonces se da por la ecuación:

$$\pi(a | s, \theta) = \Pr \{A_t = a | S_t = s, \theta_t = \theta\} \quad (5.2)$$

Para que sea un método de gradiente necesitamos de una función de desempeño de la política, denotada por  $J(\theta)$ . La función de desempeño es tal

que debe ser maximizada por el agente ya que es una medida cuantitativa del desempeño del agente. Dependiendo del problema se pueden definir también funciones de pérdida o costo. Estas son análogas a las funciones de desempeño excepto por que se busca minimizarlas en vez de maximizarlas, ya que representan una medida de cuan lejos está el agente del objetivo al cual busca llegar. La función de desempeño debe ser tal que se pueda calcular su gradiente para poder maximizarla. Particularmente se utilizaran métodos de gradiente ascendente para obtener los valores de  $\theta$  que maximizan el desempeño  $J(\theta)$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)} \quad (5.3)$$

donde  $\widehat{\nabla J(\boldsymbol{\theta}_t)}$  es un estimador estocástico cuyo valor esperado aproxima el gradiente de la función de desempeño.

## 5.2. Teorema del gradiente de la política

Antes de introducir el primer algoritmo de aproximación de políticas debemos conocer primero el Teorema del gradiente de la política, pues sobre este se basará el desarrollo del algoritmo.

Comenzamos definiendo la función de desempeño del agente como

$$J(\theta) = v_{\pi_\theta}(s_0) \quad (5.4)$$

donde  $s_0$  es el estado inicial del episodio y  $v_{\pi_\theta}$  es la función de valor para la política  $\pi_\theta$ , la cual está parametrizada por  $\theta$ .

El propósito será ir modificando los valores de los parámetros  $\theta$  de tal manera que nos asegure una mejora de la función de desempeño. Si observamos con detenimiento la función de desempeño que se definió en (5.2) ésta dependerá tanto de las acciones que fueron tomadas como de la distribución de estados sobre los cuales se seleccionaron las acciones. Conociendo los parámetros de la política podemos fácilmente conocer su efecto sobre las acciones. Sin embargo, el efecto que tienen los parámetros de la política sobre la distribución de los estados resulta ser una función del ambiente, la cual es desconocida y por lo tanto no la podremos derivar. ¿Como estimamos entonces la función de desempeño si su gradiente depende del efecto desconocido que tienen los cambios de los parámetros de la política en la distribución de los estados?

Para esto emplearemos el teorema del gradiente de la política, el cual define el gradiente de tal manera que no depende de la derivada de la distribución de estados bajo la política parametrizada:

$$\nabla J(\pi_\theta) \propto \sum_s \mu_\pi(s) \sum_a Q_\pi(s, a) \nabla \pi_\theta(a | s) \quad (5.5)$$

donde  $\mu_\pi(s)$  es la distribución de probabilidad de los estados bajo la política  $\pi_\theta$  y  $\propto$  significa "proporcional a".

### 5.3. Algoritmo REINFORCE

A continuación se presenta el primer algoritmo de aproximación de políticas. Recordando que el agente debe aprender una política mediante la ecuación (5.3) utilizamos la ecuación (5.5) que permite calcular el gradiente de la política como nuestro gradiente de la función de desempeño. Observando la ecuación (5.5) vemos que el gradiente es proporcional a una suma pesada por la probabilidad  $\mu_\pi$  del estado  $s$  bajo la política  $\pi$ . Esto es equivalente al valor esperado bajo la política y por lo tanto reemplazamos

$$\begin{aligned} \nabla J(\theta) &\propto \sum_s \mu_\pi(s) \sum_a Q_\pi(s, a) \nabla \pi_\theta(a | s) \\ &= \mathbb{E}_\pi \left[ \sum_a Q_\pi(S_t, a) \nabla \pi_\theta(a | S_t) \right] \text{ donde } S_t \sim \mu_\pi. \end{aligned} \quad (5.6)$$

Introducimos en la ecuación (5.6)  $A_t$  tal como introducimos  $S_t$ , reemplazando la suma sobre los valores posibles de una variable aleatoria por la expectativa de la misma bajo  $\pi$ . La ecuación (5.6) tiene una suma sobre las acciones, pero esta no se encuentra pesada por  $\pi(a|S_t, \theta)$ , tal como lo necesitamos para tener la esperanza de  $\frac{Q_\pi(S_t, a) \nabla \pi_\theta(a|S_t)}{\pi_\theta(a|S_t)}$  bajo  $\pi$ . Para lograr esto, multiplicamos y dividimos cada término de la sumatoria por  $\pi(a|S_t, \theta)$ :

$$\begin{aligned} \nabla J(\theta) &\propto \mathbb{E}_\pi \left[ \sum_a \pi_\theta(a | S_t) Q_\pi(S_t, a) \frac{\nabla \pi_\theta(a | S_t)}{\pi_\theta(a | S_t)} \right] \\ &= \mathbb{E}_\pi \left[ Q_\pi(S_t, A_t) \frac{\nabla \pi_\theta(A_t | S_t)}{\pi_\theta(A_t | S_t)} \right] \text{ donde } A_t \sim \pi_\theta(S_t) \\ &= \mathbb{E}_\pi \left[ G_t \frac{\nabla \pi_\theta(A_t | S_t)}{\pi_\theta(A_t | S_t)} \right] \text{ porque } \mathbb{E}_\pi[G_t | S_t, A_t] = Q_\pi(S_t, A_t) \end{aligned} \quad (5.7)$$

donde  $G_t$  es el retorno según  $A_t$  dado  $S_t$  y  $A_t \sim \pi_\theta(S_t)$ . La ecuación (5.7) nos presenta finalmente un estimador del gradiente de la función de desempeño que puede ser muestreado en cada paso temporal. Con esto finalmente instanciamos el paso de actualización de REINFORCE:

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi_{\theta_t}(A_t | S_t)}{\pi_{\theta_t}(A_t | S_t)}, \quad (5.8)$$

lo cual solamente requiere que la política sea derivable.

Una forma de instanciar el aproximador es mediante el uso de redes neuronales artificiales (RNA). En particular, nosotros usaremos perceptrones multi capa con funciones de activación que pueden ser tanto lineales como no lineales.

Finalmente se instancia el algoritmo como:

Entrada: un aproximador de la política  $\pi(a|s, \theta)$  derivable respecto a theta

Parámetro: tamaño de paso  $\alpha > 0$

Inicializar los parámetros de la política  $\theta$

Para cada episodio:

Generar el episodio  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , siguiendo  $\pi(\cdot | \cdot, \theta)$

Para cada paso del episodio  $t = 0, 1, \dots, T - 1$  :

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

$$\theta \leftarrow \theta + \alpha G \nabla \ln \pi(A_t | S_t, \theta)$$

dado que

$$\frac{\nabla \pi_{\theta_t}(A_t | S_t)}{\pi_{\theta_t}(A_t | S_t)} = \ln \nabla \pi_{\theta_t}(A_t | S_t). \quad (5.9)$$

## 5.4. Inicialización de los pesos de la red

Para inicializar los pesos se utiliza una estrategia fan-in en donde cada peso es inicializado a  $d * x$  donde  $x \sim \mathcal{N}(0, 1)$  y  $d$  es la cantidad de neuronas en la capa anterior.

## 5.5. Componentes de la red

Como vemos en la ecuación (5.8) requerimos de un aproximador que dado un estado obtengamos las probabilidades de las acciones. Esto significa que cada salida de la RNA pueda ser interpretada como la probabilidad de tomar una acción dado un estado.

La RNA estará compuesta por capas cuyas neuronas tendrán función de activación sigmoidea y una capa de salida cuyas neuronas tendrán función de activación softmax. Tanto las capas sigmoideas como la capa softmax son densas en el sentido de que todas las neuronas de la capa anterior se encuentran conectadas con todas las neuronas de la capa siguiente. El motivo de usar esta función de activación para las neuronas de la última capa es para poder interpretar su salida como una probabilidad.

### 5.5.1. Función softmax

La función softmax es una función vectorial  $S(\mathbf{a}) : \mathbb{R}^N \rightarrow \mathbb{R}^N$ . Es tal que cada elemento de la imagen pertenece al intervalo  $[0, 1]$  y la suma de los elementos del vector de 1. Estas características permiten interpretar a la salida de una capa softmax como un vector de probabilidades tal que

$$S_j = P(y = j \mid \mathbf{a})$$

$$S(\mathbf{a}) : \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \rightarrow \begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_N \end{bmatrix} \quad (5.10)$$

en donde cada elemento de la función vectorial se calcula como

$$S_j = \frac{e^{a_j}}{\sum_{k=1}^N e^{a_k}} \quad \forall j \in 1 \dots N. \quad (5.11)$$

### 5.5.2. Capa softmax

La capa softmax se define como una capa densa lineal seguida por la aplicación de la función softmax al vector de salidas de la capa. Podemos encontrar una representación visual en la figura 5.

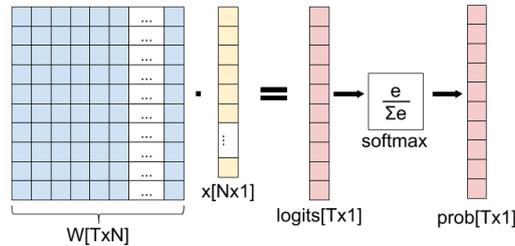


Figura 5: Representación visual de la capa softmax,  $W$  es la matriz de pesos,  $x$  el vector de entrada,  $logits$  el vector de salida y  $prob$  el vector de salida después de que se le aplica la función softmax

### 5.5.3. El gradiente de la función softmax

Parte del trabajo consistió en la implementación de las redes neuronales artificiales, para esto se tuvo que obtener las expresiones del gradiente de la función softmax para poder retro-propagar el gradiente del desempeño  $\nabla J$ . A continuación se explica como se obtiene el gradiente de la capa softmax.

Aclaremos que técnicamente lo que obtendremos en este documento no es estrictamente un gradiente, sino un jacobiano, ya que el gradiente está definido únicamente para funciones escalares y no vectoriales. Con esta aclaración hecha, hablaremos de gradiente de ahora en adelante ya que esta es la manera en la que la literatura de Aprendizaje Automático se refiere más comúnmente al jacobiano.

Se busca entonces calcular la derivada parcial de uno de los elementos de salida de la función respecto a cada uno de los elementos de la entrada de la función, lo que es:

$$\frac{\partial S_i}{\partial a_j} \quad (5.12)$$

que notaremos como  $D_j S_i$  y es:

$$D_j S_i = \frac{\partial S_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^N e^{a_k}}}{\partial a_j}. \quad (5.13)$$

Usando la regla del cociente:

$$f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{[h(x)]^2} \quad (5.14)$$

en nuestro caso, tenemos:

$$g_i = e^{a_i} \quad (5.15)$$

$$h_i = \sum_{k=1}^N e^{a_k}. \quad (5.16)$$

La derivada parcial de  $h_i$  respecto a  $a_j$  es  $e^{a_j}$  independiente del  $j$  para el cual se esté computando. La derivada de  $g_i$  en cambio es  $e^{a_i}$  solamente cuando  $j = i$ .

Ahora se divide el cálculo de la derivada parcial en dos, cuando  $j = i$  y cuando  $j \neq i$ .

Cuando  $j = i$ :

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^N e^{a_k}}}{\partial a_j} = \frac{e^{a_i} \Sigma - e^{a_j} e^{a_i}}{\Sigma^2} \quad (5.17)$$

donde  $\Sigma$  representa  $\sum_{k=1}^N e^{a_k}$ .

Reordenando los términos tenemos:

$$\begin{aligned} \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^N e^{a_k}}}{\partial a_j} &= \frac{e^{a_i} \Sigma - e^{a_j} e^{a_i}}{\Sigma^2} \\ &= \frac{e^{a_i} \Sigma - e^{a_j}}{\Sigma \Sigma} \\ &= S_i (1 - S_j). \end{aligned} \quad (5.18)$$

De este modo podemos re-escribir la derivada parcial en términos de la salida de la función:  $D_j S_i = S_i(1 - S_j)$ .

Cuando  $j \neq i$ :

$$\begin{aligned} \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^N e^{a_k}}}{\partial a_j} &= \frac{0 - e^{a_j} e^{a_i}}{\Sigma^2} \\ &= -\frac{e^{a_j} e^{a_i}}{\Sigma \Sigma} \\ &= -S_j S_i. \end{aligned} \quad (5.19)$$

Juntando ambos casos se obtiene:

$$D_j S_i = \begin{cases} S_i(1 - S_j) & i = j \\ -S_j S_i & i \neq j. \end{cases} \quad (5.20)$$

#### 5.5.4. Derivada respecto a los pesos y sesgos

Para calcular la derivada respecto a los pesos y sesgos de la RNA (que son los parámetros  $\theta$  del aproximador) se debe obtener la derivada de la función de desempeño respecto a la excitación de una neurona, esto es:

$$\frac{\partial G}{\partial a_j} \quad (5.21)$$

Se escribe la ecuación como:

$$\frac{\partial G}{\partial a_j} = \sum_{i=1}^N \frac{\partial G}{\partial S_i} \frac{\partial S_i}{\partial a_j} \quad (5.22)$$

para una capa densa se tiene que:

$$\frac{\partial a_j}{\partial w_{ij}} = x_j \quad (5.23)$$

recordando que  $x$  es un elemento del vector de entrada de la capa densa  $X$ .

Entonces finalmente queda:

$$\frac{\partial G}{\partial w_{ji}} = x_j \sum_{i=1}^N \frac{\partial G}{\partial S_i} \frac{\partial S_i}{\partial a_j} \quad (5.24)$$

y también se tiene que:

$$\frac{\partial a_j}{\partial b_i} = 1 \quad (5.25)$$

resultando:

$$\frac{\partial G}{\partial b_i} = \sum_{i=1}^N \frac{\partial G}{\partial S_i} \frac{\partial S_i}{\partial a_j} \quad (5.26)$$

## 5.6. Optimizadores

Al momento de actualizar la red, no es necesario seguir la regla de actualización de gradiente estocástica (5.8). Se pueden utilizar otras reglas que buscan mejorar la actualización y resolver problemas puntuales que pueden llegar a tener. A continuación se explican todos los optimizadores que fueron utilizados para las redes neuronales implementadas.

Muchos de los optimizadores serán tratados asumiendo gradiente descendiente. Aclaremos que hacer gradiente descendiente con una función de pérdida es análogo a hacer gradiente ascendente con una función de desempeño como se hace usualmente en los métodos de aprendizaje por refuerzo.

### 5.6.1. Momentum

El método de gradiente descendente estocástico (GDS) encuentra problemas cuando se encuentra con quebradas [9], es decir áreas donde la superficie se curva de una manera muy empinada en una dimensión en comparación al resto. Estas áreas son muy comunes alrededor de mínimos locales de la función de costo, o máximos locales si se usa gradiente ascendente y una función de desempeño en vez de costo. En estos casos Momentum tiende a oscilar alrededor de las laderas de la quebrada haciendo muy poco progreso por el fondo de la misma. Podemos entenderlo de manera visual comparando la figura 6 y la figura 7



Figura 6: Gradiente descendente sin Momentum



Figura 7: Gradiente descendente con Momentum

Momentum[3] es un método que ayuda a acelerar el GDS agregando una fracción  $\beta$  de la actualización del paso anterior al paso actual. La idea intuitiva es asociar Momentum a la noción de momento de inercia de una pelota cuando rueda en una quebrada. A medida que va bajando acumula momento en la dirección que baja la quebrada, mas allá de subir y bajar por las laderas:

$$\begin{aligned} m_n &= \beta m_{n-1} + \alpha \widehat{\nabla J(\pi_{\theta_n})} \\ \theta_{n+1} &= \theta_n + m_n. \end{aligned} \tag{5.27}$$

### 5.6.2. RMSProp

A medida que se hacen más complejos los problemas de aprendizaje por refuerzo resulta más común que la distribución de las recompensas sea muy dispersa. En estos casos resulta conveniente modificar la regla de actualización como lo hace RMSProp [4].

Se agrega una estimación del segundo momento no centrado del estimador

del gradiente y modula el factor  $\alpha$  en base a éste:

$$\begin{aligned}
 g_n &= \widehat{\nabla J(\pi_{\theta_n})} \\
 v_n &= \beta v_{n-1} + (1 - \beta)(g_n \odot g_n) \\
 \theta_{n+1} &= \theta_n + \frac{\alpha}{\sqrt{v_n + \epsilon}} g_n
 \end{aligned} \tag{5.28}$$

donde  $\odot$  es el producto elemento a elemento,  $\epsilon$  es un valor muy cercano a 0 (en la implementación se uso  $10^{-8}$  y  $\beta$  es el factor de decaimiento del promedio, experimentalmente se uso 0,9). Este método atenúa los parámetros que se actualizan seguido.

### 5.6.3. ADAM

ADAM, llamado por sus iniciales Adaptative Momentum, o momento adaptativo en español, busca resolver los problemas que resuelve RMSProp y Momentum. Agrega un paso para eliminar el sesgo a cero que tienen  $m_n$  y  $v_n$  cuando  $n$  es chico. El sesgo proviene de que  $m_o = 0$  y  $v_o = 0$  y los valores futuros se promedian con estos valores iniciales:

$$\begin{aligned}
 g_n &= \widehat{\nabla J(\pi_{\theta_n})} \\
 m_n &= \beta_1 m_{n-1} + (1 - \beta_1) g_n \\
 v_n &= \beta_2 v_{n-1} + (1 - \beta_2)(g_n \odot g_n) \\
 \hat{m}_n &= \frac{m_n}{1 - \beta_1^n} \\
 \hat{v}_n &= \frac{v_n}{1 - \beta_2^n} \\
 \theta_{n+1} &= \theta_n + \frac{\alpha}{\sqrt{\hat{v}_n + \epsilon}} \hat{m}_n
 \end{aligned} \tag{5.29}$$

donde  $\hat{v}_n$  y  $\hat{m}_n$  son los estimadores sin sesgo del primer y segundo momento no centrado del estimador del gradiente,  $n > 0$  y  $\beta_1, \beta_2 \in [0, 1)$  son los factores de descuento de cada estimador.

## 6. Métodos de Actor Crítico

Dentro de los métodos de gradientes de políticas existen métodos denominados Actor - Crítico. Estos no solo optimizan la política, sino que además aproximan una función estado valor. Denotamos el vector de parámetros del aproximador para la función estado-valor como  $\mathbf{w} \in R^d$ , donde  $d$  es la dimensión del vector de parámetros. Así, podemos escribir la función estado-valor como  $\hat{v}(s, \mathbf{w})$ .

### 6.1. REINFORCE con valor base

Para entender mejor los métodos Actor-Crítico resulta conveniente entender el algoritmo REINFORCE, particularmente con la modificación de un valor base. Este es el corazón del primer acercamiento a métodos Actor-Crítico. A continuación explicamos como funciona el método REINFORCE para luego incorporar el valor base.

Primero definimos el gradiente de la función de desempeño como  $\nabla J(\boldsymbol{\theta}) = \nabla v_\pi(s_0)$ . Este no es más que el gradiente de la función valor estado. Por el teorema de la gradiente de política tenemos que

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a | s, \boldsymbol{\theta}) \\ &= \mathbb{E}_\pi \left[ \sum_a q_\pi(S_t, a) \nabla \pi(a | S_t, \boldsymbol{\theta}) \right]. \end{aligned} \quad (6.1)$$

Luego, derivando del término derecho de la igualdad que se presenta en la ecuación 6.1 tenemos que

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)} \quad (6.2)$$

en donde  $G_t$  representa el retorno en el tiempo  $t$ ,  $A_t$  representa la acción tomada en el tiempo  $t$  y  $S_t$  representa el estado en el tiempo  $t$ .

El algoritmo completo entonces queda:

Para cada episodio:  
 Generar un episodio  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , siguiendo  $\pi(\cdot | \cdot, \boldsymbol{\theta})$   
 Para cada paso en el episodio  $t = 0, 1, \dots, T - 1$ :

$$\begin{aligned} G &\leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t, \boldsymbol{\theta}) \end{aligned}$$

en donde se introduce el término  $\gamma$  como factor de descuento.

Esta expresión resulta intuitiva ya que nos dice que los parámetros de la política se ajustan en la dirección que maximiza la probabilidad de la acción  $A_t$  en el estado  $S_t$ . El ajuste es proporcional a la recompensa obtenida e

inversamente proporcional a la probabilidad de la acción. Esto causa que el parámetro se mueva más con acciones que proveen un retorno mayor. Mientras que la proporcionalidad inversa con la probabilidad se encarga de que las acciones que se seleccionen más frecuentemente no tengan una ventaja.

### 6.1.1. Introduciendo el valor base

El teorema de gradiente de política puede ser generalizado de tal manera que incluya un valor base. La expresión del teorema así queda:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a (q_\pi(s, a) - b(s)) \nabla \pi(a | s, \boldsymbol{\theta}) \quad (6.3)$$

en donde el valor base  $b(s)$  puede ser cualquier función siempre y cuando no varíe con respecto a la acción. Así tenemos que:

$$\sum_a b(s) \nabla \pi(a | s, \boldsymbol{\theta}) = b(s) \nabla \sum_a \pi(a | s, \boldsymbol{\theta}) = b(s) \nabla 1 = 0. \quad (6.4)$$

Esto nos dice que si la función de valor de base es independiente de la acción esta no va a influir sobre el gradiente ya que es efectivamente 0. Con esto dicho podemos derivar una regla de actualización con valor base de la siguiente manera:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha (G_t - b(S_t)) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)}. \quad (6.5)$$

Esta actualización no afecta el valor esperado de la misma pero si tiene efectos sobre su varianza. Como estamos lidiando con un proceso de decisión de markov (MDP), resulta útil que el valor de base varíe con el estado. Si el valor base para un estado se corresponde con el valor del estado entonces tendríamos un valor base apropiado. Si la función de valor base es similar a  $G_t$ , la diferencia  $G_t - b(S_t)$  será pequeña y por lo tanto los parámetros tita no sufrirán una mayor modificación. Si en cambio, la estimación de la función de valor base es muy diferente a  $G_t$ , la política, que depende de tita, será modificada en mayor manera. Por esta razón resulta natural usar la función estado valor como la función de valor base. Dicho esto, resulta conveniente aprender la función estado valor de manera similar que aprendemos el parámetro de la política  $\theta$ . A continuación se muestra el algoritmo REINFORCE completo con valor base:

Para cada episodio:  
 Generar un episodio  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , siguiendo  $\pi(\cdot | \cdot, \boldsymbol{\theta})$   
 Para cada paso en el episodio  $t = 0, 1, \dots, T - 1$  :

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

$$\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla \hat{v}(S_t, \mathbf{w})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta \gamma^t \nabla \ln \pi(A_t | S_t, \boldsymbol{\theta})$$

Este método tiene dos hiper parámetros,  $\alpha^\theta$  y  $\alpha^w$  en donde estos son el tamaño de paso para el vector de parámetros de la política y el vector de pesos de la función estado valor respectivamente. Ya entendiendo REINFORCE con valor base pasamos a la primera implementación Actor-Crítico.

## 6.2. Actor-Crítico de un paso

Para que la función estado valor sea considerado un Crítico ésta debe ser utilizada para actualizar la estimación del valor de un estado en base a la estimación de los estados siguientes. Esto se denomina bootstrapping. Más adelante veremos por que es beneficioso el bootstrapping que trae el Crítico. Ya introducido el uso de bootstrapping pasamos a ver el método Actor-Crítico de un paso, así mediante un ejemplo podemos entenderlo mejor. El Actor-Crítico de un paso reemplaza el retorno completo de REINFORCE por el retorno de un paso y utiliza la función estado valor aprendida como valor base:

$$\begin{aligned}
 \boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha (G_{t:t+1} - \hat{v}(S_t, \mathbf{w})) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)} \\
 &= \boldsymbol{\theta}_t + \alpha (R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)} \quad (6.6) \\
 &= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)}
 \end{aligned}$$

donde  $\delta_t = R_{t+1} + \gamma v(S_{t+1}, w) - v(S_t, w)$  lo cual remite al calculo de este mismo delta para el método TD(0) y luego modificado valiéndose de métodos Montecarlo[5].

Ya con esta regla de actualización podemos instanciar el algoritmo para el Actor-Crítico de un paso:

```

Para cada episodio:
Inicializar  $S$  (Primer estado del episodio)
 $I \leftarrow 1$ 
Mientras  $S$  no sea terminal (para cada paso temporal):
 $A \sim \pi(\cdot | S, \boldsymbol{\theta})$ 
Ejecutar acción  $A$ , observar  $S', R$ 
 $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$  (si  $S'$  es terminal, sino  $\hat{v}(S', \mathbf{w}) \doteq 0$ )
 $\mathbf{w} \leftarrow \mathbf{w} + \alpha^w \delta \nabla \hat{v}(S, \mathbf{w})$ 
 $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\theta I \delta \nabla \ln \pi(A | S, \boldsymbol{\theta})$ 
 $I \leftarrow \gamma I$ 
 $S \leftarrow S'$ 

```

## 6.3. Proximal Policy Optimization

Proximal Policy Optimization (PPO) es un método de aproximación de política que busca optimizar una función objetivo usando gradiente ascen-

diente estocástica. A diferencia de aproximación de política estándar donde se hace una actualización del gradiente por muestra. Con PPO se tiene una función objetivo tal que permite múltiples épocas de actualizaciones en mini-lotes.

### 6.3.1. Métodos de gradiente de política

En las secciones anteriores describimos los métodos de gradiente de política. Comúnmente estiman el gradiente de la siguiente manera:

$$\hat{g} = \hat{\mathbb{E}}_t \left[ \nabla_{\theta} \ln \pi_{\theta}(a_t | s_t) \hat{A}_t \right] \quad (6.7)$$

donde  $\pi_{\theta}$  es una política estocástica y  $\hat{A}_t$  es un estimador de la ventaja. Se define entonces una función de pérdida idéntica a la anterior:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[ \ln \pi_{\theta}(a_t | s_t) \hat{A}_t \right]. \quad (6.8)$$

Si hacemos múltiples pasos de optimización sobre la función de pérdida utilizando la misma trayectoria se demuestra empíricamente que a menudo se toman actualizaciones destructivamente grandes sobre la política.

### 6.3.2. Métodos de región de confianza

Los métodos de región de confianza buscan resolver los pasos destructivamente grandes que efectúan los métodos de gradiente de políticas clásicos. Estos definen una región de confianza que restringe el espacio de parámetros sobre el cual el paso de actualización se puede mover. Concretamente maximizan respecto a  $\theta$  la función:

$$\hat{\mathbb{E}}_t \left[ \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t \right] \quad (6.9)$$

en donde  $\theta_{\text{old}}$  es el vector de parámetros de la política antes de la actualización. Estas actualizaciones son restringidas por la divergencia KL[1], en donde intuitivamente esta es una medida de cuan diferente es una distribución de probabilidades respecto a otra. Entonces la ecuación 6.9 queda sujeta a:

$$\hat{\mathbb{E}}_t [\text{KL} [\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)]] \leq \delta \quad (6.10)$$

donde la ecuación 6.9 es reemplazada por:

$$\hat{\mathbb{E}}_t \left[ \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t - \beta \text{KL} [\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)] \right] \quad (6.11)$$

donde  $\beta$  es un coeficiente que se toma como hiper parámetro.

### 6.3.3. Objetivo Recortado

Entendiendo el trasfondo que subyace a PPO, pasamos a mostrar como PPO construye su propia función de pérdida que buscara minimizar. Particularmente mostramos la versión de objetivo recortado:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( r_t(\theta) \hat{A}_t, \text{clip} \left( r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right] \quad (6.12)$$

donde

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}. \quad (6.13)$$

El objetivo del segundo término dentro de la expresión de mínimo es remover el incentivo de mover  $r_t$  afuera del intervalo  $[1 - \epsilon, 1 + \epsilon]$ . En cuanto a el estimador de la ventaja se puede utilizar cualquier estimador de ventaja. En particular se usa  $\hat{A}_t = G_t - \hat{v}(s_t)$ , en donde  $G_t$  es la suma de las recompensas descontadas desde el tiempo  $t$  en adelante y  $\hat{v}(s_t)$  es el valor estimado por el crítico para el estado  $s_t$ .

### 6.3.4. Algoritmo completo

Para cada episodio =  $1, 2, \dots, N$   
  Para cada bache =  $1, 2, \dots, N$   
    Para cada elemento del bache  
      Calcular la ventaja  
      Calcular el gradiente del crítico  
      Calcular el gradiente del actor en base a la ventaja  
    Calcular el promedio de gradientes para el crítico  
    Calcular el promedio de gradientes para el actor  
    Actualizar el crítico en base al promedio de sus gradientes con ADAM  
    Actualizar el actor en base al promedio de sus gradientes con ADAM

## 6.4. Funciones de pérdida y gradientes

Para terminar de entender el algoritmo que se presentó en la sección anterior se debe detallar la función de pérdida que se usa para el Actor y el Crítico y sus gradientes que serán propagados por sus respectivas redes.

### 6.4.1. Función de pérdida y gradiente del Actor

La función de pérdida del actor se puede ver en la ecuación 6.12. Lo que se hace entonces es obtener el gradiente de esta función respecto la salida de la red, más propiamente dicho la salida de la capa Softmax, para que después pueda ser propagada por la red.

El gradiente se obtiene como:

Calculamos la ventaja

$$\hat{A}_t = G_t - \hat{v}(s_t)$$

Calculamos la razón de probabilidades como

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Calculamos el objetivo sustituto:

$$\text{surr} = \hat{A}_t * r_t(\theta)$$

Calculamos el objetivo sustituto recortado:

$$\text{surr}_{\text{clip}} = \min \left( r_t(\theta) \hat{A}_t, \text{clip} (r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right)$$

si  $\text{surr} < \text{surr}_{\text{clip}}$

$$\nabla J(\pi(\theta)) = \frac{\hat{A}_t}{r_t(\theta)}$$

sino

si  $(r_t < 1 - \epsilon \text{ o } r_t > 1 + \epsilon)$ :

$$\nabla J(\pi(\theta)) = 0$$

sino

$$\nabla J(\pi(\theta)) = \frac{\hat{A}_t}{r_t(\theta)}$$

en donde  $\nabla J(\pi_\theta)$  es el gradiente de la función de pérdida del Actor respecto a la salida de la red del actor.

## 6.5. Crítico

La función de pérdida en el crítico, en cambio, es considerablemente más simple. Su función de pérdida consiste en el error cuadrático medio entre la salida del crítico y el retorno que obtuvo el agente.

$$J(\hat{v}(s_t)) = (G_t - \hat{v}(s_t))^2 \tag{6.14}$$

y su gradiente.

$$\nabla J(\hat{v}(s_t)) = 2 * (G_t - \hat{v}(s_t)) \tag{6.15}$$

## 7. Resultados Experimentales

En esta sección se exponen los resultados experimentales obtenidos a lo largo del trabajo. El objetivo es obtener conclusiones empíricas acerca de del impacto que tienen los distintos aspectos del aprendizaje por refuerzo que fueron enunciados a lo largo del informe.

Como caso de estudio, para las secciones 7.1 y 7.2, se utiliza el agente y ambiente del péndulo invertido tal como se describió en la sección 4.2. En la práctica, todos los agentes logran aprender una política que consistentemente llega a 200 pasos por episodio, solamente tenemos que seleccionarla durante el entrenamiento. Por está razón las comparaciones en las secciones de péndulo invertido son a fin de entender las diferencias en eficiencia. Finalmente en la sección 7.3 se aplica PPO a un problema complejo para demostrar la flexibilidad de los métodos obtenidos para tratar con problemas de diferente naturaleza y complejidad.

### 7.1. Optimizadores y su impacto en el aprendizaje

Se realizó un experimento que consistió en 3 series, de 30 ejecuciones cada uno, del método REINFORCE. Cada ejecución consistió en 2500 episodios. Un episodio termina si el agente llegó a los 200 pasos o si el péndulo pierde el equilibrio. En la primera serie se usó Momentum, en la segunda RMSProp y en la tercera ADAM. Para cada una de estas series se graficó la cantidad de pasos realizado en cada episodio vs la cantidad de episodios. En cada gráfico se puede observar en rojo el promedio y en azul la desviación estándar.

Se utilizo la implementación de REINFORCE como se ve en la sección 5.3. Para el aproximador de la política se utilizo una RNA con dos capas, una capa sigmoidea seguida por una capa softmax. La capa sigmoidea tiene cuatro neuronas de entrada, una por cada dimensión del estado. La capa oculta tiene diez neuronas y finalmente la capa de salida de la softmax tiene dos neuronas, una para cada acción. Todas las capas están completamente conectadas, es decir, cada neurona de una capa está conectada con todas las neuronas de la capa anterior. Se utilizo un learning rate fijo  $\alpha = 10^{-3}$  y un factor de descuento  $\gamma = 0,9$ . Se hizo uso de la técnica de mini-lotes con un tamaño de lote de 6

En la figura 8 vemos la evolución del agente utilizando Momentum a lo largo de 2500 episodios. Podemos notar una clara tendencia creciente en el desempeño (cuando más pasos realiza durante el episodio mejor), con algún ruido dado que el aprendizaje es estocástico. También se observa una progresión en el desvío estándar.

La figura 9 corresponde a la serie donde se usó RMSProp. Notamos una clara diferencia con Momentum en que no solo el promedio resulta evolucionar más rápidamente y finalmente estabilizarse, sino que además los desvíos estándar son comparativamente menores hacia el final del entrenamiento. Esto es un claro indicador de la superioridad de RMSProp sobre Momentum

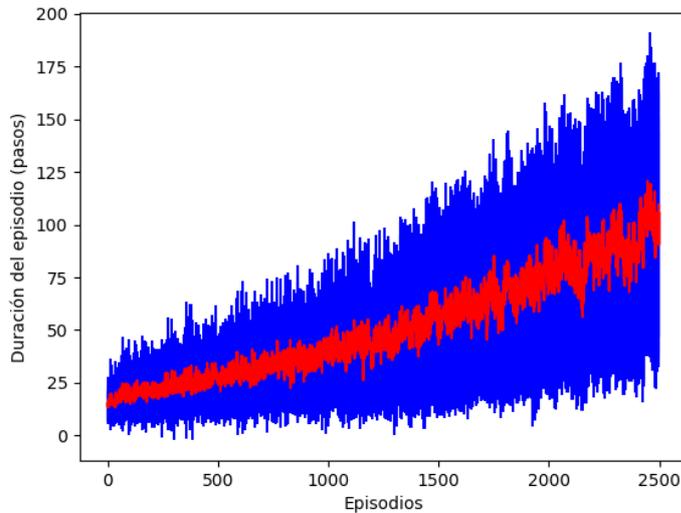


Figura 8: Curva de aprendizaje de un agente REINFORCE utilizando Momentum. la línea roja representa el promedio y el área azul representa el desvío estándar.

para este problema en particular.

En la tercera serie correspondiente a ADAM, se busca la combinación de los beneficios de Momentum y RMSProp. Observando la figura 10 notamos una evolución del desempeño parecida a la que vimos anteriormente en RMSProp. Lo que podemos decir es que ADAM logra estabilizarse antes de los 2000 episodios mientras que RMSProp lo hace posterior a estos. Este resultado confirma que ADAM, integrando Momentum a RMSProp, logra mejores resultados.

las gráficas de la figura 9 y 10 (para RMSProp y ADAM respectivamente) se asemejan más entre sí de lo que lo hacen las gráficas de las figuras 10 y 10 (Momentum y ADAM respectivamente). Esto podría probar experimentalmente para el problema del péndulo invertido que la distribución de las recompensas representan un problema más grande que las quebradas en la superficie de desempeño.

Se comprueba experimentalmente que para este problema de aprendizaje por refuerzo mediante la optimización de políticas los optimizadores logran distintos efectos sobre el aprendizaje. Por lo tanto, ahora en adelante, se utiliza ADAM como optimizador para los sucesivos experimentos, considerándolo el mejor optimizador encontrado.

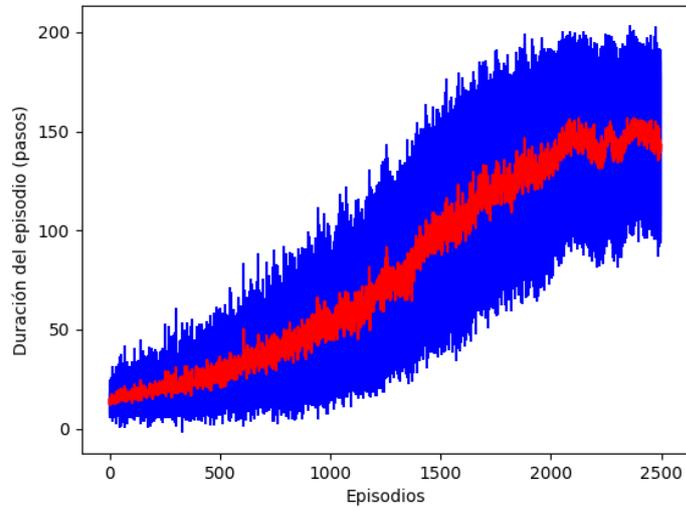


Figura 9: Curva de aprendizaje de un agente REINFORCE utilizando RMS-Prop. la línea roja representa el promedio y el área azul representa el desvío estándar.

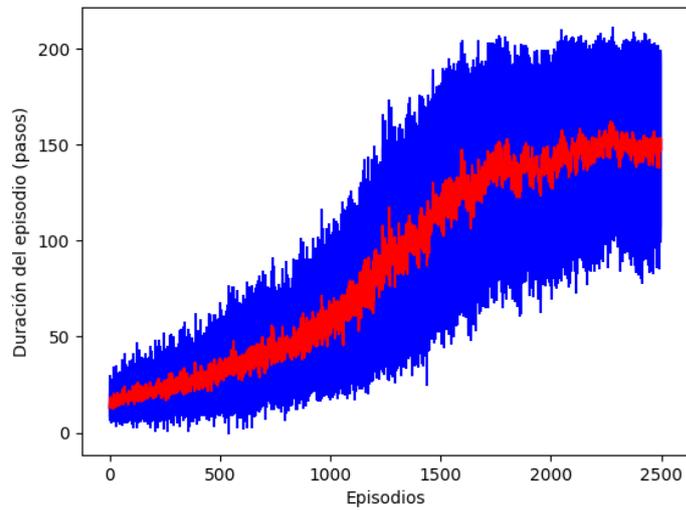


Figura 10: Curva de aprendizaje de un agente REINFORCE utilizando ADAM. la línea roja representa el promedio y el área azul representa el desvío estándar.

## 7.2. REINFORCE vs. PPO

Tanto REINFORCE como PPO constituyen algoritmos de optimización de políticas. En esta sección buscamos compararlos empíricamente mediante experimentos que, tal como en la sección anterior, utilizarán treinta corridas por experimento.

El agente REINFORCE estará implementado de la misma manera que en la sección anterior. El agente PPO tendrá dos RNAs, una para el actor y otra para el crítico.

La RNA del actor consiste en dos capas, una sigmoidea y una softmax. La sigmoidea tiene cuatro neuronas de entrada, hay 20 en la capa oculta y luego cuatro en la salida de la capa softmax. Ambas capas están completamente conectadas. se utiliza un learning rate constante de  $\alpha = 10^{-3}$

El crítico utiliza una RNA también de dos capas. La primera capa es una sigmoidea y la segunda una lineal. la capa sigmoidea tiene 4 neuronas de entrada y 20 de salida, la capa lineal tiene 20 neuronas de entrada y 1 de salida. Ambas capas están completamente conectadas y se utiliza un learning rate constante de  $\alpha = 10^{-3}$ .

Ambas RNAs utilizan la técnica de mini-lotes con un tamaño de lote de 6. El umbral del clip sera de  $\epsilon = 0,1$  y el factor de descuento  $\gamma = 0,99$

### 7.2.1. Curvas de aprendizaje

La figura 11 muestra la curva de aprendizaje de PPO con ADAM. Si comparamos la gráfica de la figura 10 (REINFORCE con ADAM) con la de la figura 11 (PPO con ADAM) podemos observar como PPO a los 500 episodios ya se estabiliza próximo a los 200 pasos por episodio, mientras que REINFORCE tarda aproximadamente 1500 episodios en estabilizarse. Adicionalmente, REINFORCE tiene una desviación estándar mayor y un promedio menor (luego de estabilizarse) comparado con PPO.

Se observa entonces como el algoritmo de tipo actor crítico PPO logra mejores resultados que REINFORCE en términos de velocidad de aprendizaje, varianza entre corridas y promedio de desempeño.

### 7.2.2. Comparación de la entropía

Al ser las acciones seleccionadas, en base a la probabilidad dada por la salida de la red, un punto interesante de análisis para los métodos de gradiente de política resulta ser la entropía del agente. En este caso podemos modelar al agente como una variable aleatoria que llamaremos  $X$ . Esta puede tomar valores de 0 o 1 dependiendo en si el carro debe ser empujado hacia la izquierda o la derecha. Entonces la entropía queda definida como

$$H(X) = - \sum_a p(a) \log_2 p(a). \quad (7.1)$$

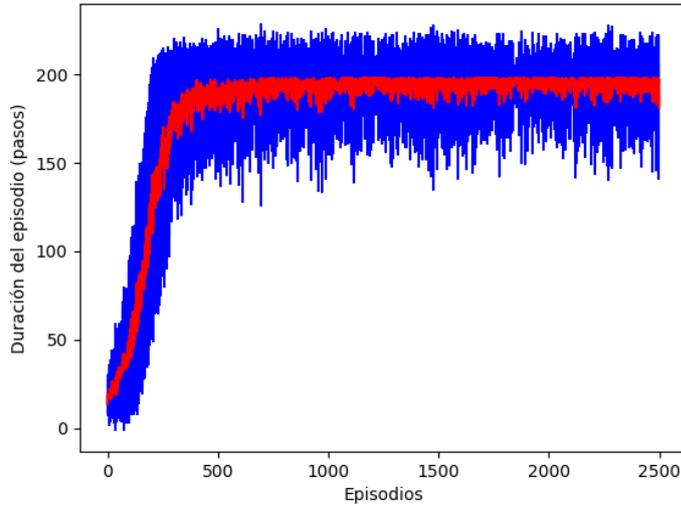


Figura 11: Curva de aprendizaje de un agente REINFORCE utilizando Momentum. la línea roja representa el promedio y el área azul representa el desvío estándar.

En donde  $H(X)$  es la entropía medida en bits de la variable aleatoria previamente definida, y  $p(a)$  es la probabilidad de la acción  $a$ . La sumatoria es sobre todas las acciones. Para entender mejor los posibles valores de  $H(X)$  en base a las probabilidades de las acciones, podemos ver la figura 12.

La entropía nos deja entender a que tipo de política se aproxima un agente, ya que a medida que la entropía de una política se acerca a 0, esta tiende a ser más y más determinista, mientras que cuanto más cerca del máximo de entropía esté, más uniforme es la distribución de probabilidades de acción respecto a un estado.

Si vemos la figura 13 vemos como a lo largo del entrenamiento, el agente REINFORCE reduce la entropía, con la excepción del comienzo en donde al subir, adquiere un carácter más exploratorio. Notamos como la entropía sigue bajando incluso cuando el agente ya se estabilizó (observar la curva de aprendizaje que se mostró anteriormente en la figura 10). Esto quiere decir que incluso estabilizado su desempeño, va hacia una política más determinista aún.

Comparando con la evolución de la entropía de un agente PPO, como se ve en la figura 14, se observa que estos dos algoritmos de optimización de políticas producen políticas distintas. Mientras REINFORCE continua hacia lo que podría ser una política determinista, PPO logra encontrar un óptimo con una política estocástica. Así vemos como, experimentalmente, hay políticas más y menos estocásticas que resuelven el problema, y en este caso, la política con mayor entropía resulta ser la mejor.

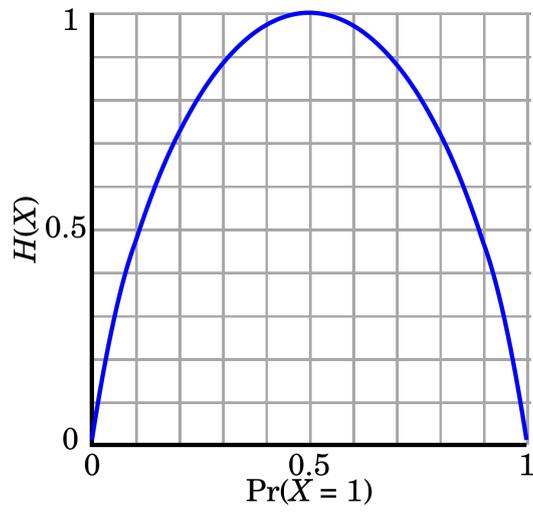


Figura 12: Entropía de una variable aleatoria binaria en base a su probabilidad.

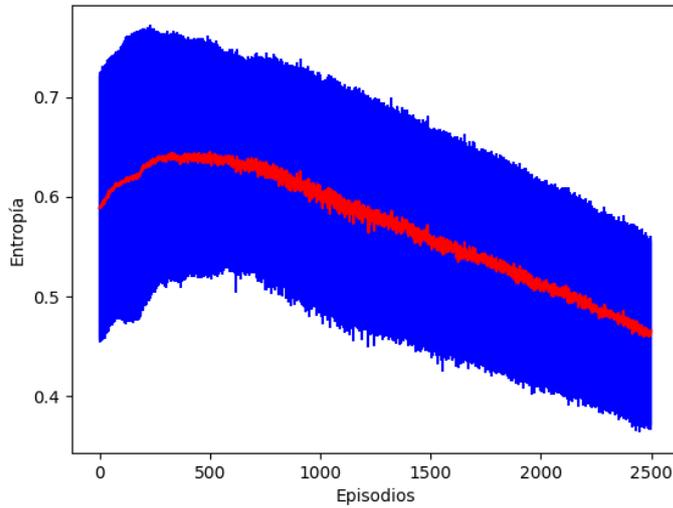


Figura 13: Evolución de la entropía para un agente REINFORCE.

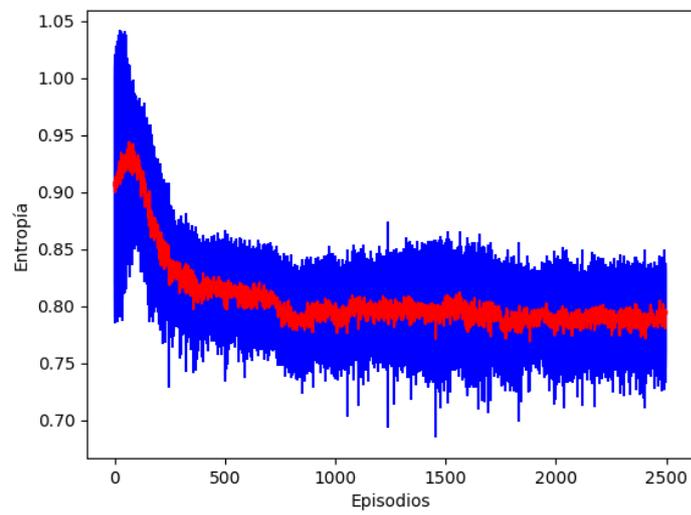


Figura 14: Evolución de la entropía para un agente PPO.

### 7.3. Aplicando PPO a un problema complejo

Hasta ahora se utilizó el problema del péndulo invertido para el análisis de los algoritmos que fueron expuestos en este informe. Si bien el problema simple resulta útil para el análisis comparativo, debido a su simpleza, no explota el total del potencial de los métodos de optimización de políticas.

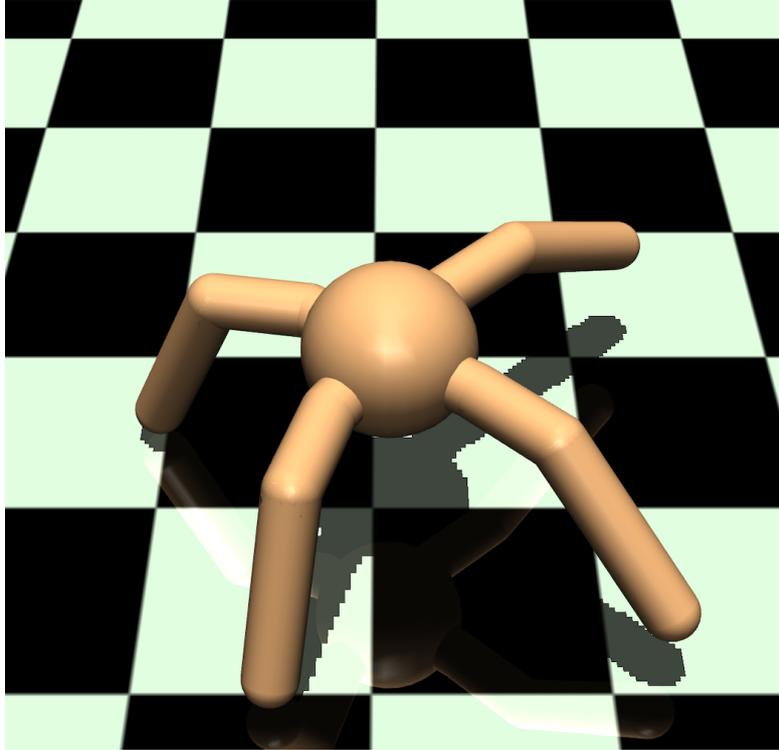


Figura 15: Robot hormiga de cuatro patas.

Para demostrar la flexibilidad de los métodos de optimización de políticas se decidió entrenar a un robot en forma de hormiga de cuatro patas, como se puede ver en la figura 15. Este robot debe aprender a caminar de tal manera que maximice la velocidad con la cual este camina hacia un objetivo fijo dado. A continuación detallamos el agente, su ambiente y la función de refuerzo que se utilizó, para luego analizar los resultados del entrenamiento.

#### 7.3.1. Agente

El agente consiste en ocho rotores que aplican torque sobre las piernas. Esto es, para cada pierna, uno en la junta entre la pierna y el cuerpo y otro en la articulación que tiene el robot en la pierna. La acción consiste entonces en un vector de ocho dimensiones, cada una representando el torque que se aplicará a cada junta. Los torques están acotados en el intervalo  $[-1, 1]$ .

Ahora bien, para poder utilizar la capa softmax necesitamos que las acciones sean discretas, y no un intervalo de los reales tal como se definió anteriormente. Para esto se discretiza el vector de acciones de manera tal

que se tienen cinco posibles acciones para cada dimensión de la acción. De esta manera el torque que se le aplicará a cada junta mediante la ejecución de una acción en un paso temporal será tal que  $t \in \{-1; -0; 5; 0; 0; 5; 1\}$  siendo  $t$  el torque.

De esta manera utilizamos ocho redes con salida softmax independientes en donde cada una de estas será responsable de una dimensión (o cabezal) de acción del agente. Es posible separar el agente por dimensiones de manera tal que no comparten capas de la red gracias a la representación del estado que se describe en la siguiente sección.

La arquitectura de la RNA de cada cabezal es tal como se describe para PPO en 7.2, con la diferencia de que la capa sigmoidea tendrá 28 neuronas de entrada en vez de 4 y la capa softmax tendrá 5 salidas, acorde a la discretización de la acción del cabezal.

### 7.3.2. Representación del estado

El estado del agente consiste en un vector de 28 dimensiones. La primera dimensión es la distancia vertical (eje  $z$ ) que tiene el cuerpo relativa a la posición inicial del agente. Permite entender si el robot está dado vuelta o no ya que una mayor distancia significa que el cuerpo está más cerca del piso. La segunda y tercera dimensión son el seno y el coseno del ángulo al objetivo. La cuarta, quinta y sexta dimensión consisten en las velocidades  $v_x, v_y, v_z$  que tiene el robot. La séptima y octava dimensión son el cabeceo y alabeo del robot. Luego siguen otras 16 dimensiones representando las posiciones y velocidades de cada una de las ocho juntas relativas al cuerpo del robot. Finalmente las cuatro dimensiones restantes son tales que pueden valer 0 o 1 en donde 0 indica que una pata no está tocando el piso y 1 indica que sí. Así se suma cómo parte del estado la noción de contacto con el piso para las cuatro patas.

Esta función de estado contiene la información de velocidades y posiciones de todas las patas y el agente lo cual permite tener redes disjuntas que cada una representando una dimensión de acción ya que, la red de la acción tendrá contexto del estado de las otras juntas mediante la representación del estado.

### 7.3.3. Función de Refuerzo

La función de refuerzo se puede escribir como:

$$\mathcal{R}_t = \text{alive}_t + \text{progress}_t \quad (7.2)$$

en donde  $\text{alive}_t$  vale 1 si el robot no está dado vuelta, 0 si está dado vuelta.

Por otro lado,  $\text{progress}_t$  está definido según:

$$\text{progress}_t = \text{potential}_t - \text{potential}_{t-1} \quad (7.3)$$

y  $potential_t$  como:

$$potential_t = -\frac{targetDistance}{dt} \quad (7.4)$$

donde  $targetDistance$  es la distancia al objetivo y  $dt$  es el paso temporal del ambiente. Esto nos da una medida de la velocidad con la cual el robot está acercando al objetivo. Entonces  $progress$  indica si el agente está mejorando o no la velocidad de acercamiento al objetivo.

#### 7.3.4. Progreso del entrenamiento

Para entender el progreso del entrenamiento del agente se toma como medida el retorno que este obtiene en cada episodio. Sin embargo, a los efectos de visualizar más claramente la curva de aprendizaje, se toma como variable dependiente el promedio de los valores de retorno usando una ventana deslizante de longitud 100. Más específicamente se define la ventana en un tiempo  $t$  tal como:

$$w_t = \frac{1}{100} \sum_t^{t+100} G_t \quad (7.5)$$

y su desvío estándar como:

$$\sigma(w_t) = \sqrt{\frac{1}{100} \sum_t^{t+100} |G_t - w_t|^2}. \quad (7.6)$$

La figura 16 muestra la curva de aprendizaje para el agente. Podemos observar como a medida de que progresan los episodios el retorno por episodio crece. A medida que el agente progresa en el entrenamiento observamos como la curva se va aplanando.

#### 7.3.5. Análisis de la entropía

Para poder diagnosticar el entrenamiento del agente resulta útil el análisis de la entropía. El agente cuenta con ocho cabezales de acción, uno por cada dimensión de acción. Cada cabezal tiene cuatro acciones posibles, por lo cual la entropía de Shannon en bits es  $H(X) \in [2, 0]$  donde  $H(X)$  es la entropía de un cabezal. Viendo variaciones de la entropía podemos entender como progresa el agente. Si la entropía está bajando, este está moviéndose hacia una política con mayor explotación ya que está aumentando la probabilidad de las acciones que considera buenas y disminuyendo las que considera malas. Mientras que si vemos que la entropía sube está moviéndose hacia una política más exploratoria, ya que la distribución de las acciones sobre los estados está tendiendo hacia una uniforme. Podemos ver la entropía durante el entrenamiento del agente en la figura 17.

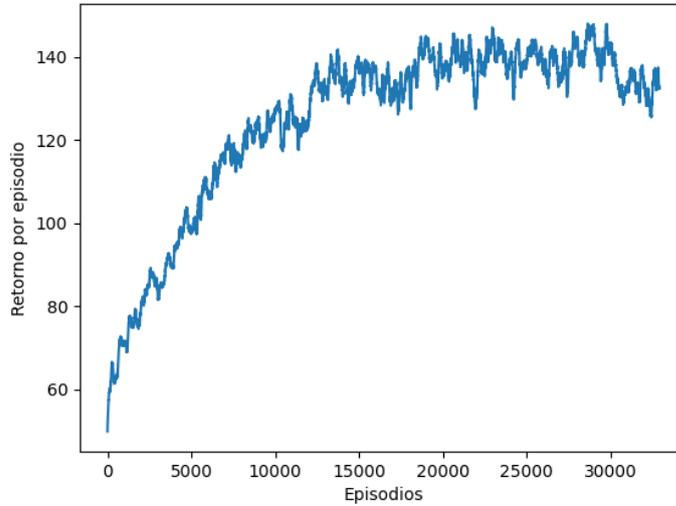


Figura 16: Retorno por episodio vs. Cantidad de episodios durante el aprendizaje de caminata de un robot hormiga de 4 patas usando PPO (los puntos de las gráfica se obtuvieron usando una ventana deslizante de longitud 100).

### 7.3.6. Explotando el agente

Dado que lo que aprende el agente es a caminar hacia un objetivo, una forma de evaluar en qué medida progresa el aprendizaje es midiendo la distancia recorrida del agente obtenido para distintas cantidades de episodios acumulados. Durante el entrenamiento se guardó una versión del agente cada 500 episodios. Luego se hacen 30 corridas para cada versión y se mide el progreso hacia el objetivo en metros que realiza durante 1000 pasos temporales. En la figura 18 se puede observar una gráfica de dicha evolución.

Al observar el comportamiento del agente, podemos notar una mejora progresiva a medida que avanza el entrenamiento. Alrededor de los 3000 episodios de entrenamiento el agente aprende el primer comportamiento observable: utilizar las patas delanteras para ganar distancia aunque no logra progresar mucho ya que no adquirió todavía la habilidad de utilizar las patas traseras. Pasados los primeros 5000 episodios de entrenamiento el agente ya logra caminar, aunque lento y de manera torpe. Al rededor de los 10000 episodios notamos que puede caminar pero no utiliza la extensión completa de las patas traseras, pareciendo impulsarse más con las patas delanteras que las traseras. Finalmente a los 30000 episodios ya logra una caminata más natural utilizando la extensión de sus patas traseras.

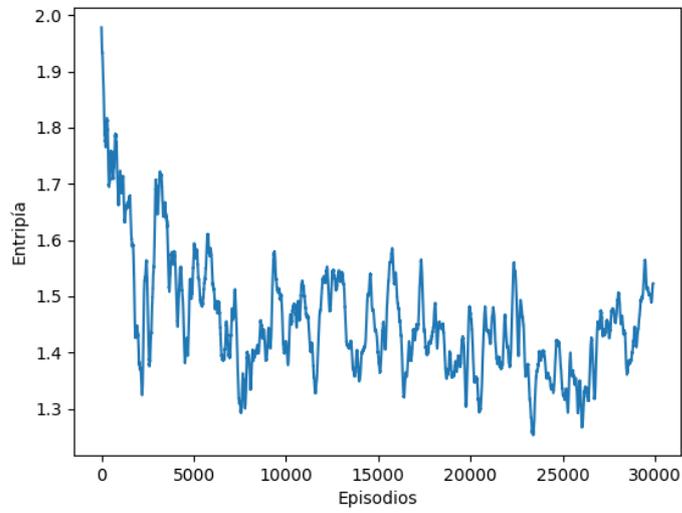


Figura 17: Entropía de un cabezal de acción en base a los episodios de entrenamiento.

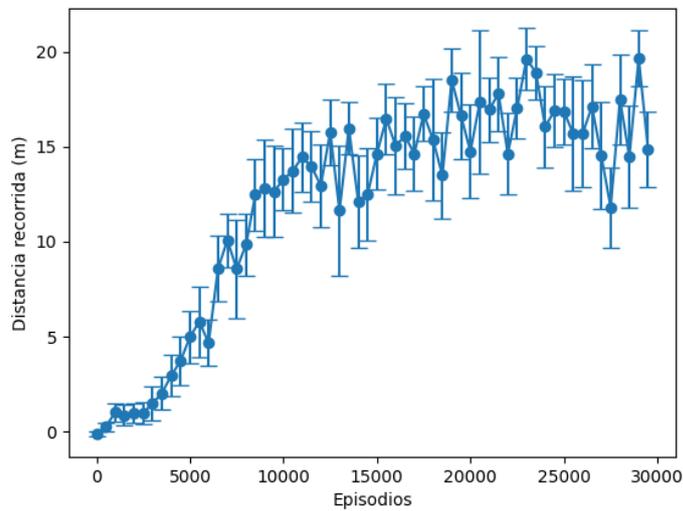


Figura 18: Promedio y desviación estándar de la distancia recorrida de el agente hormiga en función de la cantidad de episodios entrenados.

## 8. Conclusiones

En este informe se logró una síntesis que describe el camino recorrido para llegar a la implementación de un esquema Actor-Crítico basado en PPO, que representa su estado del arte. Se mostraron los fundamentos y el contexto necesario para entender el problema que se quería resolver.

Es importante mencionar que el trabajo fue realizado utilizando solamente librerías de cálculo matricial y simulación física, es decir, se implementaron todas las redes neuronales artificiales y variantes de optimización descritas en el informe. Esto brindó un conocimiento importante ya que al entender el funcionamiento interno de las mismas, se pudo diagnosticar más efectivamente su comportamiento durante el desarrollo. Además, sirvió para entender por qué resultan buenos aproximadores de funciones no lineales y su popularidad tanto en aprendizaje por refuerzo como en otros campos de la inteligencia artificial.

Al momento de elegir el optimizador correcto se experimentó con tres opciones: Momentum, RMSProp y ADAM. Ellos fueron comparados experimentalmente y se concluyó que ADAM era el que permitía obtener el aprendizaje más eficiente para los casos estudiados. Una vez elegido el optimizador a utilizar, se comparó el esquema Actor-Crítico con un método predecesor sin crítico llamado REINFORCE. Observamos como PPO resultó definitivamente superior a la luz de los resultados experimentales. Finalmente, Se logró implementar un esquema Actor-Crítico para que un agente pueda aprender a caminar en un ambiente cuyo espacio de estados tiene componentes continuas y su dimensión es 28. Se pudo observar el carácter progresivo en el cual el agente aprende a caminar y controlar la hormiga: Primero a mover sus patas delanteras, después a caminar erráticamente hasta que finalmente, obtiene un comportamiento de caminata razonablemente armónico

Se llevó a cabo un análisis sobre la entropía que permitió entender el curso del aprendizaje del agente y las políticas obtenidas. Fue una medida muy utilizada en la fase de desarrollo para obtener diagnósticos en línea durante el curso de aprendizaje del agente.

Si bien se logró el objetivo central del trabajo y se pudo verificar experimentalmente al hacer caminar a la hormiga, hay alternativas que quedan por explorar. Se podría, por ejemplo, tener una estructura de red en donde todos los cabezales de acción comparten algunas capas y se diferencian en la salida. Este es un esquema frecuentemente usado en el mundo del aprendizaje por refuerzo con redes neuronales artificiales que permite estructuras más compactas y en donde se pueden representar dependencias entre las componentes del vector de acciones.

Finalmente, el esquema Actor-Crítico implementado puede ser aplicado a una cantidad diversa de problemas de control. Se espera que este trabajo, tanto por este informe como por la implementación realizada en Python, represente una contribución al estudio y uso del esquema de Actor-Crítico para futuras aplicaciones.

## Referencias

- [1] S. Kullback y R. A. Leibler. «On Information and Sufficiency». En: *Ann. Math. Statist.* 22.1 (mar. de 1951), págs. 79-86. DOI: 10.1214/aoms/1177729694. URL: <https://doi.org/10.1214/aoms/1177729694>.
- [2] Richard Bellman. «A Markovian Decision Process». En: *Journal of Mathematics and Mechanics* 6.5 (1957), págs. 679-684. ISSN: 00959057, 19435274. URL: <http://www.jstor.org/stable/24900506>.
- [3] Ning Qian. «On the momentum term in gradient descent learning algorithms». En: *Neural Networks* 12.1 (1999), págs. 145-151. ISSN: 0893-6080. DOI: [https://doi.org/10.1016/S0893-6080\(98\)00116-6](https://doi.org/10.1016/S0893-6080(98)00116-6). URL: <http://www.sciencedirect.com/science/article/pii/S0893608098001166>.
- [4] T. Tieleman y G. Hinton. *Lecture 6.5 - RMSProp*. Technical report. COURSERA: Neural Networks for Machine Learning, 2012.
- [5] Richard S. Sutton y Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: <http://incompleteideas.net/book/the-book-2nd.html>.
- [6] Open AI. *OpenAI Gym*. URL: <https://gym.openai.com/>. (visitado: Diciembre 2020).
- [7] Benjamin Ellenberger y contributors. *PyBullet Gymperium*. URL: <https://github.com/benelot/pybullet-gym>. (visitado: Diciembre 2020).
- [8] PyBullet open source project y contributors. *Pybullet physics simulator*. URL: <https://pybullet.org/>. (visitado: Diciembre 2020).
- [9] Richard S. Sutton. «Two Problems with Backpropagation and Other Steepest-Descent Learning Procedures for Networks». En: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 1986.