INGENIERÍA EN INFORMÁTICA

21 de Diciembre 2020

Esquema Actor-Crítico en aprendizaje por refuerzo con espacios continuos

Proyecto Final

Francisco Delgado
57101

Índice

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 influencian las recompensas inmediatas sino que también influencian 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 esta 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.

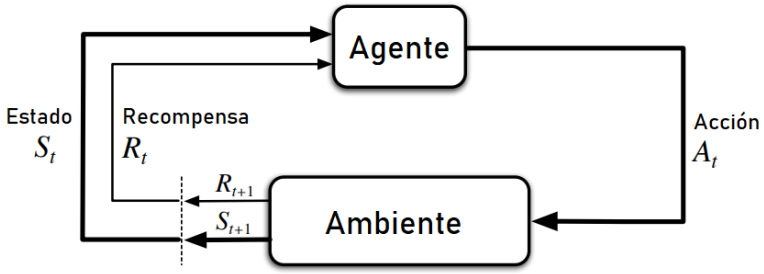

Figura 1: Ciclo de interacción entre el agente y el ambiente. Extraído de [5].

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

$$= S_0; A_0; R_1; S_1; A_1; R_2; S_2; A_2; R_3:::: \quad (3.1)$$

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:

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

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

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 de nida en términos de la recompensa futura que recibirá el agente, lo cual es precisamente el retorno.

Mas formalmente, la función de valor de un estado s bajo una política , de nido como $V(s)$, es el retorno esperado cuando se arranca en s y se sigue la política de ese estado en adelante. Para un PDM se puede de nir formalmente como:

$$V(s) = E[G_t \mid S_t = s], \text{ para todo } s \in S; \quad (3.3)$$

donde $E[:]$ denota el valor esperado de una variable aleatoria dado que el agente sigue la política y t es un paso dado. A esta función se la denomina función de valor-estado bajo .

Similarmente podemos de nir el valor de tomar la acción a en el estado s y luego continuar con la política , denotado como $Q(s; a)$:

$$Q(s; a) = E[G_t \mid S_t = s; A_t = a], \text{ para todo } s \in S; a \in A: \quad (3.4)$$

A esta función se la denomina función de valor estado-acción bajo .

Teniendo estas dos de niciones, podemos de nir $V(s)$ en base a $Q(s; a)$ como:

$$V(s) = {}_{a} E_{(s)}[Q(s; a)]: \quad (3.5)$$

3.4. Ecuaciones de Bellman

Bellman demostro que un problema de optimizacion dinamica en tiempo discreto puede ser descripto de manera recursiva en una forma conocida como induccion invertida[2]. Se hace escribiendo la funcion de valor en un periodo dependiendo solamente del periodo posterior. En el caso de aprendizaje por refuerzo sabemos que el valor de un estado es la suma del refuerzo recibido en s por ejecutar una accion a mas el valor del estado que sigue s^0:

$$V(s) = \underset{\substack{a \\ s^0, r}}{E}{}_{p(s;a)}^{(s)} [r + V(s^0)] \tag{3.6}$$

donde a es la accion que se toma siguiendo la politica, r es la recompensa que se recibe por realizar dicha accion dictada por la funcion de transicion $p(s^0, r \mid s, a)$. es el factor de descuento.

De manera analoga se define para Q

$$Q(s;a) = \underset{s^0, r}{E}{}_{p(s;a)} \left[r + \underset{a^0}{E}{}_{(s^0)} [Q(s^0, a^0)] \right] : \tag{3.7}$$

3.5. Politicas

La politica determina como el agente elige una accion dado que se encuentra en un determinado estado. Estas pueden ser tanto deterministas como estocasticas. Cuando estas son deterministas se encuentran representadas por la letra

$$a = (s); \tag{3.8}$$

y cuando son estocasticas se representan por la letra

$$a = (s;a) \tag{3.9}$$

Esta asocia estados a la probabilidad de tomar cada accion posible. Si el agente sigue la politica en un tiempo t entonces $(s;a)$ 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 de nido por el ambiente. Este espacio puede ser tanto nito como in nito, sin embargo cuando los espacios de estados son in nitos estos no podran ser abordados por los metodos tabulares ya que signi cara una tabla in nita. 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. Funcion de Ventaja

La ventaja es un concepto que permite evaluar el desempeno de una accion relativo al estado en el cual se esta tomando la accion. Esta da una medida de cuan mejor o peor es la accion respecto a las acciones disponibles en ese estado. Esto se cuanti ca en la funcion de ventaja $A : (S; A) ! R$.

$$A (s; a) = Q (s; a) V (s): \quad (3.10)$$

4. Q-Learning tabular

El primer metodo de aprendizaje por refuerzo que se implemento fue Q-Learning. Consiste en aproximar la funcion de estado valor $Q(s;a)$ mediante el uso de una tabla. A continuacion se presenta en detalle como se implemento Q-Learning y un ejemplo puntual del algoritmo.

4.1. Q-Learning

Este es un metodo tabular de aprendizaje por refuerzo que consiste en mantener una tabla Q la cual contiene una estimacion del valor de $Q(s;a)$ para cada posible valor de estado y accion. Al tener que estimar $Q(s;a)$ se lo conoce como un metodo 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 derivacion 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 de nicion de la funcion de valor Q. Esto signi ca que puede ser estimado como el promedio de los retornos de las trayectorias luego de tomar la accion a en el estado s:

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

donde G_i es el retorno de tomar la accion 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.

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

Observando detenidamente la ecuación tiene la estructura básica de:

$$\text{NuevaEst.} = \text{ViejaEst.} + \text{Paso}[\text{Objetivo} - \text{ViejaEst.}]; \tag{4.3}$$

Se interpreta entonces a $[\text{Objetivo} - \text{ViejaEst.}]$ como el error que se cometió en la estimación, este error es aplicado tomando un paso de tamaño α 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 $\frac{1}{n}$. Cuando usamos un paso constante obtenemos el efecto anteriormente mencionado como se ve en el siguiente desarrollo:

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

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

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

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

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: Pendulo 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 posicion del carro, la velocidad del carro, el angulo del palo respecto a la vertical y la velocidad angular del palo.

El problema se considera resuelto cuando el agente mantiene el pendulo balanceado por mas de doscientos pasos. Si el pendulo excede los 15° de inclinacion de la vertical, o el carro se mueve mas de 2.4 unidades del centro se considera terminado el episodio.

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

4.3. Discretizacion del espacio de estados

Debido a que tenemos dimensiones que constituyen un intervalo de $\mathbb{R}$ contamos con un espacio de estados in nito, esto nos imposibilita la implementacion de un agente tabular y por esto debemos discretizar el ambiente. La velocidad angular del palo y el angulo se dividiran en 6 y 12 intervalos de igual tamano respectivamente. El angulo del palo se tendra en el intervalo $[-24; 24]$ grados. La velocidad angular se tendra en el intervalo $[-50; 50]$.

4.4. Resultados

Se utilizo un $\gamma = 1$ y $\alpha = 0{,}1$ como hiper parametros del metodo. Para entender la evolucion del agente a lo largo del aprendizaje, se uso como medida de desempeno el promedio y la desviacion estandar 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 utilizo para la gra ca, una ventana deslizante de 30 episodios. Cuantos mas 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 pendulo invertido, hemos usado la tecnica 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 discretizacion dependera 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 tendra n^d a entradas donde d es la dimension del espacio de estados, n es la cantidad de intervalos que se toman para cada dimension 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 dimension. En dimensionalidades altas y/o con una alta cantidad de intervalos puede resultar en el fenomeno conocido como explosion combinatoria. Por ejemplo si tenemos dos dimensiones con 60 intervalos tendremos $60^2 \quad a = 3600 \quad a$ entradas. Pero si tenemos 8 dimensiones en cambio, tendremos $60^8 \quad a = 1;67 \quad 10^{14}$ entradas.

Por esto resulta necesaria la introduccion de nuevos metodos distintos a los tabulares capaces de abordar estos problemas mas 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 parametros, notados usualmente con

5.1. Metodos de gradientes de poltica

En los metodos del gradiente de poltica, esta ultima es aproximada por una funcion $F(S) = A$ en donde el dominio de F es el espacio de estados y cada punto de la imagen A representa las probabilidades de elegir cada accion dado $s \ 2 \ S$. Esto es, $F(s) = a$ donde $a \ 2 \ [0;1]^{jaj}$, $a = (\ (a_1 js); \ (a_2 js); :::; \ (a_{jA\,j} js)$ y A es el conjunto nito de acciones. Como F esta parametrizada por entonces, podemos escribir.

$$F(s; \ ) = (\quad (a_1 js); \ (a_2 js); :::; \ (a_{jA\,j} js): \tag{5.1}$$

Los metodos de gradientes de politicas deben aprender una poltica parametrizada. Estos tienen como espacio de parametros $2 \ R^{d'}$ donde d' es la dimension del espacio de parametros. La probabilidad que una accion se tome en un tiempo t entonces se da por la ecuacion:

$$(a \, j \, s; \ ) = Pr \, f A_t = a \, j \, S_t = s; \ _t = \quad g \tag{5.2}$$

Para que sea un metodo de gradiente necesitamos de una funcion de desempeno de la poltica, denotada por $J(\ )$. La funcion de desempeno 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 que maximizan el desempeño $J(\theta)$

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J}(\theta_t) \tag{5.3}$$

donde $\widehat{\nabla J}(\theta_t)$ 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

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

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

El propósito será ir modificando los valores de los parámetros θ 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) esta 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:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a Q_\pi(s,a) \nabla \pi(a \mid s, \theta) \tag{5.5}$$

donde $\mu(s)$ es la distribución de probabilidad de los estados bajo la política π_θ 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 del estado s bajo la política π. Esto es equivalente al valor esperado bajo la política y por lo tanto reemplazamos

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

Introducimos en la ecuación (5.6) A_t tal como introdujimos S_t, reemplazando la suma sobre los valores posibles de una variable aleatoria por la expectativa de la misma bajo π. 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 $Q_\pi(S_t,a)\frac{\nabla \pi(a|S_t,\theta)}{\pi(a|S_t,\theta)}$ bajo π. Para lograr esto, multiplicamos y dividimos cada término de la sumatoria por $\pi(a|S_t,\theta)$:

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

donde G_t es el retorno según A_t dado S_t y $A_t \sim \pi(S_t)$. 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:

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

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

$$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$
$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t; \theta)$$

dado que

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

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 $\frac{1}{\sqrt{d}} x$ donde $x \sim N(0, 1)$ 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 ultima capa es para poder interpretar su salida como una probabilidad.

5.5.1. Función softmax

La función softmax es una función vectorial $S(a) : R^N \to R^N$. 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

$S_j = P(y = j \mid a)$

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

en donde cada elemento de la funcion vectorial se calcula como

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

5.5.2. Capa softmax

La capa softmax se de ne como una capa densa lineal seguida por la aplicacion de la funcion softmax al vector de salidas de la capa. Podemos encontrar una representacion visual en la gura 5.

Figura 5: Representacion visual de la capa softmax.W es la matriz de pesos, x el vector de entrada,logits el vector de salida yprob el vector de salida despues de que se le aplica la funcion softmax

5.5.3. El gradiente de la funcion softmax

Parte del trabajo consistio en la implementacion de las redes neuronales arti ciales, para esto se tuvo que obtener las expresiones del gradiente de la funcion softmax para poder retro-propagar el gradiente del desempeno J. A continuacion se explica como se obtiene el gradiente de la capa softmax.

Aclaramos que tecnicamente lo que obtendremos en este documento no es estrictamente un gradiente, sino un jacobiano, ya que el gradiente esta de nido unicamente para funciones escalares y no vectoriales. Con esta aclaracion hecha, hablaremos de gradiente de ahora en adelante ya que esta es la manera en la que la literatura de Aprendizaje Automatico se re ere mas comunmente al jacobiano.

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

$$\frac{\partial S_i}{\partial a_j} \tag{5.12}$$

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 este computando. La derivada de g_i en cambio es e^{a_i} solamente cuando $j = i$.

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

Cuando $j = i$:

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

donde $\sum$ representa $\sum_{k=1}^{N} e^{a_k}$.
Reordenando los terminos tenemos:

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

De este modo podemos re-escribir la derivada parcial en terminos de la salida de la funcion: $D_j S_i = S_i(1 - S_j)$:
Cuando $j \neq i$:

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

Juntando ambos casos se obtiene:

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

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 parametros del aproximador) s se debe obtener la derivada de la funcion de desempeno respecto a la excitacion de una neurona, esto es:

$$\frac{\partial G}{\partial a_j} \tag{5.21}$$

Se escribe la ecuacion como:

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

para una capa densa se tiene que:

$$\frac{\partial a_j}{\partial w_{ij}} = x_j \tag{5.23}$$

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

Entonces nalmente queda:

$$\frac{\partial G}{\partial w_{ij}} = x_j \sum_{i=1}^{N} \frac{\partial G}{\partial S_i} \frac{\partial S_i}{\partial a_j} \tag{5.24}$$

y tambien se tiene que:

$$\frac{\partial a_j}{\partial b_i} = 1 \tag{5.25}$$

resultando:

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

5.6. Optimizadores

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

Muchos de los optimizadores seran tratados asumiendo gradiente descendiente. Aclaramos que hacer gradiente descendiente con una funcion de perdida es analogo a hacer gradiente ascendente con una funcion de desempeno como se hace usualmente en los metodos de aprendizaje por refuerzo.

5.6.1. Momentum

El metodo de gradiente descendente estocastico(GDS) encuentra problemas cuando se encuentra con quebradas [9], es decir areas donde la super cie se curva de una manera muy empinada en una dimensi on en comparaci on al resto. Estas areas son muy comunes alrededor de m nimos locales de la funci on de costo, o m aximos locales si se usa gradiente ascendente y una funci on de desempe no 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 gura 6 y la gura 7

Figura 6: Gradiente descendiente sin Momentum

Figura 7: Gradiente descendiente con Momentum

Momentum[3] es un metodo que ayuda a acelerar el GDS agregando una fracci on de la actualizaci on del paso anterior al paso actual. La idea intuitiva es asociar Momentum a la noci on de momento de inercia de una pelota cuando rueda en una quebrada. A medida que va bajando acumula momento en la direcci on que baja la quebrada, mas alla de subir y bajar por las laderas:

$$\begin{aligned} m_n &= \ m_{n\ 1} + \ r \backslash J(\ _n) \\ {}_{n+1} &= \ _n + m_n : \end{aligned} \tag{5.27}$$

5.6.2. RMSProp

A medida que se hacen mas complejos los problemas de aprendizaje por refuerzo resulta mas comun que la distribuci on de las recompensas sea muy dispersa. En estos casos resulta conveniente modi car la regla de actualizaci on como lo hace RMSProp [4].

Se agrega una estimaci on del segundo momento no centrado del estimador

del gradiente y modula el factor en base a este:

$$
\begin{aligned}
g_n &= r\,J(\ _n) \\
v_n &= \ v_{n\ 1} + (1 \qquad)(g_n \quad g_n) \\
_{n+1} &= \ _n + p\frac{}{v_n + }g_n
\end{aligned}
\tag{5.28}
$$

donde es el producto elemento a elemento, es un valor muy cercano a 0 (en la implementación se uso 10^{8} y es el factor de decaimiento del promedio, experimentalmente se uso 0.9). Este método atenua 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 $m_0 = 0$ y $v_o = 0$ y los valores futuros se promedian con estos valores iniciales:

$$
\begin{aligned}
g_n &= r\,J(\ _n) \\
m_n &= \ _1 m_{n\ 1} + (1 \qquad _1) g_n \\
v_n &= \ _2 v_{n\ 1} + (1 \qquad _2)(g_n \quad g_n) \\
\hat{m}_n &= \frac{m_n}{1 \qquad ^{n}_{1}} \\
\hat{v}_n &= \frac{v_n}{1 \qquad ^{n}_{2}} \\
_{n+1} &= \ _n + p\frac{}{\hat{v}_n + }\hat{m}_n
\end{aligned}
\tag{5.29}
$$

donde $\hat{v}_n$ y $\hat{m}_n$ son los estimadores sin sesgo del primer y segundo momento no centrado del estimador del gradiente, > 0 y $_1;\ _2 \ 2\ [0; 1)$ son los factores de descuento de cada estimador.

6. Metodos de Actor Crtico

Dentro de los metodos de gradientes de polticas existen metodos denominados Actor - Crtico. Estos no solo optimizan la poltica, sino que ademas aproximan una funcion estado valor. Denotamos el vector de parametros del aproximador para la funcion estado-valor como$w \in \mathbb{R}^d$, donded es la dimension del vector de parametros. As, podemos escribir la funcion estado-valor como$\hat{v}(s;w)$.

6.1. REINFORCE con valor base

Para entender mejor los metodos Actor-Crtico resulta conveniente entender el algoritmo REINFORCE, particularmente con la modi cacion de un valor base. Este es el corazon del primer acercamiento a metodos Actor-Crtico. A continuacion explicamos como funciona el metodo REINFORCE para luego incorporar el valor base.

Primero de nimos el gradiente de la funcion de desempe no como$\nabla J(\theta) = \nabla v_\pi(s_0)$. Este no es mas que el gradiente de la funcion valor estado. Por el teorema de la gradiente de poltica tenemos que

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

Luego, derivando del termino derecho de la igualdad que se presenta en la ecuacion 6.1 tenemos que

$$\theta_{t+1} \doteq \theta_t + \alpha G_t \frac{\nabla \pi(A_t \mid S_t,\theta_t)}{\pi(A_t \mid S_t,\theta_t)} \tag{6.2}$$

en dondeG_t representa el retorno en el tiempot, A_t representa la accion tomada en el tiempot y S_t representa el estado en el tiempot.
El algoritmo completo entonces queda:

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

$$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$
$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t \mid S_t,\theta)$$

en donde se introduce el termino γ como factor de descuento.
Esta expresion resulta intuitiva ya que nos dice que los parametros de la poltica se ajustan en la direccion que maximiza la probabilidad de la accion A_t en el estadoS_t. El ajuste es proporcional a la recompensa obtenida e

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

6.1.1. Introduciendo el valor base

El teorema de gradiente de poltica puede ser generalizado de tal manera que incluya un valor base. La expresion del teorema as queda:

$$r\ J(\ ) / \sum_{s} (s) \sum_{a} (q\ (s;a)\quad b(s))\ r\quad (a\ j\ s;\ \ ) \qquad (6.3)$$

en donde el valor base $b(s)$ puede ser cualquier funcion siempre y cuando no vare con respecto a la accion. As tenemos que:

$$\sum_{a} b(s) r\quad (a\ j\ s;\ \ ) = b(s) r \sum_{a} (a\ j\ s;\ \ ) = b(s) r\ \ 1 = 0: \qquad (6.4)$$

Esto nos dice que si la funcion de valor de base es independiente de la accion esta no va a in uir sobre el gradiente ya que es efectivamente 0. Con esto dicho podemos derivar una regla de actualizacion con valor base de la siguiente manera:

$$_{t+1} \doteq\ _{t} +\quad (G_t\quad b(S_t)) \frac{r\quad (A_t\ j\ S_t;\ \ _t)}{(A_t\ j\ S_t;\ \ _t)}: \qquad (6.5)$$

Esta actualizacion no afecta el valor esperado de la misma pero si tiene efectos sobre su varianza. Como estamos lidiando con un proceso de decision de markov (MDP), resulta util que el valor de base vare con el estado. Si el valor base para un estado se corresponde con el valor del estado entonces tendramos un valor base apropiado. Si la funcion de valor base es similar a G_t, la diferencia $G_t\quad b(S_t)$ sera pequena y por lo tanto los parametros tita no sufriran una mayor modi cacion. Si en cambio, la estimacion de la funcion de valor base es muy diferente de G_t, la poltica, que depende de tita, sera modi - cada en mayor manera. Por esta razon resulta natural usar la funcion estado valor como la funcion de valor base. Dicho esto, resulta conveniente aprender la funcion estado valor de manera similar que aprendemos el parametro de la poltica . A continuacion se muestra el algoritmo REINFORCE completo con valor base:

Para cada episodio:
Generar un episodio $S_0; A_0; R_1; : : : ; S_{T\ \ 1}; A_{T\ \ 1}; R_T$; siguiendo $(\ j\ ;\ \ )$
Para cada paso en el episodio $t = 0; 1; : : : ; T\quad 1$:

$$G \quad \sum_{k=t+1}^{T} \ \ ^{k\ \ t\ \ 1} R_k$$
$$G\quad \hat{v}(S_t; w)$$
$$w \quad w + \ ^{w}\ r\ \hat{v}(S_t; w)$$
$$+\quad \ ^{t}\ r\ \ln\ (A_t\ j\ S_t;\ \ )$$

Este metodo tiene dos hiper parametros, y ^{w} en donde estos son el tamano de paso para el vector de parametros de la poltica y el vector de pesos de la funcion estado valor respectivamente. Ya entendiendo REINFORCE con valor base pasamos a la primera implementacion Actor-Crtico.

6.2. Actor-Crtico de un paso

Para que la funcion estado valor sea considerado un Crtico esta debe ser utilizada para actualizar la estimacion del valor de un estado en base a la estimacion de los estados siguientes. Esto se denomina bootstrapping. Mas adelante veremos por que es bene cioso el bootstrapping que trae el Crtico. Ya introducido el uso de bootstrapping pasamos a ver el metodo Actor-Crtico de un paso, as mediante un ejemplo podemos entenderlo mejor. El Actor-Crtico de un paso reemplaza el retorno completo de REINFORCE por el retorno de un paso y utiliza la funcion estado valor aprendida como valor base:

$$\begin{aligned} {}_{t+1} &\doteq {}_{t} + \quad (G_{t:t+1} \quad \hat{v}(S_t;w)) \frac{r \quad (A_t \mid S_t; \ {}_{t})}{(A_t \mid S_t; \ {}_{t})} \\ &= {}_{t} + \quad (R_{t+1} + \quad \hat{v}(S_{t+1};w) \quad \hat{v}(S_t;w)) \frac{r \quad (A_t \mid S_t; \ {}_{t})}{(A_t \mid S_t; \ {}_{t})} \qquad (6.6) \\ &= {}_{t} + \quad {}_{t} \frac{r \quad (A_t \mid S_t; \ {}_{t})}{(A_t \mid S_t; \ {}_{t})} \end{aligned}$$

donde $_t = R_{t+1} + v(S_{t+1};w) \quad v(S_t;w)$ lo cual remite al calculo de este mismo delta para el metodo TD(0) y luego modi cado valiendose de metodos Montecarlo[5].

Ya con esta regla de actualizacion podemos instanciar el algoritmo para el Actor-Crtico de un paso:

Para cada episodio:
Inicializar S (Primer estado del episodio)
I 1
Mientras S no sea terminal (para cada paso temporal):
A (j S;)
Ejecutar accion A; observar S^0; R
R + $\hat{v}(S^0;w)$ $\hat{v}(S;w)$ (si S^0 es terminal, sino $\hat{v}(S^0;w) \doteq 0$)
w w + ^{w} r $\hat{v}(S;w)$
+ I r ln (A j S;)
I I
S S^0

6.3. Proximal Policy Optimization

Proximal Policy Optimization (PPO) es un metodo de aproximacion de poltica que busca optimizar una funcion objetivo usando gradiente ascen-

diente estocastica. A diferencia de aproximacion de poltica estandar donde se hace una actualizacion del gradiente por muestra. Con PPO se tiene una funcion objetivo tal que permite multiples epocas de actualizaciones en mini-lotes.

6.3.1. Metodos de gradiente de poltica

En las secciones anteriores describimos los metodos de gradiente de poltica. Comunmente estiman el gradiente de la siguiente manera:

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

donde π_θ es una poltica estocastica y $\hat{A}_t$ es un estimador de la ventaja. Se de ne entonces una funcion de perdida identica a la anterior:

$$L^{PG}(\theta) = \hat{E}_t \left[\ln \pi_\theta(a_t \mid s_t) \hat{A}_t \right] : \tag{6.8}$$

Si hacemos multiples pasos de optimizacion sobre la funcion de perdida utilizando la misma trayectoria se demuestra empricamente que a menudo se toman actualizaciones destructivamente grandes sobre la poltica.

6.3.2. Metodos de region de con anza

Los metodos de region de con anza buscan resolver los pasos destructivamente grandes que efectuan los metodos de gradiente de polticas clasicos. Estos de nen una region de con anza que restringe el espacio de parametros sobre el cual el paso de actualizacion se puede mover. Concretamente maximizan respecto a θ la funcion:

$$\hat{E}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \hat{A}_t \right] \tag{6.9}$$

en donde θ_{old} es el vector de parametros de la poltica antes de la actualizacion. Estas actualizaciones son restringidas por la divergencia KL[1], en donde intuitivamente esta es una medida de cuan diferente es una distribucion de probabilidades respecto a otra. Entonces la ecuacion 6.9 queda sujeta a:

$$\hat{E}_t \left[KL\left[\pi_{\theta_{old}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right] \right] \tag{6.10}$$

donde la ecuacion 6.9 es reemplazada por:

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

donde β es un coe ciente que se toma como hiper parametro.

6.3.3. Objetivo Recortado

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

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

donde

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}. \tag{6.13}$$

El objetivo del segundo termino dentro de la expresion de mnimo 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 $\hat{A}_t = G_t - v(s_t)$, en donde G_t es la suma de las recompensas descontadas desde el tiempo t en adelante y $v(s_t)$ es el valor estimado por el crtico para el estado s_t.

6.3.4. Algoritmo completo

```
Para cada episodio = 1, 2, ..., N
    Para cada bache = 1, 2, ..., N
        Para cada elemento del bache
            Calcular la ventaja
            Calcular el gradiente del crtico
            Calcular el gradiente del actor en base a la ventaja
        Calcular el promedio de gradientes para el crtico
        Calcular el promedio de gradientes para el actor
        Actualizar el crtico 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 perdida y gradientes

Para terminar de entender el algoritmo que se presento en la seccion anterior se debe detallar la funcion de perdida que se usa para el Actor y el Crtico y sus gradientes que seran propagados por sus respectivas redes.

6.4.1. Funcion de perdida y gradiente del Actor

La funcion de perdida del actor se puede ver en la ecuacion 6.12. Lo que se hace entonces es obtener el gradiente de esta funcion respecto la salida de la red, mas propiamente dicho la salida de la capa Softmax, para que despues pueda ser propagada por la red.

El gradiente se obtiene como:

Calculamos la ventaja
$\hat{A}_t = G_t \quad \hat{v}(s_t)$
Calculamos la razon de probabilidades como
$r_t(\ ) = \frac{(a_t|s_t)}{_{old}(a_t|s_t)}$
Calculamos el objetivo sustituto:
$surr = \hat{A}_t \quad r_t(\ )$
Calculamos el objetivo sustituto recortado:
$surr_{clip} = mn \quad r_t(\ )\hat{A}_t; clip(r_t(\ ); 1 \quad ; 1 + \ )\hat{A}_t$
si $surr < surr_{clip}$
    $r\ J(\ (\ )) = \frac{\hat{A}_t}{r_t(\ )}$
sino
    si ($r_t < 1 \quad$ o $r_t > 1 + \ $):
        $r\ J(\ (\ )) = 0$
    sino
        $r\ J(\ (\ )) = \frac{\hat{A}_t}{r_t(\ )}$

en donder $J(\ )$ es el gradiente de la funcion de perdida del Actor respecto a la salida de la red del actor.

6.5. Crtico

La funcion de perdida en el crtico, en cambio, es considerablemente mas simple. Su funcion de perdida consiste en el error cuadratico medio entre la salida del crtico y el retorno que obtuvo el agente.

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

y su gradiente.

$$r\ J(\hat{v}(s_t)) = 2 \quad (G_t \quad \hat{v}(s_t)) \tag{6.15}$$

7. Resultados Experimentales

En esta seccion se exponen los resultados experimentales obtenidos a lo largo del trabajo. El objetivo es obtener conclusiones empricas 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 pendulo invertido tal como se describio en la seccion 4.2. En la practica, todos los agentes logran aprender una poltica que consistentemente llega a 200 pasos por episodio, solamente tenemos que seleccionarla durante el entrenamiento. Por esta razon las comparaciones en las secciones de pendulo invertido son a n de entender las diferencias en e ciencia. Finalmente en la seccion 7.3 se aplica PPO a un problema complejo para demostrar la exibilidad de los metodos obtenidos para tratar con problemas de diferente naturaleza y complejidad.

7.1. Optimizadores y su impacto en el aprendizaje

Se realizo un experimento que consistio en 3 series, de 30 ejecuciones cada uno, del metodo REINFORCE. Cada ejecucion consistio en 2500 episodios. Un episodio termina si el agente llego a los 200 pasos o si el pendulo pierde el equilibrio. En la primera serie se uso Momentum, en la segunda RMSProp y en la tercera ADAM. Para cada una de estas series se gra co la cantidad de pasos realizado en cada episodio vs la cantidad de episodios. En cada gra co se puede observar en rojo el promedio y en azul la desviacion estandar.

Se utilizo la implementacion de REINFORCE como se ve en la seccion 5.3. Para el aproximador de la poltica 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 dimension del estado. La capa oculta tiene diez neuronas y nalmente la capa de salida de la softmax tiene dos neuronas, una para cada accion. Todas las capas estan completamente conectadas, es decir, cada neurona de una capa esta conectada con todas las neuronas de la capa anterior. Se utilizo un learning rate jo $= 10^{-3}$ y un factor de descuento $= 0;9$. Se hizo uso de la tecnica de mini-lotes con un tamano de lote de 6

En la gura 8 vemos la evolucion del agente utilizando Momentum a lo largo de 2500 episodios. Podemos notar una clara tendencia creciente en el desempeno (cuando mas pasos realiza durante el episodio mejor), con algun ruido dado que el aprendizaje es estocastico. Tambien se observa una progresion en el desvo estandar.

La gura 9 corresponde a la serie donde se uso RMSProp. Notamos una clara diferencia con Momentum en que no solo el promedio resulta evolucionar mas rapidamente y nalmente estabilizarse, sino que ademas los desvos estandar son comparativamente menores hacia el nal 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 lnea roja representa el promedio y el area azul representa el desvo estandar.

para este problema en particular.

En la tercera serie correspondiente a ADAM, se busca la combinacion de los bene cios de Momentum y RMSProp. Observando la gura 10 notamos una evolucion del desempe no 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 con rma que ADAM, integrando Momentum a RMSProp, logra mejores resultados.

las gra cas de la gura 9 y 10 (para RMSProp y ADAM respectivamente) se asemejan mas entre s de lo que lo hacen las gra cas de las guras 10 y 10 (Momentum y ADAM respectivamente). Esto podra probar experimentalmente para el problema del pendulo invertido que la distribucion de las recompensas representan un problema mas grande que las quebradas en la super cie de desempe no.

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

Figura 9: Curva de aprendizaje de un agente REINFORCE utilizando RMS-Prop. la lnea roja representa el promedio y el area azul representa el desvo estandar.

Figura 10: Curva de aprendizaje de un agente REINFORCE utilizando ADAM. la lnea roja representa el promedio y el area azul representa el desvo estandar.

7.2. REINFORCE vs. PPO

Tanto REINFORCE como PPO constituyen algoritmos de optimizacion de polticas. En esta seccion buscamos compararlos empricamente mediante experimentos que, tal como en la seccion anterior, utilizaran treinta corridas por experimento.

El agente REINFORCE estara implementado de la misma manera que en la seccion anterior. El agente PPO tendra dos RNAs, una para el actor y otra para el crtico.

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 estan completamente conectadas. se utiliza un learning rate constante de $= 10^{-3}$

El crtico utiliza una RNA tambien 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 estan completamente conectadas y se utiliza un learning rate constante de $= 10^{-3}$.

Ambas RNAs utilizan la tecnica de mini-lotes con un tamano de lote de 6. El umbral del clip sera de $= 0;1$ y el factor de descuento $= 0;99$

7.2.1. Curvas de aprendizaje

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

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

7.2.2. Comparacion de la entropa

Al ser las acciones seleccionadas, en base a la probabilidad dada por la salida de la red, un punto interesante de analisis para los metodos de gradiente de poltica resulta ser la entropa 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 entropa queda de nida como

$$H(X) = \sum_{a} p(a) \log_2 p(a): \qquad (7.1)$$

Figura 11: Curva de aprendizaje de un agente REINFORCE utilizando Momentum. la linea roja representa el promedio y el area azul representa el desvio estandar.

En donde $H(X)$ es la entropia medida en bits de la variable aleatoria previamente definida, y $p(a)$ es la probabilidad de la accion 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 entropia nos deja entender a que tipo de politica se aproxima un agente, ya que a medida que la entropia de una politica se acerca a 0, esta tiende a ser mas y mas determinista, mientras que cuanto mas cerca del maximo de entropia este, mas uniforme es la distribucion de probabilidades de accion respecto a un estado.

Si vemos la figura 13 vemos como a lo largo del entrenamiento, el agente REINFORCE reduce la entropia, con la excepcion del comienzo en donde al subir, adquiere un caracter mas exploratorio. Notamos como la entropia sigue bajando incluso cuando el agente ya se estabilizo (observar la curva de aprendizaje que se mostro anteriormente en la figura 10). Esto quiere decir que incluso estabilizado su desempeño, va hacia una politica mas determinista aun.

Comparando con la evolucion de la entropia de un agente PPO, como se ve en la figura 14, se observa que estos dos algoritmos de optimizacion de politicas producen politicas distintas. Mientras REINFORCE continua hacia lo que podria ser una politica determinista, PPO logra encontrar un optimo con una politica estocastica. As vemos como, experimentalmente, hay politicas mas y menos estocasticas que resuelven el problema, y en este caso, la politica con mayor entropia resulta ser la mejor.

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

Figura 13: Evolucion de la entropa para un agente REINFORCE.

Figura 14: Evolucbn de la entropa para un agente PPO.

7.3. Aplicando PPO a un problema complejo

Hasta ahora se utilizo el problema del pendulo invertido para el analisis de los algoritmos que fueron expuestos en este informe. Si bien el problema simple resulta util para el analisis comparativo, debido a su simpleza, no explota el total del potencial de los metodos de optimizacion de politicas.

Figura 15: Robot hormiga de cuatro patas.

Para demostrar la flexibilidad de los metodos de optimizacion de politicas se decidio 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 continuacion detallamos el agente, su ambiente y la funcion de refuerzo que se utilizo, 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 articulacion que tiene el robot en la pierna. La accion consiste entonces en un vector de ocho dimensiones, cada una representando el torque que se aplicara a cada junta. Los torques estan 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 definio 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 signi ca 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 s . Así se suma como 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:

$$R_t = alive_t + progress_t \tag{7.2}$$

en donde $alive_t$ vale 1 si el robot no está dado vuelta, 0 si está dado vuelta.

Por otro lado, $progress_t$ está de nido según:

$$progress_t = potential_t - potential_{t-1} \tag{7.3}$$

y $potential_t$ como:

$$potential_t = \frac{targetDistance}{dt} \tag{7.4}$$

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 esta acercando al objetivo. Entonces progress indica si el agente esta 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 mas claramente la curva de aprendizaje, se toma como variable dependiente el promedio de los valores de retorno usando una ventana deslizantes de longitud 100. Mas espec camente se de ne la ventana en un tiempo t tal como:

$$w_t = \frac{1}{100} \sum_{t}^{t-100} G_t \tag{7.5}$$

y su desvo estandar como:

$$(w_t) = \sqrt{\frac{1}{100} \sum_{t}^{t-100} jG_t \quad w_t j^2}: \tag{7.6}$$

La gura 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. Analisis de la entropa

Para poder diagnosticar el entrenamiento del agente resulto util el analisis de la entropa. El agente cuenta con ocho cabezales de accion, uno por cada dimension de accion. Cada cabezal tiene cuatro acciones posibles, por lo cual la entropa de Shannon en bits esta $H(X) \ 2 \ [2; 0]$ donde $H(X)$ es la entropa de un cabezal. Viendo variaciones de la entropa podemos entender como progresa el agente. Si la entropa esta bajando, este esta moviendose hacia una poltica con mayor explotacion ya que esta aumentando la probabilidad de las acciones que considera buenas y disminuyendo las que considera malas. Mientras que si vemos que la entropa sube esta moviendose hacia una poltica mas exploratoria, ya que la distribucion de las acciones sobre los estados esta tendiendo hacia una uniforme. Podemos ver la entropa durante el entrenamiento del agente en la gura 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 que 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: Entropa de un cabezal de accon en base a los episodios de entrenamiento.

Figura 18: Promedio y desviacon estandar de la distancia recorrida de el agente hormiga en funcon de la cantidad de episodios entrenados.

8. Conclusiones

En este informe se logro una sntesis que describe el camino recorrido para llegar a la implementacion de un esquema Actor-Crtico basado en PPO, que representa su estado del arte. Se mostraron los fundamentos y el contexto necesario para entender el problema que se quera resolver.

Es importante mencionar que el trabajo fue realizado utilizando solamente libreras de calculo matricial y simulacion fsica, es decir, se implementaron todas las redes neuronales arti ciales y variantes de optimizacion descriptas en el informe. Esto brindo un conocimiento importante ya que al entender el funcionamiento interno de las mismas, se pudo diagnosticar mas efectivamente su comportamiento durante el desarrollo. Ademas, sirvio para entender por que resultan buenos aproximadores de funciones no lineales y su popularidad tanto en aprendizaje por refuerzo como en otros campos de la inteligencia arti cial.

Al momento de elegir el optimizador correcto se experimento con tres opciones: Momentum, RMSProp y ADAM. Ellos fueron comparados experimentalmente y se concluyo que ADAM era el que permita obtener el aprendizaje mas e ciente para los casos estudiados. Una vez elegido el optimizador a utilizar, se compar o el esquema Actor-Crtico con un metodo predecesor sin crtico llamado REINFORCE. Observamos como PPO resulto de nitivamente superior a la luz de los resultados experimentales. Finalmente, Se logro implementar un esquema Actor-Crtico para que un agente pueda aprender a caminar en un ambiente cuyo espacio de estados tiene componentes continuas y su dimension es 28. Se pudo observar el caracter progresivo en el cual el agente aprende a caminar y controlar la hormiga: Primero a mover sus patas delanteras, despues a caminar erraticamente hasta que nalmente, obtiene un comportamiento de caminata razonablemente armonico

Se llevo a cabo un analisis sobre la entropa que permitio entender el curso del aprendizaje del agente y las polticas obtenidas. Fue una medida muy utilizada en la fase de desarrollo para obtener diagnosticos en lnea durante el curso de aprendizaje del agente.

Si bien se logro el objetivo central del trabajo y se pudo veri car experimentalmente al hacer caminar a la hormiga, hay alternativas que quedan por explorar. Se podra, por ejemplo, tener una estructura de red en donde todos los cabezales de accion 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 arti ciales que permite estructuras mas compactas y en donde se pueden representar dependencias entre las componentes del vector de acciones.

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