INGENIERÍA EN INFORMÁTICA

17 de Agosto 2020

Contribución al relevamiento y estado del arte en Aprendizaje por Refuerzo

Proyecto Final

Lucas Emery
57341

Índice

1. Resumen

Este informe es el resultado del trabajo realizado en el relevamiento, estudio y análisis sobre métodos de aprendizaje por refuerzo y aprendizaje por refuerzo profundo, y fue realizado pensando en constituirse en un recurso para la formación de los interesados en el área. Comenzamos con los fundamentos del campo para tener una base firme y una notación unificada. Luego, introducimos una clasificación para las distintas familias de métodos de aprendizaje por refuerzo haciendo hincapié sobre aquellos que luego nos focalizaremos y ayudan a comprender el resto del trabajo. Una vez sentadas las bases, exponemos algunos de los avances más importantes y otros que consideramos de interés en el campo del aprendizaje por refuerzo profundo. Para ilustrar el grado de desarrollo y alcance del aprendizaje por refuerzo, analizamos dos casos novedosos de aprendizaje por refuerzo profundo que utilizan los métodos a ser expuestos. Finalmente, exponemos algunas conclusiones sobre el tema y el trabajo realizado durante el proyecto, caminos alternativos de investigación que no fueron incluidos en este informe y hacia donde dirigiremos nuestra investigación en el área.

2. Introducción

El Aprendizaje Automático (más conocido como Machine Learning) es un área de la Inteligencia Artificial que estudia algoritmos de computadora capaces de mejorar automáticamente a través de la experiencia. A su vez, el campo del aprendizaje automático se divide en tres categorías dependiendo de la información que se le da al sistema:

- **Aprendizaje Supervisado** Se le dan datos de ejemplo y la salida esperada para esos datos, el sistema debe aprender una regla para predecir, inferir o estimar la salida en base a la entrada.
- **Aprendizaje No Supervisado** Se le dan solo datos de entrada y el objetivo del sistema es encontrar una estructura sobre esos datos.
- **Aprendizaje por Refuerzo** Un agente interactúa con un ambiente dinámico en el que debe lograr cierto objetivo. Mientras visita el espacio del problema, se le otorga una realimentación análoga a una recompensa, la cual trata de maximizar.

En este trabajo nos enfocamos específicamente en aprendizaje por refuerzo y aprendizaje por refuerzo profundo, que es la combinación de este campo con redes neuronales profundas. Las redes neuronales profundas son aproximadores de funciones inspirados en la estructura y funcionamiento del cerebro animal [46]. La incorporación de redes neuronales profundas al aprendizaje por refuerzo permitió el avance del campo sobre problemas previamente considerados imposibles y re-disparó la investigación en el área.

El trabajo está estructurado de la siguiente manera, en la sección 3 explicamos los conceptos básicos del campo y definimos la notación. En la sección 4 exploramos las distintas clases de algoritmos de aprendizaje por refuerzo y sus fundamentos matemáticos, enfocándonos en la información que será necesaria para comprender el resto del contenido. En la sección 5 explicamos algunos de los avances más importantes e interesantes del área, a nuestro criterio, introduciendo la matemática necesaria para su comprensión. En la sección 6 realizamos un análisis detallado de dos casos recientes de éxito en el campo del aprendizaje por refuerzo profundo que demuestran el potencial del método. Finalmente, en la sección 7 exponemos nuestras conclusiones y líneas de trabajo futuro.

3. Fundamentos

Las dos principales partes en Aprendizaje por Refuerzo son el Agente y el Ambiente. El agente es una entidad autónoma, en nuestro caso particular un programa de computadora, que actúa en base a información que observa de su entorno y cuya finalidad es lograr uno o varios objetivos. El ambiente es el mundo en el que el agente interactúa y se desenvuelve. En cada momento de esa interacción, el agente observa el estado del mundo y decide que acción tomar. El estado del ambiente puede ser alterado tanto por una acción del agente como por sí solo.

Cuando el agente lleva a cabo una acción, percibe una recompensa del ambiente como se indica en la Figura 1. La recompensa es simplemente un valor numérico que le indica al agente cuan buena o mala es la acción realizada. La recompensa acumulada se denomina retorno, y es el objetivo del Aprendizaje por Refuerzo que el agente aprenda que acciones tomar para maximizar ese retorno.

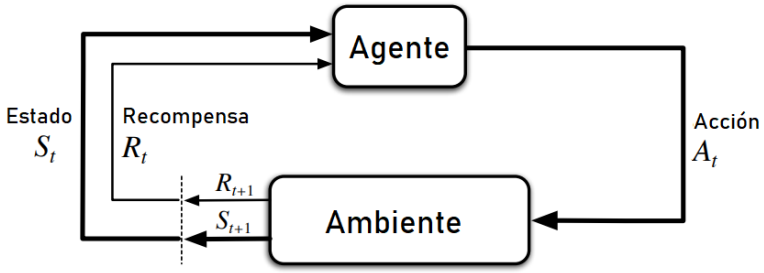

Figura 1: Ciclo de interacción entre el agente y el ambiente. Extraído de [44].

Para esto, el agente debe descubrir qué acciones dan la mayor cantidad de recompensa acumulada mediante prueba y error y, en ciertos casos, las acciones pueden afectar no solo el siguiente estado del ambiente, sino también varios en el futuro. Por lo que el verdadero valor de tomar una acción no se sabrá en ese momento, sino en el futuro próximo o lejano. Estas dos características, el aprendizaje por prueba y error y el desconocimiento del valor real de las acciones, son las que distinguen al Aprendizaje por Refuerzo de otros métodos de Aprendizaje Automático.

3.1. Proceso de Decisión de Markov

Formalmente, el problema que plantea el aprendizaje por refuerzo se modela con un Proceso de Decisión de Markov (PDM). En este modelo, el agente y el ambiente interactúan en pasos temporales discretos $t = 1, 2, 3...$ En cada paso t, el agente recibe el estado del ambiente $S_t \in \mathcal{S}$, en base al cual elige una acción $A_t \in \mathcal{A}(s)$, donde $\mathcal{S}$ es el conjunto de estados posibles y $\mathcal{A}(s)$ es el conjunto de acciones posibles en el estado s. En el siguiente paso temporal, como consecuencia de su acción, el agente recibe un refuerzo $R_{t+1} \in \mathcal{R}$ y

se encuentra en un nuevo estado $S_{t+1} \in \mathcal{S}$, donde $\mathcal{R} \subset \mathsf{R}$ es el conjunto de recompensas posibles y la dinámica de transición del PDM esta dictada por la función $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$

$$p(s^{\emptyset}, r \mid s, a) = Pr\{S_{t+1} = s^{\emptyset}, R_{t+1} = r \mid S_t = s, A_t = a\} \tag{3.1}$$

donde $s, s^{\emptyset} \in \mathcal{S}$, $r \in \mathcal{R}$, $a \in \mathcal{A}(s)$ y $p(s^{\emptyset}, r|s, a)$ es la probabilidad de que el agente alcance el estado $s^{\emptyset}$ y reciba el refuerzo r dado que estaba en el estado s y llevó a cabo la acción a. De esta manera, el agente y el PDM producen lo que se denomina una trayectoria, denotada por la letra τ:

$$\tau = S_0, A_0, R_1, S_1, A_1, R_2, S_2, ... \tag{3.2}$$

3.2. Retorno

Como se indicó anteriormente, el objetivo final del agente es maximizar el retorno, que es la recompensa acumulada en la trayectoria τ. El retorno en el paso t, denotado G_t, se calcula de diferentes maneras dependiendo del problema que se desea resolver. Las dos formas más comunes son el retorno no descontado de horizonte finito:

$$G_t = \sum_{k=t}^{T} {}^{1} R_{k+1} \tag{3.3}$$

donde T es el paso temporal final, y el retorno descontado de horizonte infinito:

$$G_t = \sum_{k=t}^{\infty} \gamma^k R_{k+1} \tag{3.4}$$

donde $\gamma \in (0, 1)$ es un factor de descuento que sirve para garantizar que el retorno no tienda a infinito.

3.3. Política

La política es la regla que utiliza el agente para elegir que acción a tomar en un estado s. Puede ser determinista, en cuyo caso se denota con la letra μ:

$$a = \mu(s), \tag{3.5}$$

o puede ser estocástica, en cuyo caso se denota con la letra π:

$$a \sim \pi(s) \tag{3.6}$$

donde π representa la distribución de probabilidad de acciones para el estado s y la acción a es seleccionada a partir de esa distribución. La probabilidad de tomar la acción a en el estado s bajo la política π se denota $\pi(a|s)$.

3.4. Funciones de Valor

La función de valor de un estado s bajo una política π, denotada $V_\pi(s)$, es el retorno esperado cuando se inicia en s y se sigue π de ahí en adelante. Para un PDM se puede definir formalmente como:

$$V_\pi(s) = \mathsf{E}_\pi[G_t \mid S_t = s], \text{ para todo } s \in \mathcal{S}, \tag{3.7}$$

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

De manera similar, se define la función de valor de tomar una acción a en un estado s bajo una política π, denotada $Q_\pi(s, a)$, como el retorno esperado cuando se inicia en s, se toma la acción a y luego se sigue la política π de ahí en adelante:

$$Q_\pi(s, a) = \mathsf{E}_\pi[G_t \mid S_t = s, A_t = a], \text{ donde } a \in \mathcal{A}(s), \tag{3.8}$$

a esta función se la denomina *funcion de valor estado-accion bajo* π.

Las funciones de valor definen un orden parcial entre políticas. Se dice que una política π es mejor o igual que una política π^0 si el retorno esperado es mayor o igual que el de π^0 para todos los estados. En otras palabras, $\pi \geq \pi^0$ si y sólo si $V_\pi(s) \geq V_{\pi^0}(s)$ para todo $s \in \mathcal{S}$. Existe siempre al menos una política que es mejor o igual que todas las demás [8], esta política se denomina *pol tica optima* y se denota π .

De esta manera, se define la *funcion optima de valor-estado* $V(s)$ como el retorno esperado si se inicia en s y se sigue la política óptima de ahí en adelante:

$$V(s) = \max_\pi \mathsf{E}_\pi[G_t \mid S_t = s] = \max_\pi V_\pi(s), \tag{3.9}$$

y la *funcion optima de valor estado-accion* $Q\ (s, a)$ como el retorno esperado si se inicia en s, se toma la acción a y luego se sigue la política óptima de ahí en adelante:

$$Q\ (s, a) = \max_\pi \mathsf{E}_\pi[G_t \mid S_t = s, A_t = a] = \max_\pi Q_\pi(s, a). \tag{3.10}$$

Podemos observar que existe una dependencia entre estas ecuaciones, ya que el valor óptimo de un estado es aquel que corresponde a tomar la acción óptima en dicho estado, podemos definir $V(s)$ en función de $Q\ (s, a)$:

$$V(s) = \max_{a2A(s)} Q\ (s, a), \tag{3.11}$$

de manera similar podemos definir $V_\pi(s)$ en función de $Q_\pi(s, a)$ si la acción que se toma en s es la correspondiente a la política π para el estado s:

$$V_\pi(s) = \underset{a}{\mathsf{E}}_{\pi(s)}[Q_\pi(s, a)]. \tag{3.12}$$

3.5. Ecuaciones de Bellman

Bellman demostró que un proceso de programación dinámica de tiempo discreto se puede describir de manera recursiva mediante la relación entre la función de valor para un cierto instante t y la función de valor para el instante siguiente $t+1$ [1]. En el caso del aprendizaje por refuerzo, el valor de un estado s es la suma del refuerzo recibido en s, por efectuar una acción a, más el valor del estado siguiente s^0. Formalmente y para el caso del retorno descontado (Ec. 3.4) sería:

$$V_\pi(s) = \mathop{\mathsf{E}}_{\substack{a \\ s^0,r}}^{\pi(s)}{}_{p(s,a)} [r + \gamma V_\pi(s^0)] \tag{3.13}$$

donde a es la acción dictada por la política π en s, s^0 es el estado siguiente, r es el refuerzo por tomar la acción a y están dictados por la función de transición $p(s^0, r \mid s, a)$ (Ec. 3.1). De manera similar, podemos definir las ecuaciones de Bellman para Q_π :

$$Q_\pi(s,a) = \mathop{\mathsf{E}}_{s^0,r}{}_{p(s,a)} \left[r + \gamma \mathop{\mathsf{E}}_{a^0}{}_{\pi(s^0)} [Q_\pi(s^0, a^0)] \right], \tag{3.14}$$

para V :

$$V\,(s) = \max_a \mathop{\mathsf{E}}_{s^0,r}{}_{p(s,a)} [r + \gamma V\,(s^0)], \tag{3.15}$$

y para Q :

$$Q\;(s,a) = \mathop{\mathsf{E}}_{s^0,r}{}_{p(s,a)} \left[r + \gamma \max_{a^0} Q\;(s^0, a^0) \right], \tag{3.16}$$

estas últimas dos ecuaciones se denominan *ecuaciones optimas de Bellman*.

3.6. Función de Ventaja

Muchos de los algoritmos más modernos de aprendizaje por refuerzo utilizan el concepto de ventaja para evaluar el desempeño de una acción, es decir, cuán mejor o peor es esa acción que las demás. La forma de cuantificar esa ventaja es con la función $A_\pi : (\mathcal{S}, \mathcal{A}) \to \mathsf{R}$ e indica cuán mejor es tomar la acción a en un estado s en comparación con seguir la política π:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s). \tag{3.17}$$

4. Familias de Algoritmos en Aprendizaje por Refuerzo

A la hora de clasificar algoritmos de aprendizaje por refuerzo, es difícil de definir una taxonomía clara capaz de clasificar todos los distintos tipos que existen hoy en día. Por eso, vamos a enfocarnos en las diferencias que consideramos más importantes a nivel general.

4.1. Basados en modelo vs. libres de modelo

La primera es si el agente tiene acceso o no a un *modelo del ambiente*. Con modelo nos referimos a una función capaz de predecir transiciones de estado y sus respectivas recompensas, es decir, la función p (3.1). Este modelo puede ser dado o puede ser aprendido por el agente a medida que interactúa con el ambiente.

La principal ventaja de tener un modelo es que el agente puede *planear* a futuro, ver los efectos de las distintas acciones y elegir entre las opciones. Luego, el agente puede destilar una política a partir de esta búsqueda o plan. Los algoritmos basados en modelo son mucho más eficientes con respecto a la cantidad de muestras de experiencia requeridas para su entrenamiento.

Un claro y exitoso ejemplo de esta técnica es AlphaZero [40] que utiliza un modelo dado del ambiente y búsqueda monte-carlo sobre un árbol de jugadas (MCTS de sus siglas en inglés) para obtener un nivel de juego sobrehumano en Go, Shogi y Ajedrez.

Existen también desventajas a este tipo de algoritmos, la principal es que normalmente no disponemos de un modelo preciso del ambiente y el agente debe aprenderlo a través de interacciones con el ambiente real. El problema es que el modelo aprendido puede tener errores, si el agente aprende a explotar esos errores durante el entrenamiento va a tener un rendimiento muy alto cuando interactúe con el modelo pero su rendimiento no va a ser tan bueno (o va a ser malo) cuando tenga que interactuar con el ambiente real.

Los algoritmos libres de modelo sacrifican esta potencial ganancia en eficiencia de muestras a cambio de ser más fáciles de implementar y ajustar. Hoy en día este tipo de algoritmos es más utilizado y, por ende, está más probado. Esto es importante porque el campo del aprendizaje por refuerzo profundo está avanzando tan rápido que no existen pruebas formales de convergencia para la mayoría de los métodos nuevos y la comunidad se basa más en pruebas empíricas y estandarizadas para justificar la efectividad de los mismos.

4.2. Métodos basados en funciones de valor

Los métodos basados en funciones de valor son aquellos en que el principal objetivo es estimar $V(s)$ o $Q(s,a)$. Normalmente se utiliza la función $Q(s,a)$ ya que esta nos permite saber el valor de cada acción a disponible en

un estado s y por ende elegir la mejor. De esta manera, estaríamos eligiendo la acción a siguiendo una política *codiciosa* (aquella que elige la acción con mayor valor esperado) con respecto a $Q*$.

El problema es que en la práctica no tenemos $Q*$. Lo que se hace es partir de una función Q con valores arbitrarios y se define una política π codiciosa con respecto a esta Q. Luego se hace lo que se denomina *iteracion de pol tica*, que consiste en *evaluar la pol tica* π interactuando con el ambiente tomando acciones según esta política y estimando a partir de esta experiencia los valores de la función Q_π. En base a los nuevos y mejorados valores de Q definimos una nueva política π^0 codiciosa con respecto a Q_π, este paso se llama *mejora de pol tica*. De esta manera, obtenemos una política π^0 mejor que π en cada iteración. Esta mejora es fácilmente demostrable, supongamos que la política π elige la acción a para el estado s y la política π^0 elige la acción a^0, es decir, $a = \pi(s)$ y $a^0 = \pi^0(s)$. Si $a \neq a^0$, como:

$$\pi^0(s) = \underset{a^0}{\operatorname{argmax}}\ Q_\pi(s, a^0) \tag{4.1}$$

entonces a^0 es mejor que a en s y por ende π^0 es mejor que π en s, y esto se cumple para todo s donde $\pi(s) \neq \pi^0(s)$. En consecuencia, si seguimos iterando hasta que $\pi^0 = \pi$ habremos obtenido la política óptima π y por ende Q como se indica en la figura 2.

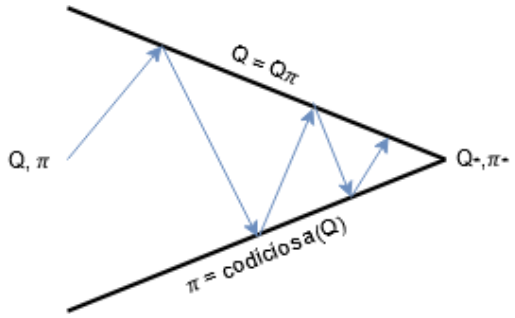

Figura 2: Mejora continua con iteración de política. Extraído de [44].

4.2.1. Q-Learning

Como el valor de la función $Q(s, a)$ es el valor esperado del retorno G (3.8), podríamos estimarlo luego de n trayectorias a partir del promedio de los n retornos observados en cada una de esas trayectorias luego de tomar la acción a en el estado s :

$$Q_{n+1}(s, a) = \frac{1}{n} \sum_{i=1}^{n} G_i(s, a), \tag{4.2}$$

donde $G_i(s, a)$ es el retorno observado luego de tomar la acción a en el estado s en la trayectoria τ_n y la estimación inicial del valor $Q_1(s, a) = 0$.

Con un poco de álgebra podemos ver que el valor de Q_{n+1} se puede determinar simplemente a partir de Q_n y G_n:

$$\begin{aligned} Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} G_i \\ &= \frac{1}{n}\left(G_n + \sum_{i=1}^{n\ \ 1} G_i\right) \\ &= \frac{1}{n}\left(G_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n\ \ 1} G_i\right) \\ &= \frac{1}{n}\left(G_n + (n-1)Q_n\right) \\ &= \frac{1}{n}\left(G_n + nQ_n - Q_n\right) \\ &= Q_n + \frac{1}{n}\left(G_n - Q_n\right), \end{aligned} \tag{4.3}$$

donde Q es $Q(s,a)$ y G es $G(s,a)$. La ecuación (4.3) tiene una estructura característica que se repite en la mayoría de los métodos de aprendizaje por refuerzo, que es la siguiente:

$$\textit{NuevaEst.} = \textit{ViejaEst.} + \textit{Paso}\left[\textit{Objetivo} - \textit{ViejaEst.}\right], \tag{4.4}$$

donde *Est.* significa *Estimacion* y el *Objetivo* es la variable que estamos intentando estimar. La expresión [*Objetivo* − *ViejaEst.*] se puede interpretar como el *error* de la vieja estimación y el *Paso* dicta cuanto va a cambiar nuestra estimación hacia el objetivo.

En la práctica el *Paso* se suele denotar con la letra α y suele no ser igual a $1/n$, esto es porque el objetivo no suele ser *estacionario* y por ende las recompensas de pasos temporales muy lejanos suelen ya no ser representativas del objetivo actual. Por esto se utiliza un paso $\alpha \in (0,1]$ constante, el cual tiene el siguiente efecto:

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha[G_n - Q_n] \\ &= \alpha G_n + (1-\alpha)Q_n \\ &= \alpha G_n + (1-\alpha)[\alpha G_{n\ \ 1} + (1-\alpha)Q_{n\ \ 1}] \\ &= \alpha G_n + (1-\alpha)\alpha G_{n\ \ 1} + (1-\alpha)^2 Q_{n\ \ 1} \\ &= \alpha G_n + (1-\alpha)\alpha G_{n\ \ 1} + (1-\alpha)^2\alpha G_{n\ \ 2} + \\ &\cdots + (1-\alpha)^{n\ \ 1}\alpha G_1 + (1-\alpha)^n Q_1 \\ &= (1-\alpha)^n Q_1 + \alpha\sum_{i=1}^{n}(1-\alpha)^{n\ \ i} G_i, \end{aligned} \tag{4.5}$$

a esto se lo llama *promedio pesado*. Es un promedio porque:

$$(1-\alpha)^n + \alpha\sum_{i=1}^{n}(1-\alpha)^{n\ \ i} = 1$$

y es pesado porque el peso, $\alpha(1-\alpha)^{n\ \ i}$, que se le da a G_i depende de hace cuantos pasos, $n-i$, se observó esa recompensa.

Con estas bases podemos finalmente definir *Q-Learning* [6] (que se traduce como aprendizaje-Q), que es un método de aprendizaje por refuerzo cuyo objetivo es aprender la función Q , para esto utiliza el promedio pesado (4.5) y toma como objetivo la ecuación de Bellman para Q (3.16) y define la regla de actualización:

$$Q_{n+1}(s,a) = Q_n(s,a) + \alpha(r + \gamma \max_{a^{\emptyset}} Q_n(s^{\emptyset}, a^{\emptyset}) - Q_n(s,a)) \tag{4.6}$$

donde $s^{\emptyset}$ es el estado en que terminó el agente luego de tomar la acción a en el estado s.

4.2.2. Exploración vs. Explotación

Para completar el algoritmo de Q-Learning falta una parte, la política. El agente necesita una política para poder interactuar con el ambiente y obtener experiencia, y además, para que funcione la *iteracion de pol tica* y poder mejorar nuestra estimación de Q necesitamos que esta política mejore en el tiempo. Los algoritmos de aprendizaje por refuerzo basados en funciones de valor (como Q-Learning) definen su política en base a la estimación actual de la función de valor. Podríamos simplemente utilizar una política codiciosa con respecto a Q:

$$\pi_n(s) = \operatorname*{argmax}_a \ Q_n(s,a), \tag{4.7}$$

pero al hacer esto aparece un problema, la política π_n elige siempre la acción que cree que es mejor en base al conocimiento que posee actualmente. Esto es un problema porque si la acción $a^{\emptyset}$ es realmente la óptima, como la política codiciosa nunca va a elegir $a^{\emptyset}$ (ya que $Q(s,a) > Q(s,a^{\emptyset})$) su verdadero valor Q nunca va a ser descubierto. Por ende, esta política no va a converger a la política óptima. A este problema se lo conoce como *exploracion vs. explotacion* y hace referencia al balance que debemos hallar entre explorar las acciones de las cuales tenemos poca información y explotar el conocimiento que ya tenemos.

Una solución simple (y muy utilizada) a este problema es definir una política ε-*codiciosa*:

$$\pi_n(s) = \begin{cases} \operatorname{argmax}_a Q_n(s,a) & \textit{con probabilidad } 1-\varepsilon \\ \text{una acción aleatoria} & \textit{con probabilidad } \varepsilon \end{cases} \tag{4.8}$$

con $\varepsilon \in [0,1]$. De esta manera, todas las acciones tienen una probabilidad mayor a cero de ser elegidas. La ventaja de este tipo de políticas es que en el infinito todas las acciones van a ser probadas una infinita cantidad de veces, garantizando la convergencia de Q_π al valor real. Con el parámetro ε podemos regular la cantidad de exploración o explotación, si elegimos un ε más cercano a 0 la política será más codiciosa y si elegimos un ε más cercano a 1 será más aleatoria y en consecuencia más exploratoria.

4.2.3. En-política vs. Fuera-de-política

Los métodos de aprendizaje por refuerzo se categorizan tambien dependiendo de si la *pol tica objetivo* y la *pol tica de comportamiento* son la misma o no, cuando son la misma se dice que el método es *En-pol tica* (On-Policy en inglés) y cuando son distintas se dice que el método es *Fuera-de-pol tica* (Off-policy en inglés), como es el caso de Q-Learning (cuyo objetivo es codicioso con respecto a Q) con una política de comportamiento ε-codiciosa.

4.3. Espacios de estado continuos

Hasta este punto analizamos los métodos asumiendo que el espacio de estados $\mathcal{S}$ era discreto y finito, por lo que la estimación de la función de valor Q podía ser almacenada en una tabla donde cada celda correspondía a un par estado-acción (s, a). Pero en la práctica, en los problemas más interesantes, este espacio suele ser muy grande o simplemente infinito, que es el caso de los espacios continuos donde $\mathcal{S} \subset \mathsf{R}^n$. Por esta razón, se utilizan los denominados *aproximadores de funciones*, como redes neuronales artificiales, que tienen la capacidad de aprender a partir de muestras de una función (por ejemplo Q) y construir una *aproximacion* de la función Q_w donde w representa los parámetros del aproximador. Esto es útil y necesario, ya que es imposible explorar todos los estados posibles en estos espacios tan grandes.

Es a partir de este cruce entre aprendizaje por refuerzo y redes neuronales profundas que nace el término *Aprendizaje por Refuerzo Profundo* (DRL en inglés). Afortunadamente, las redes neuronales profundas son exploradas por otros campos de la inteligencia artificial y el aprendizaje automático (como el aprendizaje supervisado), por lo que los avances en este área se comparten entre estos campos.

4.4. Espacios de acción continuos

De la misma manera que el estado puede ser representado en dimensiones continuas, muchas veces ocurre que las acciones también. Como es el caso en que queremos controlar un brazo robótico mediante el torque de cada uno de los motores de sus juntas, donde el torque es un número real.

Si quisiéramos usar una tabla, tendríamos que definir una acción para cada valor posible de torque para cada motor lo cual es inviable. Si usamos una red neuronal, cuya salida es numérica, podemos cubrir todo el espacio de acciones posibles simplemente definiendo una neurona para cada motor, donde el valor de salida de cada una de estas neuronas correspondería al torque que le queremos aplicar a cada motor. Y esto es sólo un ejemplo, existen muchos métodos diferentes para representar las acciones, como por ejemplo las políticas gaussianas diagonales [19, 50].

4.5. Métodos basados en optimización de la política

A diferencia de los métodos basados en funciones de valor, donde la política se define a partir de las funciones V o Q, en los métodos basados en *optimizacion de la pol tica* la política es representada directamente por un aproximador de función π_θ, donde $\theta \in \mathsf{R}^d$ representa los parámetros del aproximador. Esto significa que no necesitamos conocer los valores de los estados para definir la política.

En los métodos basados en funciones de valor, el problema de aprendizaje por refuerzo es bastante intuitivo ya que el valor que queremos estimar es el retorno esperado para cada estado o para cada par estado-acción, y el retorno se calcula directamente a partir de la recompensa que nos da el ambiente en cada interacción.

Pero en los métodos basados en optimización de la política no es tan simple, la política elige una acción a para un estado s y recibimos un refuerzo r al pasar al estado s^0. Lo que queremos maximizar es el refuerzo acumulado que recibe el agente en toda la trayectoria τ, es decir el retorno G del estado inicial S_0 del episodio. Para esto lo que se hace es definir una función de desempeño $J(\pi_\theta)$ de la política π_θ:

$$J(\pi_\theta) = V_{\pi_\theta}(S_0) \tag{4.9}$$

donde S_0 es el estado inicial de la trayectoria τ bajo π_θ y por ende $V_{\pi_\theta}(S_0)$ es el retorno esperado en τ.

Como la trayectoria τ está dictada por la política π_θ, mejorar el rendimiento J implica mejorar la trayectoria τ. Por esto, los métodos basados en optimización de la política actualizan los parámetros θ del aproximador en el sentido del gradiente del rendimiento J con respecto a θ:

$$\theta_{n+1} = \theta_n + \alpha \nabla J(\pi_{\theta_n}) \tag{4.10}$$

donde $\nabla J(\pi_\theta) \in \mathsf{R}^d$ es un estimador del gradiente del rendimiento J de la política π_θ con respecto a θ.

4.5.1. Teorema del gradiente de la política

El problema es que el rendimiento depende tanto de las acciones que se toman como de los estados en que se toman esas acciones, y ambos dependen de la política π_θ y por ende de los parámetros θ. Si el aproximador de la política π_θ es derivable, el efecto de los parámetros θ sobre la probabilidad de elegir una acción a en un estado s es fácilmente calculable. Pero el efecto de la política sobre la probabilidad de llegar a un estado s depende también de la función de transición (3.1) del ambiente y normalmente no la conocemos.

Afortunadamente, existe una respuesta teórica a este dilema: el *teorema del gradiente de la pol tica* [17], que define una expresión para el gradiente

del rendimiento con respecto a los parámetros de la política que no depende de la derivada de la distribución de probabilidad de los estados bajo π:

$$\nabla J(\pi_\theta) \propto \sum_s \mu_\pi(s) \sum_a Q_\pi(s,a) \nabla \pi_\theta(a \mid s) \tag{4.11}$$

donde $\mu_\pi(s)$ es la distribución de probabilidad de los estados cuando se sigue π y $\propto$ significa "proporcional a".

4.5.2. Algoritmo REINFORCE

Para poder optimizar nuestra política, siguiendo de la ecuación (4.10), necesitamos un estimador del gradiente del rendimiento J. El teorema del gradiente de la política nos da una expresión (4.11) proporcional a este gradiente. Si observamos con detenimiento, (4.11) realiza una suma sobre los estados s pesada según la frecuencia μ_π con que se visita cada estado bajo la política π, por lo que podemos remplazar la suma pesada por la expectativa bajo π E_π:

$$\begin{aligned} \nabla J(\pi_\theta) &\propto \sum_s \mu_\pi(s) \sum_a Q_\pi(s,a) \nabla \pi_\theta(a \mid s) \\ &= \mathsf{E}_\pi \left[\sum_a Q_\pi(S_t,a) \nabla \pi_\theta(a \mid S_t) \right] \text{ donde } S_t \sim \mu_\pi. \end{aligned} \tag{4.12}$$

Para hacer lo mismo con la suma sobre las acciones necesitamos que sea una suma pesada según la frecuencia con que se toma cada acción bajo π, dicha frecuencia es $\pi(s)$. Para no alterar la igualdad, multiplicamos y dividimos por $\pi(s)$:

$$\begin{aligned} \nabla J(\pi_\theta) &\propto \mathsf{E}_\pi \left[\sum_a Q_\pi(S_t,a) \nabla \pi_\theta(a \mid S_t) \right] \\ &= \mathsf{E}_\pi \left[\sum_a \pi_\theta(a \mid S_t) Q_\pi(S_t,a) \frac{\nabla \pi_\theta(a \mid S_t)}{\pi_\theta(a \mid S_t)} \right] \\ &= \mathsf{E}_\pi \left[Q_\pi(S_t,A_t) \frac{\nabla \pi_\theta(A_t \mid S_t)}{\pi_\theta(A_t \mid S_t)} \right] \text{ donde } A_t \sim \pi_\theta(S_t) \\ &= \mathsf{E}_\pi \left[G_t \frac{\nabla \pi_\theta(A_t \mid S_t)}{\pi_\theta(A_t \mid S_t)} \right] \text{ porque } \mathsf{E}_\pi[G_t | S_t, A_t] = Q_\pi(S_t,A_t) \end{aligned} \tag{4.13}$$

donde G_t es el retorno y como $A_t \sim \pi_\theta(S_t)$ entonces $\pi_\theta(A_t|S_t) > 0$. La expresión final (4.13) tiene la forma que necesitamos, es una cifra que puede ser muestreada en cada paso temporal cuyo valor esperado es proporcional al gradiente del rendimiento. De esta manera, y a partir de la ecuación (4.10), obtenemos la ecuación REINFORCE [7] de actualización:

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi_{\theta_t}(A_t \mid S_t)}{\pi_{\theta_t}(A_t \mid S_t)}. \tag{4.14}$$

Para actualizar los parámetros θ a partir de la ecuación REINFORCE necesitamos que el aproximador de la política sea derivable, como es el caso de las redes neuronales artificiales. Supongamos que tenemos una red de 2 capas densas $f_{\theta_1} : \mathcal{S} \to \mathsf{R}^n$ y $g_{\theta_2} : \mathsf{R}^n \to [0,1]^{j\,\mathcal{A}\,j}$ con parámetros θ_1 y θ_2 respectivamente tal que:

$$h = f_{\theta_1}(s) \tag{4.15}$$

$$\pi_\theta(s) = g_{\theta_2}(h) \tag{4.16}$$

donde s es un estado, $h \in \mathsf{R}^n$ es el resultado de la primer capa (llamada capa oculta) y $\pi_\theta(s)$ es un vector de probabilidades de longitud $|\,\mathcal{A}\,|$ donde el $i-ésimo$ elemento es la probabilidad de tomar la acción a_i en el estado s, denotada $\pi_\theta(a_i|s)$.

En base a esto, podemos calcular el efecto de los parámetros $\theta = \{\theta_1, \theta_2\}$ sobre la probabilidad de elegir la acción a_i, es decir el gradiente de $\pi_\theta(a_i|s)$ con respecto a θ:

$$\nabla\pi_\theta(a_i|s) = \left\{ \frac{\partial\pi_\theta}{\partial\theta_1}, \frac{\partial\pi_\theta}{\partial\theta_2} \right\} \tag{4.17}$$

$$\frac{\partial\pi_\theta}{\partial\theta_1} = \frac{\partial g_{\theta_2}}{\partial\theta_1} = \frac{\partial g_{\theta_2}}{\partial f_{\theta_1}}\frac{\partial f_{\theta_1}}{\partial\theta_1} = \frac{\partial g_{\theta_2}}{\partial h}\frac{\partial f_{\theta_1}}{\partial\theta_1}, \text{ por (4.15)} \tag{4.18}$$

$$\frac{\partial\pi_\theta}{\partial\theta_2} = \frac{\partial g_{\theta_2}}{\partial\theta_2} \tag{4.19}$$

y luego actualizar los parámetros de cada capa utilizando la ecuación REINFORCE para θ_1 y θ_2.

4.5.3. Métodos Actor-Crítico

El algoritmo REINFORCE clásico (4.14), al utilizar el retorno como señal de refuerzo que es una variable aleatoria con varianza elevada, sufre de aprendizaje lento [4, 7]. Para subsanar esto, se agrega una *base* $b(s)$ al teorema del gradiente de la política (4.11):

$$\nabla J(\pi_\theta) \propto \sum_s \mu_\pi(s) \sum_a \left(Q_\pi(s,a) - b(s)\right) \nabla\pi_\theta(a \mid s). \tag{4.20}$$

La base puede ser una función cualquiera mientras que no dependa de a. De esta manera, el teorema sigue siendo válido porque la cantidad substraída es cero:

$$\sum_a b(s)\nabla\pi_\theta(a|s) = b(s)\nabla\sum_a \pi_\theta(a|s) = b(s)\nabla 1 = 0. \tag{4.21}$$

A partir del teorema del gradiente de la política con base (4.20) podemos derivar una ecuación REINFORCE de actualización con base:

$$\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t))\frac{\nabla\pi_{\theta_t}(A_t \mid S_t)}{\pi_{\theta_t}(A_t \mid S_t)}. \tag{4.22}$$

Una función base que surge de manera intuitiva al observar que b puede depender de s es la función de valor-estado V, para esto podemos definir un nuevo aproximador de función $V_w(s)$ con parámetros w. De esta manera surgen los denominados *metodos actor-cr tico* [16] en los cuales el *actor* es la política π_θ y el *cr tico* es la función base V_w, este tipo de métodos son un híbrido entre los basados en funciones de valor y los basados en optimización de política y son el estado-del-arte al día de hoy en múltiples aplicaciones de aprendizaje por refuerzo [43, 48].

Normalmente se utiliza una red neuronal como aproximador de V, podemos utilizar una red de 2 capas densas similar a la de la política pero con la diferencia que en lugar de tener tantas salidas como acciones posibles, tiene una sola salida que indica el valor del estado s:

$$h = f_{w_1}(s) \tag{4.23}$$

$$V_w(s) = g_{w_2}(h), \tag{4.24}$$

donde $f_{w_1} : \mathcal{S} \to \mathsf{R}^n$ y $g_{w_2} : \mathsf{R}^n \to \mathsf{R}$. Como el valor del estado s es el retorno esperado en s (3.7), para entrenar la red hay que minimizar el error entre el valor estimado por la red y el retorno obtenido desde el estado s. A este error se los llama *perdida* y se denota con la letra $\mathcal{L}$:

$$\mathcal{L}(V_w) = (G_t - V_w(s))^2 \tag{4.25}$$

donde G_t es el retorno observado en el estado s y en este caso particular $\mathcal{L}$ es el error cuadrático.

Para entrenar la red, hay que actualizar los parámetros w en el sentido contrario al gradiente de $\mathcal{L}$ (porque queremos minimizar la pérdida, a diferencia del rendimiento J de la política que queríamos maximizarlo):

$$w_{t+1} = w_t - \alpha \nabla \mathcal{L}(V_w). \tag{4.26}$$

A diferencia del rendimiento, el gradiente de $\mathcal{L}$ con respecto a w es fácilmente calculable:

$$\begin{aligned} \nabla \mathcal{L}(V_w) &= \nabla \left[(G_t - V_w(s))^2 \right] \\ &= 2(G_t - V_w(s)) \nabla V_w(s) \\ &\propto (G_t - V_w(s)) \nabla V_w(s), \end{aligned} \tag{4.27}$$

por lo que la regla de actualización para w sería entonces:

$$w_{t+1} = w_t - \alpha (G_t - V_w(s)) \nabla V_w(s) \tag{4.28}$$

y ∇V_w se calcula de igual manera que $\nabla \pi_\theta$ ya que su estructura es muy similar:

$$\nabla V_w(s) = \left\{ \frac{\partial V_w}{\partial w_1}, \frac{\partial V_w}{\partial w_2} \right\} \tag{4.29}$$

$$\frac{\partial V_w}{\partial w_1} = \frac{\partial g_{w_2}}{\partial w_1} = \frac{\partial g_{w_2}}{\partial f_{w_1}} \frac{\partial f_{w_1}}{\partial w_1} = \frac{\partial g_{w_2}}{\partial h} \frac{\partial f_{w_1}}{\partial w_1}, \text{ por (4.23)} \tag{4.30}$$

$$\frac{\partial V_w}{\partial w_2} = \frac{\partial g_{w_2}}{\partial w_2}. \tag{4.31}$$

Finalmente, se actualizan los parámetros w_1 y w_2 con la ecuación (4.28) y según sus respectivas derivadas parciales.

5. Avances en Aprendizaje por Refuerzo Profundo

En esta sección exploramos algunos de los avances en el campo del aprendizaje por refuerzo profundo. Comenzamos con avances específicos en entrenamiento de redes neuronales profundas que fueron fundacionales para el éxito de los algoritmos de aprendizaje por refuerzo profundo. Luego, exploramos los avances que nos parecieron más interesantes en algunas de las ramas de este tipo de algoritmos.

5.1. Optimizadores

El entrenamiento de redes neuronales mediante actualización de los parámetros en el sentido del gradiente de la función objetivo sufren de aprendizaje lento por varias razones [51]. Para subsanar estos problemas se introdujeron los denominados *optimizadores* que son alteraciones de la expresión (4.14) que conservan el efecto deseado pero introduciendo beneficios que reducen las perturbaciones en las actualizaciones y mejoran la velocidad de aprendizaje.

En este informe solo analizaremos los optimizadores más utilizados y relevantes al resto del contenido, para un análisis más extenso ver [32].

5.1.1. Momentum

El primer problema que se intentó resolver es que el algoritmo simple de actualización por el gradiente tiene problemas cuando se encuentra en *quebradas*, en otras palabras zonas donde la pendiente del gradiente es mucho más fuerte en una dimensión que en las demás, las cuales son bastante comunes cerca de mínimos locales. Momentum [2, 14] es un método que intenta subsanar esto agregando a la actualización una fracción β de la actualización del paso anterior:

$$\begin{aligned} m_n &= \beta m_{n-1} + \alpha \nabla J(\pi_{\theta_n}) \\ \theta_{n+1} &= \theta_n + m_n. \end{aligned} \tag{5.1}$$

5.1.2. RMSprop

El otro problema es que en los casos en que hay características o rasgos de los datos que son muy comunes y otros que son poco comunes, como es el caso del aprendizaje por refuerzo donde las recompensas pueden estar muy dispersas, resulta necesario poder variar el *paso* α en función de estas frecuencias de aparición. Para esto, RMSprop [21] computa una estimación v_n del segundo momento no centrado del estimador del gradiente y modula

el factor α en base a este estimador:

$$\begin{aligned} g_n &= \nabla J(\pi_{\theta_n}) \\ v_n &= \beta v_{n-1} + (1-\beta)(g_n \odot g_n) \\ \theta_{n+1} &= \theta_n + \frac{\alpha}{\sqrt{v_n + \epsilon}} g_n \end{aligned} \tag{5.2}$$

donde $\odot$ es el producto elemento a elemento, β es el factor de decaimiento del promedio y ϵ es un valor muy cercano a cero (por ejemplo 10^{-8}) para evitar la división por cero. Esto resulta en una atenuación de las actualizaciones de los parámetros que se actualizan seguido (porque v_n es un promedio con decaimiento exponencial) y en una gran amplificación de las actualizaciones de los parámetros que no.

5.1.3. ADAM

ADAM [24] combina los beneficios de Momentum y de RMSprop, y agrega un paso extra para eliminar el sesgo a cero que tienen m_n y v_n cuando n es pequeño. Este sesgo viene de que $m_0 = 0$, $v_0 = 0$ y los valores futuros se promedian con estos valores iniciales:

$$\begin{aligned} g_n &= \nabla J(\pi_{\theta_n}) \\ m_n &= \beta_1 m_{n-1} + (1-\beta_1) g_n \\ v_n &= \beta_2 v_{n-1} + (1-\beta_2)(g_n \odot g_n) \\ \hat{m_n} &= \frac{m_n}{1-\beta_1^n} \\ \hat{v_n} &= \frac{v_n}{1-\beta_2^n} \\ \theta_{n+1} &= \theta_n + \frac{\alpha}{\sqrt{\hat{v_n} + \epsilon}} \hat{m_n} \end{aligned} \tag{5.3}$$

donde $\hat{m_n}$ y $\hat{v_n}$ 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 por promedio con decaimiento exponencial.

ADAM y RMSprop son los optimizadores más utilizados al día de hoy en los algoritmos del estado del arte en aprendizaje por refuerzo profundo [23, 40, 43, 48].

5.2. Avances en métodos basados en funciones de valor

Luego del éxito de TD-Gammon [9] y en base a que no se pudieron obtener resultados igual de exitosos en otros dominios similares [11], el campo del aprendizaje por refuerzo abandonó las redes neuronales en favor de aproximadores con mejores garantías de convergencia [12].

Las redes neuronales volvieron al foco del aprendizaje por refuerzo luego de que en 2013 Mnih et al. [23] mejoraron el estado del arte con la Deep

Q-Network. Lo novedoso de la DQN es que podía jugar juegos de Atari 2600 [22] a partir de una representación del estado obtenida directamente de los píxeles de la pantalla. Para esto incorporaron 3 partes clave: una red neuronal convolucional, repetición de experiencia y objetivo fijo. El funcionamiento de la red convolucional [3, 13] es relevante al campo del aprendizaje profundo, se encuentra detallado en el paper de DQN pero no vamos a analizarlo en este trabajo.

5.2.1. Repetición de experiencia

La repetición de experiencia [5] consiste en mantener una memoria de las experiencias del agente, es decir las transiciones $(S_t, A_t, R_{t+1}, S_{t+1})$, y realizar el entrenamiento sobre muestras uniformemente aleatorias de esta memoria.

Esto trae varias ventajas en comparación al algoritmo clásico de Q-Learning. En primer lugar, al utilizar cada transición (potencialmente) más de una vez la cantidad de información que se extrae de cada muestra es mucho más alta. En segundo lugar, entrenar sobre muestras temporalmente consecutivas es ineficiente debido a las correlaciones que existen entre ellas [44]; utilizar muestras aleatorias rompe estas correlaciones y hace al entrenamiento más estable. En tercer lugar, durante el entrenamiento la política actual determina la distribución de los estados. Conservar transiciones de políticas pasadas y repetirlas durante el aprendizaje permite suavizar este último y evitar oscilaciones o divergencias en los parámetros de la red.

5.2.2. Objetivo fijo

Fijar del objetivo de aprendizaje consiste en no alterar los parámetros del aproximador de la función de valor que se utiliza para calcular el objetivo en la ecuación (4.4). Para esto, lo que hicieron Mnih et al. fue conservar los parámetros θ de la red Q_θ de la iteración anterior para calcular el objetivo:

$$\begin{aligned} \mathcal{L}_n(\theta_n) &= \mathsf{E}_\pi \left[(y_n - Q_{\theta_n}(s,a))^2 \right] \\ y_n &= \mathop{\mathsf{E}}_{\substack{s^0,r \\ p(s,a)}} [r + \gamma \max_{a^0} Q_{\theta_n \ 1}(s^0, a^0)] \end{aligned} \tag{5.4}$$

donde $\mathcal{L}_n$ se denomina *perdida* y es la medida del error de la red que se intenta minimizar, y y_n es el *objetivo jo* que depende de los parámetros de la iteración anterior $\theta_{n \ 1}$. Fijar el objetivo durante el aprendizaje provee estabilidad al entrenamiento, pero la justificación para utilizarlo fue simplemente que la técnica funcionaba en otras áreas como el aprendizaje supervisado.

5.2.3. Doble Q-Learning

Hado van Hasselt estudió en 2010 el fenómeno de la sobre-estimación que se presenta en Q-Learning [20], que es un sesgo a sobre-estimar los valores de los pares estado-acción provocado por la utilización del operador *max*. Para subsanarlo, Hasselt propuso el método *Double Q-Learning* el cual utiliza

dos estimadores de la función de valor Q^A y Q^B y los actualiza de manera alternada utilizando el otro como objetivo.

En 2015 Hasselt aplicó su método a la DQN de Mnih [26] y obtuvo mejores resultados en varios de los juego de Atari 2600, subiendo la vara del estado del arte. Hoy en día la generalización de este método se conoce como *red objetivo* (target network en inglés) y varía dependiendo de la implementación.

5.3. Avances en métodos basados en optimización de la política

Kakade identificó dos problemas clave con los algoritmos clásicos de optimización de política. En primer lugar, el gradiente del rendimiento $\nabla J(\pi_\theta)$ en función de los parámetros θ es preciso solamente en las cercanías de donde se calculó, es decir, dada la política π_{θ^0}, $\nabla J(\pi_{\theta^0}) \approx \nabla J(\pi_\theta)$ se cumple sólo si π_{θ^0} es muy similar a π_θ (i.e. la divergencia KL entre π_θ y π_{θ^0} es pequeña). Es por esta razón que las actualizaciones sobre θ no pueden ser muy grandes y se utiliza un factor α pequeño en la actualización. El problema que tienen los algoritmos clásicos es que restringen la magnitud de las actualizaciones en el espacio de parámetros θ de manera uniforme (i.e. utilizan el mismo α para todos los parámetros) pero el rendimiento J depende de la distribución π_θ y un paso de actualización pequeño en θ no necesariamente es pequeño en π_θ. Para subsanar esto, Kakade propuso un método denominado *optimizacion de pol tica por el gradiente natural* que limita la divergencia KL entre π_θ y π_{θ^0} en lugar de la distancia entre θ y θ^0. El gradiente natural $\widetilde{\nabla} J(\pi_\theta)$ es el gradiente $\nabla J(\pi_\theta)$ pero corregido en base a la curvatura local en el espacio de la política:

$$\widetilde{\nabla} J(\pi_\theta) = F^{\ 1} \nabla J(\pi_\theta) \tag{5.5}$$

donde F es la matriz de información de Fisher de π_θ. La demostración de (5.5) está disponible en [49]. De esta manera, la ecuación de actualización del método del gradiente natural es:

$$\theta_{n+1} = \theta_n + \alpha \widetilde{\nabla} J(\pi_\theta). \tag{5.6}$$

En segundo lugar, Kakade y Langford demostraron que el problema fundamental con los métodos clásicos de optimización de política por el gradiente del rendimiento utilizan una definición de rendimiento que es insensible a las mejoras de la política en estados poco probables de ser visitados, mientras que la mejora de la política en estos estados poco probables probablemente sea necesaria para obtener una política óptima [18]. En pocas palabras, la medida de rendimiento esta sesgada porque la distribución de estados esta dada por π y por ende no es uniforme.

5.3.1. Optimización de política por región de confianza

En base a lo establecido por Kakade y Langford, Schulman et al. [30] describieron un método de optimización de políticas que garantiza una mejora

monotónica. En base a su nuevo método e introduciendo algunas aproximaciones inventaron el algoritmo de *optimizacion de pol tica por region de con anza* (conocido como TRPO por sus siglas en inglés) que a pesar de las aproximaciones suele obtener una mejora monotónica de la política.

Para explicar el método primero hay que definir los fundamentos del mismo. En primer lugar, se puede expresar el rendimiento J de una política π' en base al rendimiento de otra política π:

$$J(\pi') = J(\pi) + \mathsf{E}_{\pi'}\left[\sum_{t=0}^{1} \gamma^t A_\pi(S_t, A_t)\right] \tag{5.7}$$

donde $\mathsf{E}_{\pi'}$ indica que la trayectoria $(S_0, A_0, S_1, A_1, ...)$ esta dada por la política π' y A_π es la ventaja respecto a la política π (3.17). En base a esto, se define la aproximación local $L_\pi(\pi')$ a $J(\pi')$:

$$L_\pi(\pi') = J(\pi) + \sum_s \mu_\pi(s) \sum_a \pi'(a|s) A_\pi(s, a) \tag{5.8}$$

donde μ_π es la distribución de estados bajo π. A diferencia de (5.7) donde la distribución de estados esta dada por π', por lo que $L_\pi(\pi') \approx J(\pi')$ vale sólo para políticas similares a π.

En el paper [30], Schulman et al. prueban que realizando la siguiente optimización la mejora de J está garantizada:

$$\underset{\pi'}{\text{maximizar}} \; [L_\pi(\pi') - \beta D_{KL}^{max}(\pi, \pi')] \tag{5.9}$$

donde $D_{KL}^{max} = \text{máx}_s \, D_{KL}(\pi(s), \pi'(s))$ es la máxima divergencia KL entre π y π' y β es el coeficiente de penalización de esta divergencia. El problema es que si se utilizara el β recomendado por la teoría, las actualizaciones serían muy pequeñas y por ende el entrenamiento sería lento. Por esto, Schulman et al. utilizan una restricción sobre la divergencia KL entre las políticas en lugar de una penalización, es decir, una *region de con anza*:

$$\begin{aligned} &\underset{\pi'}{\text{maximizar}} \; L_\pi(\pi') \\ &\text{sujeto a } D_{KL}^{max}(\pi, \pi') \leq \delta \end{aligned} \tag{5.10}$$

donde δ es la máxima divergencia tolerada.

En la práctica, no se puede conocer con exactitud en un tiempo razonable ni $L_\pi(\pi')$ ni $D_{KL}^{max}(\pi, \pi')$ porque esto requeriría comparar π y π' en todo el espacio de estados. Por esto, el algoritmo TRPO utiliza estimadores para ambos:

$$L_\pi(\pi') = \mathsf{E}_\pi\left[\frac{\pi'(A_t|S_t)}{\pi(A_t|S_t)} A_\pi(S_t, A_t)\right] \tag{5.11}$$

$$D_{KL}^{max} = \mathsf{E}_\pi[D_{KL}(\pi(S_t), \pi'(S_t))] \tag{5.12}$$

donde $S_t \sim \mu_\pi$ y $A_t \sim \pi(S_t)$ son muestras obtenidas bajo π, (5.11) es el valor esperado de la ventaja pesado por la importancia de la muestra [44] y (5.12) es la divergencia KL promedio sobre la trayectoria dada por π. Se llama *importancia de la muestra* a la razón entre la probabilidad de elegir una acción a en un estado s bajo la nueva política π' comparado con la probabilidad bajo la política π con la que se obtuvo dicha muestra.

De esta manera, el algoritmo de TRPO es:

$$\begin{aligned} \underset{\pi'}{\text{maximizar}} \ & \mathsf{E}_\pi \left[\frac{\pi'(A_t|S_t)}{\pi(A_t|S_t)} A_\pi(S_t, A_t) \right] \\ \text{sujeto a} \ & \mathsf{E}_\pi[D_{KL}(\pi(S_t), \pi'(S_t))] \leq \delta, \end{aligned} \tag{5.13}$$

y la política se mejora en tres pasos:

1. Se aproxima el gradiente natural $\eta \approx F^{-1} \nabla L_\pi(\pi')$ utilizando el método del gradiente conjugado, donde F es la matriz de información de Fisher $F = \nabla^2 \mathsf{E}_\pi[D_{KL}(\pi(S_t), \pi'(S_t))]$ evaluada en $\pi' = \pi$.

2. A partir de la divergencia KL deseada δ se obtiene el paso máximo de actualización α tal que $\theta_{n+1} = \theta_n + \alpha\eta$:

$$\delta \approx \frac{1}{2}(\alpha\eta)^T F(\alpha\eta) = \frac{1}{2}\alpha^2 \eta^T F \eta \tag{5.14}$$

$$\alpha \approx \sqrt{\frac{2\delta}{\eta^T F \eta}}. \tag{5.15}$$

   La aproximación (5.14) de la divergencia KL a partir de F surge del polinomio de taylor de orden 2, la demostración esta disponible en [49].

3. Como el cálculo es aproximado puede que α no cumpla la restricción de TRPO, por lo que se hace una búsqueda iterativa a partir de α y se lo va achicando hasta que se cumple dicha restricción.

TRPO es importante no solo porque mejoró el estado del arte principalmente en locomoción robótica sino también por las fuertes bases teóricas que estableció para la mejora monotónica en métodos basados en optimización de la política.

5.3.2. Optimización proximal de la política

En base a los avances de TRPO, Schulman et al. inventaron un nuevo algoritmo en 2017: *optimizacion proximal de la pol tica* [39], conocido como PPO por sus siglas en inglés. PPO conserva algunos de los beneficios de TRPO, pero es mucho más fácil de implementar, es más general y tiene una eficiencia muestral mucho más alta gracias a su función objetivo que permite múltiples *epocas* (i.e. ciclos) de actualización sobre las mismas muestras.

Sea $\phi_\pi(\pi^0)$ la razón de probabilidad:

$$\phi_\pi(\pi^0) = \frac{\pi^0(a|s)}{\pi(a|s)} \tag{5.16}$$

tal que $\phi_\pi(\pi) = 1$. TRPO maximiza el objetivo sustituto:

$$L^{TRPO}(\pi^0) = \mathsf{E}_\pi\left[\frac{\pi^0(A_t|S_t)}{\pi(A_t|S_t)}A_\pi(S_t, A_t)\right] = \mathsf{E}_\pi[\phi_\pi(\pi^0)A_\pi(S_t, A_t)]. \tag{5.17}$$

Sin una restricción, la maximización de L^{TRPO} llevaría a una actualización demasiado grande de la política. Por esta razón, PPO propone la utilización de otro objetivo que penaliza los cambios a la política que mueven $\phi_\pi(\pi^0)$ muy lejos de 1:

$$L^{CLIP}(\pi^0) = \mathsf{E}_\pi[\text{mín}(\phi_\pi(\pi^0)A_\pi(S_t, A_t),\ \text{clip}(\phi_\pi(\pi^0), 1-\epsilon, 1+\epsilon)A_\pi(S_t, A_t))] \tag{5.18}$$

donde $\epsilon \in (0, 1)$ (normalmente 0,1 o 0,2) y la función clip : $\mathsf{R}^3 \to \mathsf{R}$ es:

$$\text{clip}(x, min, max) = \begin{cases} min & \text{si } x < min \\ x & \text{si } min \leq x \leq max \\ max & \text{si } x > max. \end{cases} \tag{5.19}$$

De esta manera, L^{CLIP} remueve el beneficio de mover $\phi_\pi(\pi^0)$ fuera del intervalo $[1-\epsilon, 1+\epsilon]$ dependiendo de si la ventaja A_π es negativa o positiva respectivamente, por lo que es esencialmente un límite inferior (o pesimista) de L^{TRPO} como se muestra en la Figura 3.

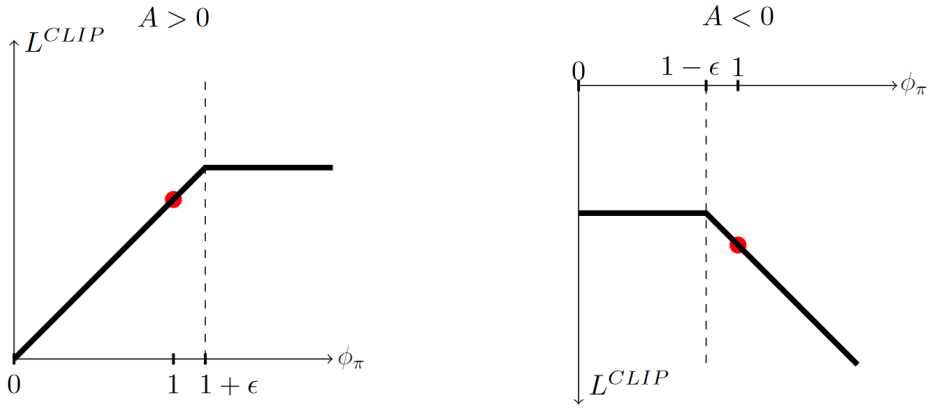

Figura 3: Gráficos de un término (sólo un paso de la trayectoria) de la función L^{CLIP} con respecto a la razón de probabilidad ϕ_π, para ventajas positivas (izq.) y negativas (der.). El círculo rojo en cada gráfico indica el punto de partida de la optimización ($\phi_\pi = 1$). Extraído de [39].

La simplicidad de la función objetivo de PPO la hace no solo fácilmente implementable sino que también es fácilmente diferenciable, por lo que el

algoritmo corre mucho más rápido que TRPO, y fácilmente extensible para arquitecturas que utilizan la misma red neuronal para el actor y el crítico. A pesar de su simpleza, PPO mejoró el estado del arte en aprendizaje por refuerzo en tareas de control continuo y es al día de hoy el método preferido para este tipo de problemas [43, 48].

5.4. Avances en métodos basados en modelo

En base a los avances recientes obtenidos por los métodos libres de modelo, la estrategia de los métodos basados en modelo se alejó de la arquitectura en que el modelo es el componente principal del agente y es en base a este que se planifica y se define la política, y comenzaron a aparecer más arquitecturas híbridas en las que el modelo se utiliza para potenciar y acelerar el aprendizaje de la política libre de modelo [41, 42].

Analizamos también el caso específico de AlphaZero [40], que al ser un agente que se desenvuelve en una ambiente fácilmente modelable (como el ajedrez) utiliza un *modelo dado* (y perfecto) para mejorar la política, a diferencia de los otros métodos que utilizan un *modelo aprendido*.

5.4.1. Expansión del Valor basada en Modelo

Dentro de la rama de algoritmos que utilizan un modelo para generar datos *imaginados* con la intención de reducir la cantidad de datos reales requeridos para el aprendizaje, se destaca el método de Feinberg et al. *Expansion del Valor basada en Modelo* [42], conocido como MVE por sus siglas en inglés.

Su algoritmo aprende un modelo del ambiente $\hat{f}_\pi : \mathcal{S} \to \mathcal{S}$ pero controla la incertidumbre que introduce a las predicciones del valor de los estados limitando la profundidad H de las predicciones que realiza el modelo. El método obtiene su nombre de su estrategia, que consiste en *expandir* la ecuación de Bellman (3.13) para calcular el valor de un estado $V_\pi(S_0)$ con H transiciones imaginadas:

$$V_\pi(S_0) \approx \hat{V}_H(S_0) = \sum_{t=0}^{H\ \ 1} \gamma^t \hat{R}_{t+1} + \gamma^H V_w(\hat{S_H}) \tag{5.20}$$

donde $\hat{S}_t = \hat{f}_\pi(S_{t\ \ 1})$ es un estado imaginado a partir del modelo, $\hat{R}_{t+1} = r(\hat{S}_t, \pi(\hat{S}_t))$ es la recompensa real para el estado imaginado $\hat{S}_t$, H es un hiperparámetro que regula la profundidad de expansión y V_w es el crítico.

Los resultados que obtuvieron fueron muy buenos en cuanto a reducción de la complejidad muestral (i.e. cantidad de interacciones reales requeridas para el aprendizaje) y reforzó las bases para este tipo de métodos que utilizan un modelo aprendido para potenciar al agente libre de modelo.

5.4.2. AlphaZero

AlphaZero [40] es una clase especial de algoritmo, combina funciones de valor, optimización de política, un modelo del ambiente y un árbol de búsqueda monte carlo (MCTS de sus siglas en inglés) para obtener como resultado un algoritmo genérico de aprendizaje por refuerzo (i.e. aplicable a varios dominios sin modificaciones) capaz de llegar a un nivel de juego sobrehumano en pocas horas de entrenamiento, evaluando 1000 veces menos posiciones que otros programas considerados el estado del arte en ajedrez y shogi, partiendo de un comportamiento completamente aleatorio y sin ningún tipo de información del dominio más que las reglas.

Analizamos AlphaZero en esta sección porque es un caso interesante de algoritmo basado en modelo, en el cual el modelo no es aprendido como en el caso de MVE sino que es *dado*. Esto permite la aplicación de MCTS para realizar búsquedas en el árbol de posibles jugadas (i.e. el proceso de decisión de markov subyacente) y optimizar la política en base a los resultados de esta búsqueda.

Más específicamente, AlphaZero utiliza una red neuronal $(P, v) = f_\theta(s)$ con parámetros θ que tiene como entrada el estado s y calcula un vector de probabilidades $P = (p_0, p_1, ..., p_n)$ donde $p_i = \pi(a_i|s)$ para cada acción a_i posible en s, y un valor escalar $v \approx \mathsf{E}[z|s]$ que corresponde al resultado esperado z desde el estado s donde:

$$z = \begin{cases} 1 & \text{si gana} \\ 0 & \text{si empata} \\ -1 & \text{si pierde.} \end{cases}$$

Cada búsqueda consiste en una serie de partidas simuladas contra sí mismo que recorren el árbol desde S_{raiz} hasta una hoja, donde cada simulación elige para un estado s una acción a con un bajo número de visitas, alta probabilidad p y alto valor esperado (que se calcula en base al promedio de v en las hojas de esa rama).

Finalmente, se entrena la red neuronal para maximizar la similitud entre la política π y la distribución final P obtenida a partir de MCTS, y minimizar el error entre el resultado predicho v y el real z.

5.5. Como lidiar con recompensas dispersas

Uno de los grades desafíos en aprendizaje por refuerzo es el de entrenar agentes en ambientes donde las recompensas son muy dispersas. Supongamos que queremos entrenar a un agente para que meta un gol con una pelota de fútbol, la descripción de la tarea es simple:

$$resuelto = \begin{cases} Verdadero & \text{si la pelota esta dentro del arco} \\ Falso & \text{si no.} \end{cases}$$

Sin embargo, si definimos la función de recompensa en base a estas reglas el agente probablemente nunca descubra un estado terminal S_T en que la pelota está dentro del arco debido a que la tarea es muy compleja y la función de recompensa no da ningún tipo de información acerca de si el agente está progresando hacia finalmente lograr meter un gol. Por esto, lo que se hace en la practica es lo denominado *ingenier a de la recompensa* que consiste en diseñar una función de recompensa que *gu e* al agente hacia S_T. El problema de esto es que estamos recompensando al agente por algo que en realidad no nos interesa, que es la trayectoria que recorre el agente para lograr el objetivo, que es simplemente meter un gol sin importar la técnica utilizada.

Otro problema que surge de utilizar una función de recompensa que no representa puramente nuestro verdadero objetivo, es que en muchos casos puede dificultar el aprendizaje si no se la diseña meticulosamente. Volviendo al ejemplo del gol, si utilizo una función de recompensa que premia al agente por acercar la pelota al arco y lo penaliza si la aleja y el agente se encuentra siempre entre el arco y la pelota al comienzo del episodio, probablemente la primer interacción del agente con la pelota sea alejarla, por lo que recibirá una penalización y, a partir de esto, aprenda a no tocar la pelota y nunca descubra que el verdadero objetivo era meter un gol.

5.5.1. Repetición retrospectiva de experiencia

La *repeticion retrospectiva de la experiencia* [33] (conocido como HER por sus siglas en inglés) es una modificación de la *repeticion de experiencia* [5] de la sección 5.2.1 que posibilita el aprendizaje a partir de recompensas dispersas y binarias (como es el caso de meter un gol), permitiendo de esta manera evitar la necesidad de diseñar funciones de recompensa complejas.

El concepto del método se basa en la habilidad que tienen los humanos de aprender casi tanto del fracaso como del éxito al intentar realizar una tarea. En el caso del gol, imaginemos que uno patea la pelota y erra al arco a la derecha. La conclusión a la que llegaría un algoritmo estándar de aprendizaje por refuerzo es que esa secuencia de acciones no lleva a un gol, de lo cual aprendería muy poco. Pero existe otra mirada para este resultado, si el arco hubiese estado un poco más a la derecha la secuencia de acciones sería exitosa. HER permite exactamente este tipo de razonamiento y puede ser combinado con cualquier algoritmo fuera-de-política (Sección 4.2.3).

Para esto, el método requiere que hayan múltiples objetivos; una posible implementación es tratar cualquier estado terminal como un objetivo. El método utiliza *pol ticas universales* [28] las cuales tienen como entrada no solo el estado actual s sino también el objetivo g:

$$a \sim \pi(s, g). \tag{5.21}$$

En base a esto, la idea de HER es repetir las experiencias previas pero con un objetivo g' diferente al que intentaba lograr el agente, por ejemplo, alguno de los objetivos que sí logró alcanzar en el episodio.

Formalmente, cada objetivo $g \in \mathcal{G}$ corresponde a un predicado $f_g : \mathcal{S} \rightarrow \{0, 1\}$ tal que el objetivo del agente es llegar a un estado s que satisface $f_g(s) = 1$. De esta manera, la función de recompensa $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{G} \rightarrow \mathsf{R}$ puede ser simplemente de la forma:

$$r(s_t, a_t, s_{t+1}, g) = -[f_g(s_{t+1}) = 0] \tag{5.22}$$

donde [.] es un predicado. La recompensa sería entonces -1 para todos los estados que no cumplan el objetivo g.

Finalmente, luego de la interacción con el ambiente se almacenan en la memoria de experiencias las transiciones con su objetivo y recompensa original $(s_t, a_t, r_t(g), s_{t+1}, g)$ y las de los objetivos alternativos $(s_t, a_t, r_t(g^0), s_{t+1}, g^0)$ con $g^0 \sim \mathsf{S}(\tau)$, donde S es una estrategia para elegir objetivos alternativos en base a la trayectoria τ.

En sus experimentos, Andrychowicz et al. lograron entrenar un brazo robótico en un simulador para realizar diversas tareas (como levantar un bloque y llevarlo a una posición especifica) y obtuvieron los mismos resultados cargando la política en un robot real sin ningún tipo de modificación.

5.6. Aprendizaje por refuerzo jerárquico

Crear agentes capaces de aprender *pol ticas jerarquicas* es un problema establecido hace mucho tiempo en aprendizaje por refuerzo [15]. Con políticas jerárquicas nos referimos a un conjunto de políticas $\pi_i \in \Pi$ donde cada política π_i aprende a operar en un nivel distinto de *abstraccion temporal* como se ejemplifica en la Figura 4. Al campo del aprendizaje por refuerzo que estudia este tipo de políticas se lo conoce como *aprendizaje por refuerzo jerarquico* [45], o HRL por sus siglas en inglés.

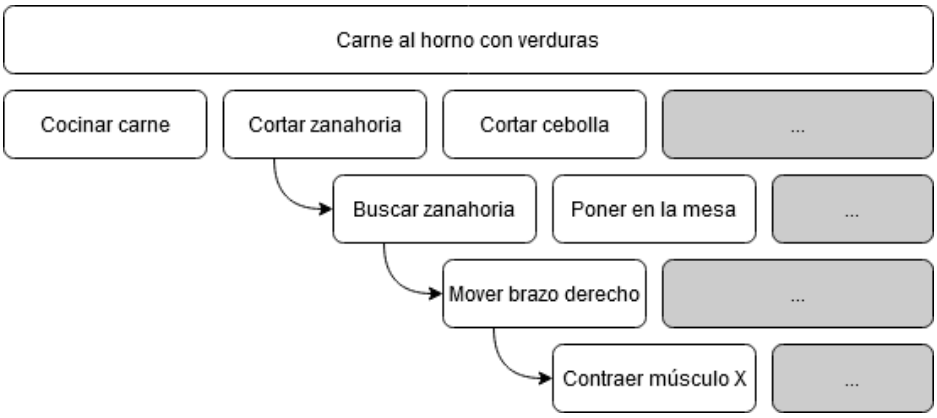

Figura 4: Ejemplo de distintos niveles de abstracción temporal en HRL. Extraído de [45].

Los agentes jerárquicos tienen el potencial de resolver problemas complejos de toma de decisiones secuenciales porque pueden descomponer estas

tareas complejas en un subconjunto de tareas más simples que requieren de secuencias cortas de decisiones.

5.6.1. Actor-Crítico Jerárquico

Para poder aprovechar este potencial y aprender rápidamente, los algoritmos de HRL deben ser capaces de aprender las políticas de los distintos niveles en simultáneo. El problema es que aprender varios niveles de políticas en paralelo es complicado porque es intrínsecamente inestable, los cambios en la política de un nivel pueden causar cambios en las transiciones y recompensas en niveles más altos de la jerarquía. Esto es un problema porque los métodos de aprendizaje por refuerzo necesitan que la distribución de estados a los que llevan las acciones sea estable para poder aprender el valor de dichas acciones.

En este informe nos enfocamos en un método novedoso de HRL capaz de superar estos obstáculos, el *Actor-Cr tico Jerarquico* [35] (HAC por sus siglas en inglés). HAC utiliza el concepto de retrospectiva de la sección 5.5.1 y lo expande al campo de HRL, esto le permite entrenar cada nivel de la jerarquía como si las políticas de los niveles inferiores ya fuesen óptimas. Con este método los autores lograron entrenar exitosamente y por primera vez en paralelo una jerarquía de 3 niveles en espacios de acción y estado continuos. Lo particularmente interesante del método, además de que es efectivo, es que sirve para entrenar políticas de k-niveles.

El método consiste de dos componentes principales:

1. Una estructura jerárquica particular.

2. Un método para aprender múltiples niveles de políticas en paralelo a partir de recompensas dispersas.

La jerarquía consiste en un conjunto de k políticas universales anidadas $\Pi_k = \{\pi_i : \mathcal{S}_i \times \mathcal{G}_i \to \mathcal{A}_i\}$ con $1 \leq i \leq k$ donde el espacio de estados de todas las políticas $\mathcal{S}_i$ es igual al espacio de estados del ambiente ($\mathcal{S}_i = \mathcal{S}, \forall i$), el espacio de objetivos $\mathcal{G}_i$ es un subespacio de $\mathcal{S}$ para todos los niveles excepto el más alto ($\mathcal{G}_i \subseteq \mathcal{S}, 1 \leq i \leq k-1$) y el espacio de objetivos del nivel más alto $\mathcal{G}_k$ es igual al del ambiente ($\mathcal{G}_k = \mathcal{G}$), el espacio de acciones $\mathcal{A}_i$ es el espacio de objetivos del nivel inferior $\mathcal{G}_{i\ 1}$ para todos los niveles excepto el más bajo ($\mathcal{A}_i \subseteq \mathcal{G}_{i\ 1}, 2 \leq i \leq k$), el cual se corresponde con el espacio de acciones del ambiente ($\mathcal{A}_1 = \mathcal{A}$). En resumen:

$$\mathcal{S}_i = \mathcal{S}, \quad \forall i \tag{5.23}$$

$$\mathcal{G}_i = \begin{cases} \mathcal{G} & \text{si } i = k \\ \mathcal{S}^0 \subseteq \mathcal{S} & \text{si no} \end{cases} \tag{5.24}$$

$$\mathcal{A}_i = \begin{cases} \mathcal{A} & \text{si } i = 1 \\ \mathcal{G}_{i\ 1} & \text{si no.} \end{cases} \tag{5.25}$$

De esta manera, la política del nivel más alto π_k toma el estado y objetivo del ambiente (s, g) y produce un sub-objetivo g^{k-1} para la política inferior π_{k-1}. Esta política toma el estado del ambiente s y su sub-objetivo g^{k-1} y produce un sub-objetivo g^{k-2} para el nivel inferior, y así hasta el último nivel. El último nivel π_1 toma el estado del ambiente s y el sub-objetivo g^1 que le dio su nivel superior y produce una acción primitiva a que se ejecuta en el ambiente. Además, cada nivel tiene un cierto número de intentos H denominado *horizonte* para cumplir su objetivo. Cuando un nivel cumple su objetivo o se queda sin intentos el nivel superior le asigna un nuevo objetivo.

En base a esto, definimos la función de transición $T_i : \mathcal{S}_i \times \mathcal{A}_i \to \mathcal{S}_i$ que dado un estado s y una acción a^i devuelve el estado $s^{\emptyset}$ al que pasó el PDM subyacente. Como a^i es el objetivo que se le da a la política inferior π_{i-1}, esta función depende de toda la jerarquía debajo del nivel i denominada Π_{i-1}. Para el nivel más bajo $(i = 1)$, esta función es equivalente a la función de transición del ambiente T, es decir $T_1 = T$.

Para permitir el aprendizaje de múltiples políticas en paralelo a partir de recompensas dispersas HAC utiliza dos tipos de transiciones retrospectivas, *transiciones retrospectivas de accion* y *transiciones retrospectivas de objetivo*. Las transiciones retrospectivas de acción permiten simular que la jerarquía inferior Π_{i-1} es óptima (y por ende estable). La jerarquía óptima Π_{i-1} es simplemente la jerarquía capaz de llegar al objetivo $g^{i-1} = a_t^i$ que se le asigna. Para esto, supongamos la transición $\upsilon_i = (s_t, a_t^i, r_{t+1}^i, s_{t+1}, g^i)$ donde i indica el nivel de la política e $i > 1$ (por lo que $\mathcal{A}_i = \mathcal{S}^{\emptyset}$), a_t^i es el estado al que π_i quería llegar desde el estado s_t (i.e. el sub-objetivo g^{i-1}), s_{t+1} es el estado al que llegó Π_{i-1} en H pasos (que puede o no ser a_t^i) y r_{t+1}^i es la recompensa que recibió π_i en base a su objetivo g^i por llegar a s_{t+1}. Para simular que Π_{i-1} es óptima lo que se hace es modificar la transición υ_i reemplazando el sub-objetivo a_t^i por s_{t+1}, que fue el verdadero resultado obtenido. De esta manera, la transición retrospectiva de acción υ_i^{RA} es $(s_t, s_{t+1}, r_{t+1}^i, s_{t+1}, g^i)$ por lo que la política Π_{i-1} parece óptima (i.e. siempre logra el objetivo que se le asigna).

Las transiciones retrospectivas de objetivo le permiten a cada nivel aprender a partir de recompensas dispersas y binarias. Extendiendo la idea de HER (Sección 5.5.1) al paradigma de HRL, al finalizar los H pasos de la política π_i se copian las H transiciones retrospectivas de acción υ_i^{RA} y se modifican reemplazando el objetivo g^i por alguno de los estados s_{t+1} de la trayectoria y recalculando la recompensa $r_{t+1}^{\emptyset i}$ en base a este nuevo objetivo $g^{\emptyset i}$. Así, las transiciones retrospectivas de objetivo υ_i^{RO} tienen la forma $(s_t, s_{t+1}, r_{t+1}^{\emptyset i}, s_{t+1}, g^{\emptyset i})$. Al ser el objetivo sustituto $g^{\emptyset i}$ un estado encontrado en la trayectoria, al igual que en HER, el agente recibe recompensas por más que no haya logrado el objetivo original g^i.

Las transiciones retrospectivas posibilitan el aprendizaje de HAC pero tienen dos problemas. El primero es que esta estrategia sólo le permite a cada nivel aprender sobre un subconjunto de los estados, el determinado por los estados alcanzables en H pasos por la jerarquía inferior. El segundo, es

que la estrategia no incentiva a las políticas que producen sub-objetivos a proponer trayectorias que los niveles inferiores sean capaces de ejecutar en H pasos con su política actual. Esto es subóptimo porque la política puede terminar prefiriendo trayectorias no ejecutables (i.e. trayectorias compuestas de objetivos imposibles de lograr) por los niveles inferiores.

Para subsanar estos problemas, se agrega un tercer tipo de transiciones llamadas *transiciones de evaluacion de sub-objetivo* cuya finalidad es permitir a un nivel aprender si un sub-objetivo puede ser alcanzado por la jerarquía inferior actual. Estas transiciones están implementadas de la siguiente manera, cada vez que el nivel i produce un sub-objetivo a^i se evalúa ese sub-objetivo con probabilidad λ. Si es evaluado y el nivel $i-1$ no llega a a^i en a lo sumo H pasos, se penaliza al nivel i con una transición de evaluación de sub-objetivo $\upsilon_i^{EO} = (s_t, a_t^i, -H, s_{t+1}, g^i)$ donde $-H$ es la penalización que toma el lugar de r_t. Para un análisis más profundo del efecto de estas transiciones ver [35].

6. Casos de Estudio

En esta sección analizamos algunos de los casos de éxito en aprendizaje por refuerzo profundo de los últimos años. En primer lugar, vemos un caso de aprendizaje por refuerzo profundo aplicado a robótica en el que los investigadores lograron entrenar un programa capaz de controlar una mano robótica para realizar tareas que requieren destreza y control fino. En segundo lugar, analizamos un sistema de inteligencia artificial multiagente con un nivel de juego sobrehumano (i.e. superior al alcanzable por un humano) en un deporte virtual complejo entrenado con aprendizaje por refuerzo distribuido a gran escala.

6.1. Manipulación hábil en la mano

Mientras que para los humanos manipular objetos con la mano es una tarea simple que realizan a diario, para los robots es un desafío complejo. En 2018, OpenAI desarrolló un método de aprendizaje por refuerzo para entrenar políticas capaces de manipular objetos hábilmente con una mano robótica basado en visión por computadora [43]. El entrenamiento lo realizaron por completo en un ambiente simulado pero lograron que las políticas funcionen en un robot real gracias a la introducción de aleatoriedad en determinadas variables físicas y en el aspecto de los objetos. El método no requiere de ningún tipo de demostración humana, pero igualmente emergen naturalmente comportamientos de control humanos típicos, como la coordinación de múltiples dedos y el uso controlado de la gravedad. Además, utilizan un sistema distribuido de aprendizaje por refuerzo para generar toda la experiencia necesaria extremadamente rápido. La política también es capaz de valerse de las imágenes cámaras RGB para estimar la pose (posición y rotación) del objeto, lo cual es sumamente importante para robots que en última instancia están pensados para trabajar fuera de un ambiente controlado como lo es un laboratorio.

6.1.1. Descripción de la tarea

El problema de *manipulacion habil en la mano* consiste en llevar un objeto que se encuentra en la palma de la mano en una pose A a una pose B sin que el objeto se caiga, como se muestra en la Figura 5. En el experimento, cuando se cumple el objetivo B se asigna una nueva pose objetivo C. El episodio termina si se cae el objeto, si pasan 80 segundos sin completar un objetivo o si se completan 50 objetivos.

6.1.2. El hardware

Para el experimento utilizan la "Shadow Dexterous Hand", que es una mano robótica humanoide con 24 grados de libertad que incluye 20 pares de tendones agonista-antagonista. También utilizan el sistema de captura

Figura 5: Mano robótica humanoide entrenada con aprendizaje por refuerzo para llevar un cubo de una pose A a una pose B. Extraído de [43].

de movimiento PhaseSpace para rastrear la posición cartesiana de las puntas de los dedos y opcionalmente la pose del objeto. Finalmente, utilizan 3 cámaras RGB para poder estimar la pose del objeto con una red neuronal convolucional.

6.1.3. La simulación

Para la simulación utilizaron dos motores, el motor físico MuJoCo para simular el sistema físico y el motor gráfico Unity para generar las imágenes con las que entrenaron la red convolucional.

Como la simulación no deja de ser una aproximación de la realidad, existen algunas inexactitudes como la falta de deformación de los cuerpos o que el simulador aplica los torques directo sobre las juntas de los dedos en lugar de utilizar tendones. Estas inexactitudes llevan a un "descalce con la realidad"que dificultan la transferencia de una política del simulador al mundo real.

Para subsanar esto problemas, los investigadores insertaron aleatoriedad en diversas maneras para que la política sea más robusta:

- Para simular el ruido que tienen los sensores en el mundo real, agregaron ruido gaussiano a las observaciones.
- Como es imposible calibrar las constantes físicas del simulador a la perfección, como el rozamiento de la superficie de la mano, se aleatorizaban (dentro de un rango aceptable) al principio de cada episodio.
- Hay varios efectos que ocurren en el robot real que no están simulados como el uso de tendones y la pérdida de seguimiento de los dedos por oclusión. Para esto, agregaron ruido y retraso sobre las acciones de la

política y congelamiento a las observaciones cuando un dedo estaba cerca del otro para simular la oclusión.

- Para que la red convolucional de estimación de pose sea robusta, aleatorizaron los colores, las texturas y la iluminación de los distintos objetos.

6.1.4. La política

Basándose en la teoría de que las distintas variables físicas aleatorias como la fricción o el peso del objeto podrían ser aprendidas en el principio del episodio (y por ende en el mundo real) por una política con estado interno, los investigadores decidieron utilizar una LSTM [10] que es un tipo de red neuronal recurrente (con memoria) cuyo nombre significa *memoria larga de corto plazo*. De esta manera, el agente (en este caso la mano) tendría la habilidad de adaptarse a una variación en el ambiente.

Para entrenar la política utilizaron PPO (Sección 5.3.2) combinado con una técnica llamada Actor-Crítico Asimétrico [37]. Esta técnica consiste en darle al crítico una representación del estado más completa de lo que es posible en la realidad (i.e. las velocidades de cada junta de la mano) para que las estimaciones de la función de valor sean más precisas, esto no trae problemas en el despliegue de la política en el mundo real porque el crítico solo se utiliza durante el entrenamiento.

El espacio de acciones $\mathcal{A}$ corresponde a rotaciones relativas de las juntas (i.e. rotar falange 2 del dedo índice $+10°$). A pesar de que PPO soporta espacios de acción continuos, notaron que el rendimiento era superior si discretizaban los ángulos en 11 acciones.

En cuanto a las recompensas, la recompensa base que utilizaron fue $R_t = d_t - d_{t+1}$ donde d_t y d_{t+1} son los ángulos relativos con respecto a la pose objetivo, antes y después de la transición, respectivamente. Además, daban una recompensa de $+5$ cuando se llegaba a una pose objetivo y una penalización de -20 si se caía el objeto.

6.1.5. Aprendizaje distribuido

Para poder entrenar la política en un tiempo razonable, utilizaron un sistema de entrenamiento distribuido compuesto por 384 máquinas de 16 núcleos, encargadas de correr la simulación y generar experiencia en base a la última política disponible, y una máquina con 8 GPUs para hacer la optimización de parámetros. Esta configuración les permitió generar alrededor de 2 años de experiencia simulada por hora. La elección de PPO se basó también en su fácil escalabilidad y en que requiere poco ajuste de parámetros.

6.1.6. Estimación de la pose a partir de la visión

La política utiliza la pose actual del objeto para operar, pero utilizar el sistema de captura de movimiento para el robot real solo es factible en un

laboratorio. Por esto, utilizaron 3 cámaras RGB y una red neuronal convolucional residual [27] para estimar la posición y rotación del objeto. La red es entrenada sobre imágenes generadas en Unity a partir de experiencia simulada en MuJoCo.

6.1.7. Resultados

La política logró resolver el problema de reorientación de objetos y aprendió naturalmente y sin ningún tipo de demostración varias estrategias conocidas por la comunidad robótica de manipulación hábil en la mano, como rotaciones con un dedo, caminar con los dedos, coordinación entre dedos, uso controlado de la gravedad y aplicación controlada de torsión y traslación.

En el robot real, la efectividad fue de aproximadamente 50 % comparado con la simulación tanto para la estimación de pose con el sistema de captura de movimiento (26 vs 44 objetivos cumplidos en promedio) como para la estimación de pose con la red convolucional (15 vs 30 objetivos cumplidos en promedio). También se comprobó la efectividad de la aleatorización, sin ella el agente solo pudo lograr 1 pose objetivo en promedio antes de fallar.

Por otra parte, se verificó la efectividad de la LSTM en comparación a una capa densa. La LSTM obtuvo el doble de rendimiento. Y finalmente se midió el error de la red convolucional tanto en el ambiente simulado como en el real, el error de cálculo sobre la rotación del objeto fue de 3° en el ambiente simulado vs 5° en el real, y el error de cálculo sobre la posición fue de 3mm en el ambiente simulado y de 9mm en el real.

6.1.8. Conclusiones

El experimento logró un nivel de manipulación hábil en la mano sin precedentes y demostró que el aprendizaje por refuerzo profundo puede resolver problemas de robótica en el mundo real que otros algoritmos no basados en aprendizaje no pueden. Además, en base a este trabajo los investigadores de OpenAI lograron en 2019 entrenar una política capaz de resolver un cubo de Rubik con esta misma mano robótica [48].

6.2. OpenAI Five

OpenAI Five [47] fue el primer sistema de inteligencia artificial en vencer a los campeones mundiales en un e-sport (deporte virtual), específicamente en Dota 2. El juego Dota 2 presenta desafíos interesantes para los sistemas de inteligencia artificial como horizontes temporales extensos, información imperfecta y espacios de estado y acción complejos y continuos, todos desafíos que son cada vez más centrales a los sistemas avanzados de IA.

OpenAI Five utilizó técnicas existentes de aprendizaje por refuerzo y las escaló a niveles sin precedencia, en el orden de 1 millón de muestras por segundo. Para esto desarrollaron un sistema distribuido de entrenamiento

y herramientas para aprendizaje continuo que les permitieron entrenar al agente por un período de 10 meses. El caso es importante porque demuestra que los sistemas de aprendizaje por refuerzo profundo que aprenden por juego contra sí mismos pueden lograr un rendimiento sobrehumano en tareas de alta dificultad.

6.2.1. Dota 2

Dota 2 es un juego situado en un mapa cuadrado con 2 equipos cuyo objetivo es defender su base, las cuales se encuentran en esquinas opuestas del mapa. Cada base tiene una estructura central llamada "nexo", el juego termina cuando un equipo destruye el nexo del otro. Los equipos están formados por cinco jugadores, cada jugador controla un héroe con un conjunto único de habilidades. Durante el transcurso de la partida, ambos nexos generan oleadas de "súbditos", que son personajes controlados por el juego, que caminan hacia la base enemiga atacando lo que encuentren en su camino. Los jugadores obtienen recursos durante la partida, como oro de los súbditos, que pueden utilizar para mejorar su héroe mediante la compra de objetos y mejora de habilidades.

Para poder jugar a Dota 2, un sistema de IA debe resolver varios desafíos:

- Horizontes de tiempo extensos: OpenAI efectúa 20.000 acciones en promedio por partida.
- Estado parcialmente observable: Cada equipo solo puede ver el estado del juego cerca de sus unidades y edificios, el resto del mapa está oculto. El juego avanzado requiere inferencia a partir de datos incompletos, ser capaz de modelar el comportamiento del oponente.
- Espacios de acción y estado altamente dimensionales: OpenAI Five observa al rededor de 16.000 valores en cada paso temporal. El espacio de acciones lo discretizaron, el agente elige entre 8.000 a 80.000 acciones en cada paso temporal. En comparación, el ajedrez requiere al rededor de 1000 valores de estado y 35 acciones válidas en cada paso.

6.2.2. La implementación

Mientras que el motor de Dota 2 opera a 30 fps, OpenAI Five actúa solo cada 4 fotogramas lo cual equivale a un paso temporal. Cada paso temporal, el agente recibe una observación del juego y ejecuta una acción.

Ciertas mecánicas del juego no son controladas por la política sino que las manejan pequeños fragmento de código, como el orden en que los objetos son comprados y la mejora de habilidades.

Algunas de las variables del ambiente se aleatorizan al principio del episodio como los héroes o los objetos que compran, para que la política sea más robusta.

Para entrenar el agente utilizaron PPO, probaron distintos métodos pero PPO fue el único que aprendía algo. Como aproximador utilizaron un bloque central LSTM compartido con dos cabezales densos (i.e. dos perceptrones multicapa conectados al mismo LSTM), uno para la política y otro para la función de valor. Además, utilizaron una función de recompensa especialmente diseñada por expertos en el dominio de Dota 2 junto con GAE [29], que es un método de reducción de la varianza para la señal de ventaja. Como optimizador utilizaron ADAM (Sección 5.1.3).

El sistema distribuido estaba compuesto por cuatro componentes:

- Los *rollout workers* son nodos que solo tienen CPUs cuyo único trabajo es correr la simulación y generar muestras.
- El *controller* es un almacenamiento que tiene la última versión de los parámetros de la red neuronal.
- Las *forward pass GPUs* son nodos que contienen la red neuronal pero su única función es calcular las acciones en base a las observaciones que obtienen de los rollout workers. Cada cierto tiempo le piden al controller la nueva versión de la red.
- Los *optimizers* reciben las transiciones de los rollout workers y calculan el gradiente. Luego, se hace el promedio entre todos los optimizers, se actualiza la red neuronal y se actualiza el controller.

Es sumamente interesante que la simulación y el agente corran en máquinas distintas, de esta manera podían utilizar una mayor cantidad de rollout workers corriendo varias simulaciones en paralelo y hacer todos los forward pass en lotes de 60 observaciones sobre pocas GPUs.

6.2.3. Adaptación del modelo por cirugía

A medida que el proyecto avanzaba, el código y el ambiente fueron cambiando por tres razones:

- A medida que fueron experimentando, aplicaron cambios al proceso de entrenamiento y a la arquitectura de la red neuronal.
- En base al principio de ingeniería de comenzar simple e ir agregando complejidad de a poco, a medida que avanzó el proyecto se fue agregando soporte para distintas mecánicas del juego en el espacio de acciones y observaciones.
- Cada cierto tiempo Dota 2 recibe actualizaciones que modifican mecánicas del juego como los héroes o los objetos. Para poder jugar contra humanos el agente debía jugar en la última versión.

En base a estos factores, el equipo implementaba modificaciones al modelo cada un período de 2 semanas. Como el proyecto duró varios meses y el costo de entrenar la red era bastante alto, era inviable entrenarlo de cero luego de cada modificación.

Lo que denominan *cirug a* es un conjunto de herramientas para hacer operaciones de transformación del estilo *Net2Net* [25] sobre un modelo π_θ para obtener un nuevo modelo $\hat{\pi}_{\hat{\theta}}$ compatible con el nuevo ambiente y con el mismo rendimiento, por más que θ y $\hat{\theta}$ sean distintos en tamaño y semánticamente. Formalmente, esto implica garantizar que ambos modelos implementan la misma función de estados a acciones:

$$\hat{\pi}_{\hat{\theta}}(s) = \pi_\theta(s), \forall s. \tag{6.1}$$

En total realizaron 20 cirugías exitosas (y varios más intentos fallidos) durante los 10 meses que duró el proyecto. Este método les permitió realizar un entrenamiento continuo sin pérdida de rendimiento.

6.2.4. Resultados

Luego de 10 meses de entrenamiento a una escala sin precedentes OpenAI Five derrotó a los campeones mundiales en un encuentro al mejor de 3 (2-0) y al 99,4 % de los jugadores durante una exposición de 3 días.

6.2.4.1. Evaluación humana

Por más que ganarle a los campeones del mundo 2-0 en un mejor de 3 es un gran logro e indica que OpenAI Five tiene un nivel de juego sumamente alto, no demuestra que tenga un amplio entendimiento acerca de la variedad de desafíos que puede presentar la comunidad humana. Para esto, hicieron *OpenAI Five Arena* donde pusieron a OpenAI Five disponible para que la comunidad de Dota 2 pueda jugar contra el agente en partidas competitivas durante un lapso de 3 días. En ese lapso se jugaron 7257 partidas de las cuales el agente ganó 99.4 %.

La versión final de OpenAI Five tenía un estilo de juego similar al humano en general, pero con algunas diferencias interesantes. Los humanos suelen asignar héroes a determinadas áreas del mapa y solo los reasignan ocasionalmente, OpenAI Five movía los héroes mucho más frecuentemente. El agente desarrolló un entendimiento muy preciso acerca del riesgo de realizar un ataque con poca vida. Finalmente, OpenAI Five utilizaba más los recursos y las habilidades con tiempo de recarga alto, los humanos suelen guardarlos por si se presenta una mejor oportunidad para usarlos.

6.2.4.2. Validación de la cirugía

Para poder evaluar los efectos de la cirugía, al finalizar el experimento entrenaron un nuevo agente al que llamaron ReRun y lo compararon con

la versión final de OpenAI Five. ReRun tomó 2 meses de entrenamiento y superó el nivel de OpenAI Five, con una tasa de victorias de 98 %.

En conclusión, la cirugía les permitió cambiar el ambiente y la política todas las semanas sin tener que comenzar el entrenamiento de cero pero tuvo un efecto secundario, el modelo terminó en un nivel más bajo de habilidad que el que obtuvo el modelo entrenado de cero. Lograr continuar experimentos largos sin perder rendimiento es un área interesante para investigar en el futuro.

6.2.4.3. Tamaño de lote y escalabilidad

En un mundo ideal, el entrenamiento con el doble de datos sería el doble de rápido, pero en la realidad no es así. Cuando entrenaron ReRun con un tamaño de lote de 983 mil muestras compararon el tiempo que llevaba llegar a cierto nivel de habilidad con un tamaño de lote de 123 mil muestras, 8 veces menos datos. ReRun entrenó solamente 2,5 veces más rápido con 8 veces más información. Si bien la escalabilidad no es lineal, este factor de 2,5 les permitió entrenar ReRun en 2 meses en lugar de 5.

6.2.4.4. Calidad de los datos

Como las partidas pueden tomar hasta 2 horas en completarse y los parámetros de la política se actualizan cada 1 minuto aproximadamente, era inviable entrenar sobre trayectorias enteras porque las transiciones del principio del episodio eran de políticas muy antiguas. Luego de estudiar el efecto de este "estancamiento" de los datos, decidieron que la mejor opción era enviar trayectorias cada 1 minuto, el tiempo de actualización de parámetros, para minimizar la degradación que esto produce.

También estudiaron el impacto de la reutilización de datos en el entrenamiento y descubrieron que utilizar los datos más de una vez puede provocar una ralentización de hasta un factor de 2, y utilizarlos 8 veces directamente frena el aprendizaje.

En conclusión, los experimentos indican que la calidad de los datos es mucho más importante que el cómputo utilizado y que pequeñas degradaciones sobre la calidad de datos pueden tener un efecto severo sobre el aprendizaje.

6.2.4.5. Asignación del crédito a largo plazo

Dota 2 tiene dependencias temporales muy extensas, los agentes tienen que ejecutar planes que se desarrollan a lo largo de varios minutos, que corresponden a miles de pasos temporales. Por esto, este experimento es una plataforma única para evaluar este tipo de comportamiento.

Los investigadores estudiaron los efectos del horizonte temporal sobre el cual el agente descuenta las recompensas y observaron que los agentes con un horizonte mayor (6 a 12 minutos) eran muy superiores a los demás.

A medida que los problemas a resolver crezcan en complejidad, el pensamiento y la planificación a largo plazo van a ser cada vez más importantes para el comportamiento inteligente.

6.2.5. Conclusiones

Cuando se los escala correctamente, los métodos modernos de aprendizaje por refuerzo profundo demuestran ser lo suficientemente poderosos como para lograr un nivel sobrehumano en e-sports.

La cirugía resultó ser un método efectivo pero no perfecto para entrenar políticas durante períodos largos de tiempo y sobre ambientes cambiantes.

Por más que el experimento de enfoca solo en Dota 2, los investigadores creen que los resultados van a ser de aplicación general y que el método es capaz de resolver cualquier ambiente continuo de suma cero en equipos que pueda ser simulado en paralelo y a gran escala.

La posibilidad de escalar métodos de aprendizaje por refuerzo va a ser más y más importante a medida que aumente la complejidad de los problemas a resolver.

7. Conclusiones

En este informe es expuso el relevamiento y análisis que se realizó en el campo del aprendizaje por refuerzo desde sus inicios y fundamentos hasta sus avances más recientes, explorando algunas de las muchas ramas que existen en este campo. Si bien el relevamiento pudo no haber sido exhaustivo, creemos que es suficiente como para comprender el estado del arte y algunos métodos muy prometedores como el actor-crítico jerárquico.

Si bien el trabajo previo a la redacción de este informe fue mucho más extenso y abarcativo, para mantener una línea de pensamiento clara para un lector no experimentado en el área limitamos la información a lo relevante para poder analizar los casos expuestos y a algunas ramas del campo que consideramos importantes como la DQN o interesantes desde nuestro punto de vista, como HAC. Algunos de los temas interesantes que no cubre este informe son: Métodos generalizados de aprendizaje por refuerzo como Path Consistency Learning [36], métodos basados en teoría de distribuciones como C51 [34], métodos de entrenamiento libres de gradiente como Evolution Strategies [38] y métodos de exploración basados en motivación intrínseca como VIME [31]. El campo del aprendizaje por refuerzo es muy prometedor y se está expandiendo rápidamente.

Por nuestro lado, aplicaremos el conocimiento adquirido en la realización de este trabajo a crear un agente capaz de jugar al e-sport Rocket League con un nivel suficientemente alto como para ganar el torneo de agentes programados por la comunidad. Rocket League es fútbol pero con autos voladores propulsados a cohete donde cada jugador controla un auto, es un ambiente complejo por múltiples razones: Es un problema de control en un espacio continuo, el agente debe aprender las dinámicas físicas del ambiente para poder planear correctamente, debe lograr un nivel de precisión de control elevado debido a la velocidad de juego, las recompensas son escasas ya que el objetivo es simplemente meter goles y el agente debe ser capaz de coordinar con su equipo en partidas de más de un jugador.

Esperamos que este trabajo sirva como una introducción rápida y robusta para aquellos lectores nuevos a la disciplina, y como un disparador para aquellos más experimentados.

Referencias

[1] 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.

[2] Boris Polyak. ⟪Some methods of speeding up the convergence of iteration methods⟫. En: *Ussr Computational Mathematics and Mathematical Physics* 4 (dic. de 1964), págs. 1-17. DOI: 10.1016/0041-5553(64)90137-5.

[3] K. Fukushima. ⟪Neocognitron: A self-organizing neural network model for a mechanish of pattern recognition unaffected by shifts in position⟫. En: *Biological Cybernetics* 36 (1980), págs. 193-202.

[4] R. S. Sutton. ⟪Temporal Credit Assignment in Reinforcement Learning⟫. Tesis doct. University of Massachusetts, Dept. of Comp. e Inf. Sci., 1984.

[5] Long J. Lin. ⟪Self-Improving Reactive Agents Based on Reinforcement Learning, Planning, and Teaching⟫. En: *Machine Learning* 8.3 (1992), págs. 293-321.

[6] Christopher J. C. H. Watkins y Peter Dayan. ⟪Q-learning⟫. En: *Machine Learning* 8.3 (mayo de 1992), págs. 279-292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[7] R. J. Williams. ⟪Simple statistical gradient-following algorithms for connectionist reinforcement learning⟫. En: *Machine Learning* 8 (1992), págs. 229-256.

[8] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. USA: John Wiley & Sons, Inc., 1994. ISBN: 0471619779.

[9] Gerald Tesauro. ⟪Temporal Difference Learning and TD-Gammon⟫. En: *Commun. ACM* 38.3 (mar. de 1995), págs. 58-68. ISSN: 0001-0782. DOI: 10.1145/203330.203343. URL: https://doi.org/10.1145/203330.203343.

[10] Sepp Hochreiter y Jürgen Schmidhuber. ⟪Long short-term memory⟫. En: *Neural computation* 9.8 (1997), págs. 1735-1780.

[11] Jordan B. Pollack y Alan D. Blair. ⟪Why did TD-Gammon Work?⟫ En: *Advances in Neural Information Processing Systems 9*. Ed. por M. C. Mozer, M. I. Jordan y T. Petsche. MIT Press, 1997, págs. 10-16. URL: http://papers.nips.cc/paper/1292-why-did-td-gammon-work.pdf.

[12] J. N. Tsitsiklis y B. Van Roy. ⟪An analysis of temporal-difference learning with function approximation⟫. En: *IEEE Transactions on Automatic Control* 42.5 (1997), págs. 674-690.

[13] Y. LeCun y col. ⟪Gradient-based learning applied to document recognition⟫. En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[14] 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.

[15] Richard S. Sutton, Doina Precup y Satinder Singh. ⟪Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning⟫. En: *Arti cial Intelligence* 112.1 (1999), págs. 181-211. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(99)00052-1. URL: http://www.sciencedirect.com/science/article/pii/S0004370299000521.

[16] Vijay R. Konda y John N. Tsitsiklis. ⟪Actor-Critic Algorithms⟫. En: *Advances in Neural Information Processing Systems 12*. Ed. por S. A. Solla, T. K. Leen y K. Müller. MIT Press, 2000, págs. 1008-1014. URL: http://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf.

[17] Richard S Sutton y col. ⟪Policy Gradient Methods for Reinforcement Learning with Function Approximation⟫. En: *Advances in Neural Information Processing Systems 12*. Ed. por S. A. Solla, T. K. Leen y K. Müller. MIT Press, 2000, págs. 1057-1063. URL: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf.

[18] Sham M. Kakade y John Langford. ⟪Approximately Optimal Approximate Reinforcement Learning.⟫ En: *ICML*. Ed. por Claude Sammut y Achim G. Hoffmann. Morgan Kaufmann, 2002, págs. 267-274. ISBN: 1-55860-873-7. URL: http://dblp.uni-trier.de/db/conf/icml/icml2002.html#KakadeL02.

[19] Chuong B. Do. *The Multivariate Gaussian Distribution*. 2008. URL: http://cs229.stanford.edu/section/gaussians.pdf. (visitado: Julio 2020).

[20] Hado V. Hasselt. ⟪Double Q-learning⟫. En: *Advances in Neural Information Processing Systems 23*. Ed. por J. D. Lafferty y col. Curran Associates, Inc., 2010, págs. 2613-2621. URL: http://papers.nips.cc/paper/3964-double-q-learning.pdf.

[21] T. Tieleman y G. Hinton. *Lecture 6.5 - RMSProp*. Technical report. COURSERA: Neural Networks for Machine Learning, 2012.

[22] M. G. Bellemare y col. «The Arcade Learning Environment: An Evaluation Platform for General Agents». En: *Journal of Arti cial Intelligence Research* 47 (jun. de 2013), págs. 253-279. ISSN: 1076-9757. DOI: 10.1613/jair.3912. URL: http://dx.doi.org/10.1613/jair.3912.

[23] Volodymyr Mnih y col. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[24] Diederik P. Kingma y Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[25] Tianqi Chen, Ian Goodfellow y Jonathon Shlens. *Net2Net: Accelerating Learning via Knowledge Transfer*. 2015. arXiv: 1511.05641 [cs.LG].

[26] Hado van Hasselt, Arthur Guez y David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: 1509.06461 [cs.LG].

[27] Kaiming He y col. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[28] Tom Schaul y col. «Universal Value Function Approximators». En: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. por Francis Bach y David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, jul. de 2015, págs. 1312-1320. URL: http://proceedings.mlr.press/v37/schaul15.html.

[29] John Schulman y col. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. arXiv: 1506.02438 [cs.LG].

[30] John Schulman y col. *Trust Region Policy Optimization*. 2015. arXiv: 1502.05477 [cs.LG].

[31] Rein Houthooft y col. *VIME: Variational Information Maximizing Exploration*. 2016. arXiv: 1605.09674 [cs.LG].

[32] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. arXiv: 1609.04747 [cs.LG].

[33] Marcin Andrychowicz y col. *Hindsight Experience Replay*. 2017. arXiv: 1707.01495 [cs.LG].

[34] Marc G. Bellemare, Will Dabney y Rémi Munos. *A Distributional Perspective on Reinforcement Learning*. 2017. arXiv: 1707.06887 [cs.LG].

[35] Andrew Levy y col. *Learning Multi-Level Hierarchies with Hindsight*. 2017. arXiv: 1712.00948 [cs.AI].

[36] Ofir Nachum y col. *Bridging the Gap Between Value and Policy Based Reinforcement Learning*. 2017. arXiv: 1702.08892 [cs.AI].

[37] Lerrel Pinto y col. *Asymmetric Actor Critic for Image-Based Robot Learning*. 2017. arXiv: 1710.06542 [cs.RO].

[38] Tim Salimans y col. *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*. 2017. arXiv: 1703.03864 [stat.ML].

[39] John Schulman y col. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[40] David Silver y col. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI].

[41] Théophane Weber y col. *Imagination-Augmented Agents for Deep Reinforcement Learning*. 2017. arXiv: 1707.06203 [cs.LG].

[42] Vladimir Feinberg y col. *Model-Based Value Estimation for E cient Model-Free Reinforcement Learning*. 2018. arXiv: 1803.00101 [cs.LG].

[43] OpenAI y col. *Learning Dexterous In-Hand Manipulation*. 2018. arXiv: 1808.00177 [cs.LG].

[44] 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.

[45] Yannis Flet-Berliac. ⟪The Promise of Hierarchical Reinforcement Learning⟫. En: *The Gradient* (2019).

[46] B. Mehlig. *Arti cial Neural Networks*. 2019. arXiv: 1901.05639 [cs.LG].

[47] OpenAI y col. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019. arXiv: 1912.06680 [cs.LG].

[48] OpenAI y col. *Solving Rubik's Cube with a Robot Hand*. 2019. arXiv: 1910.07113 [cs.LG].

[49] Agustinus Kristiadi. *Natural Gradient Descent*. URL: https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/. (visitado: Agosto 2020).

[50] Russ Salakhutdinov. *Deep Reinforcement Learning and Control: Policy Gradient I*. URL: http://www.cs.cmu.edu/~rsalakhu/10703/Lectures/Lecture_PG.pdf. (visitado: Julio 2020).

[51] 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.