Universitat Oberta de Catalunya

uoc.edu

Agente Sonic. Deep Reinforcement Learning

Cristóbal Daniel Alemán de León
Grado en Ingeniería Informática
Inteligencia Artificial

Joan M. Nunez Do Rio
Carles Ventura Royo

06/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Agente Sonic. Deep Reinforcement learning*
Nombre del autor:	*Cristóbal Daniel Alemán de León*
Nombre del consultor/a:	*Joan M. Nuez Do Rio*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Machine Learning, Deep Q-Learning, Artificial Neural Network*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El aprendizaje por refuerzo es una rama de la inteligencia artificial la cual estudia algoritmos capaces de hacer que los sistemas aprendan a realizar tareas automáticamente sin utilizar algoritmos tradicionales. Se basan en un sistema de recompensas donde las acciones correctas reciben una recompensa positiva. Dentro de estos algoritmos encontramos el **Deep Q-Network** que utiliza redes neuronales profundas para entornos complejos como son los videojuegos.

La finalidad de este proyecto es la creación de un agente **DQN** que aprenda a superar diferentes niveles de un videojuego apoyándose en el reto propuesto por el equipo de OpenIA en 2018. En este reto se propone la creación de agentes capaces de superar niveles diferentes de los usados para entrenarlos. OpenIA nos proporciona, a partir de la librería Gym Retro, las herramientas necesarias para llevar a cabo dicho reto. Estas consisten entornos que disponen de observaciones, acciones y recompensas para superar diferentes niveles del juego **Sonic the Hedgehog ™**.

El agente desarrollado será finalmente capaz de tomar acciones que le permitan obtener un mayor avance horizontal dentro de cada nivel. Los entornos donde se evalúa al agente son diferentes de los entornos de entrenamiento con lo que de esta forma se comprueba en los resultados de la generalización realizada por el algoritmo de **Deep Learning** en un entorno desconocido.

Abstract (in English, 250 words or less):

Reinforcement learning is a branch of artificial intelligence that studies algorithms capable of making the systems learn to do tasks automatically without using traditional algorithms. They are based on an achievement system in which the right actions are positively rewarded. Within these algorithms, we can find **Deep Q-Network,** which uses profound neural networks for complex environments such as video games.

The purpose of this project is the creation of a **DQN** agent that learns to overcome different levels of a video game based on the challenge proposed by the team OpenIA in 2018. In this challenge, the creation of agents able to overcome different levels than the ones used for training them is suggested. Using the Gym Retro library, OpenIA provides us with the tools needed to carry this challenge out. These tools consist of observations, actions, and rewards for completing levels of the game **Sonic the Hedgehog™.**

In the end, the agent developed will be able to take actions that allow it to obtain a larger horizontal movement within each level. The environments where we evaluate the agent are different from the training environment. This way, we check the results of the generalization made by the algorithm of **Deep Learning** in an unknown environment.

Índice

Lista de figuras

Lista de Tablas

Lista de Ilustraciones

1. Introducción

1.1 Contexto y justificación del Trabajo

En la actualidad la **IA** ha ganado mucho protagonismo en diferentes áreas de la sociedad y se pueden encontrar sistemas inteligentes en muchos campos, como, por ejemplo: en los negocios, redes sociales, smartphones, etc. [3], que varían su funcionalidad desde predecir tendencias futuras, al reconocimiento facial en videos e imágenes.

La **inteligencia artificial (IA)** es un campo dentro de la informática que estudia la creación de programas que puedan mostrar comportamientos considerados como inteligentes [1,2]. Aunque a día de hoy no se ha definido lo que se puede considerar inteligente, es ampliamente aceptada la definición de que un sistema de **IA** debería ser capaz de aprender y razonar como un ser humano.

Los primeros retos formales en este campo fueron propuestos por Alan Turing en 1950 con el test que lleva su apellido, el test de Turing, donde establece unas reglas para considerar que una máquina pueda considerarse inteligente utilizando una conversación en el lenguaje natural [4,5]. Pero el campo IA abarca mucho más que la disciplina que se conoce como **procesamiento del lenguaje natural** (**PLN**) [7].

El **Machine Learning (ML)** es una disciplina dentro de la **IA,** que investiga las formas con las que un software pueda aprender automáticamente tareas específicas sin necesidad de ser programadas [6]. Uno de sus creadores fue Arthur Samuel en 1956, que construyó un programa capaz de aprender a jugar a las damas [8,9]. Este programa aprendió de la experiencia jugando contra jugadores humanos y contra si misma durante largo tiempo, llegando a vencer al campeón de damas de Connecticut.

Dentro del **ML** existen diversas categorías en base al problema a resolver. Este proyecto se basa en e**l Aprendizaje por refuerzo (*Reinforcement Learning*).** Este tipo de aprendizaje se basa en un sistema de recompensas a partir de prueba y error [10]. Aunque conoce los resultados no sabe qué acciones son las más optimas. Posee una fase de entrenamiento donde él agente (sistema que toma la decisión) toma acciones y aprende de ellas según la recompensa obtenida. Este tipo de aprendizaje está basado en la psicología conductista. Está es la base de este proyecto.

Avanzando un escalón más en **ML,** llegamos al **Deep learning (DL**), y más concretamente al **Deep Reinforcement learning** [11,12,14]. En este nivel se usan redes neuronales profundas **(*Deep neural networks*)** con algoritmos de **RL**, siendo **DQN** uno de los algoritmos más destacados.

En el caso particular de los videojuegos, se proporciona un escenario perfecto donde investigar y experimentar con este tipo de algoritmos, ya que nos provee de un entorno (imágenes del juego, posición de los objetos, velocidad, etc.), un agente (el objeto que controlamos dentro del juego), y unas recompensas según las reglas del juego (puntuación por sobrevivir, por recolectar objetos, por desplazamiento, etc.). Además, disponemos de una gran gama de géneros: conducción, plataformas, deportes, estrategia, etc. Esto hace que las posibilidades y los desafíos a la hora de utilizar RL sean muy variados.

El objetivo de este trabajo es la creación de un agente capaz de aprender utilizando el algoritmo de **DQN** [15]. La investigación se apoya en el reto propuesto por OpenAi (compañía de desarrollo e implementación de IAs) en 2018.

Este reto propone el entrenamiento de un agente utilizando como entornos diferentes niveles de los videojuegos Sonic The Hedgehog, Sonic The Hedgehog 2, Sonic 3 & Knuckles**,** de la videoconsola MegaDrive/Genesis (consola de 16 bits desarrollada por Sega en 1988) para investigar el comportamiento del agente en entornos desconocidos para él, pero que siguen la misma lógica de recompensas**.**

La motivación principal es la de adquirir las habilidades y conocimientos en el campo del **DRL**, que sirve como base para el desarrollo e investigación de nuevas formas de suministrar información a un agente y mejorar los algoritmos existentes.

1.2 Objetivos del Trabajo

- **Objetivos Generales**
    - Investigación y aprendizaje de técnicas de **DRL.**
    - Implementación del algoritmo **DQN**
    - Realizar experimentos y analizar los resultados
- **Objetivos Específicos**
    - Implementación de agentes usando el sistema que nos proporciona el reto de OpenAI, que nos provee del entorno, las recompensas, los estados y las acciones
    - Implementación de agentes capaces de finalizar el objetivo propuesto basados en **DQN**
    - Investigación de las mejoras sobre el algoritmo de **DQN** e implementación.
    - Análisis de resultados en diferentes niveles de validación.

1.3 Enfoque y método seguido

Para la realización de la investigación se han seguido unos pasos, intentando optimizar el tiempo en la medida de lo posible, dando lugar a diferentes fases bien definidas.

1. Se investiga el marco teórico sobre el **ML,** más concretamente el **RL,** afianzando los conocimientos para abordar el desarrollo y la investigación del proyecto.

2. Se estudia el reto propuesto en el proyecto. Existen muchos tipos de formas de solventar el problema con distintos algoritmos. Finalmente se utiliza **DQN** frente a otros, como **A3C** o **PPO** ya que parece se obtienen mejores resultados.

3. Se investiga el algoritmo **DQN** y sus mejoras conocidas como **Rainbow DQN,** que serán explicadas con más detalles en el capítulo 2. Muchas de estas técnicas conllevan un cambio importante en las bases del **DQN.**

4. Búsqueda de información sobre frameworks y librerías de **RL** y su utilización. Optando por las siguientes herramientas:

   Python: Actualmente uno de los lenguajes de programación más utilizados en el campo de la IA, posee muchas bibliotecas y requiere mucho menos aprendizaje y código que otros lenguajes.

   Keras: Es una *framework* de ***DL*** de alto nivel en Python que puede ejecutarse por encima de ***TensorFlow.***

   TensorFlow: Librería desarrollada por Google que proporciona una interfaz para la creación de algoritmos de ***ML,*** que se usará como backend de **keras.**

   Anyrl-py: Librería que implementa diferentes algoritmos de **RL** de forma modular.

5. Desarrollo de prototipos básicos de **DQN** llevando a la práctica el marco teórico, base fundamental para comprender el funcionamiento y abordar los errores que surjan.

6. Desarrollo de prototipos utilizando algunas mejoras que incluye **Rainbow DQN**.

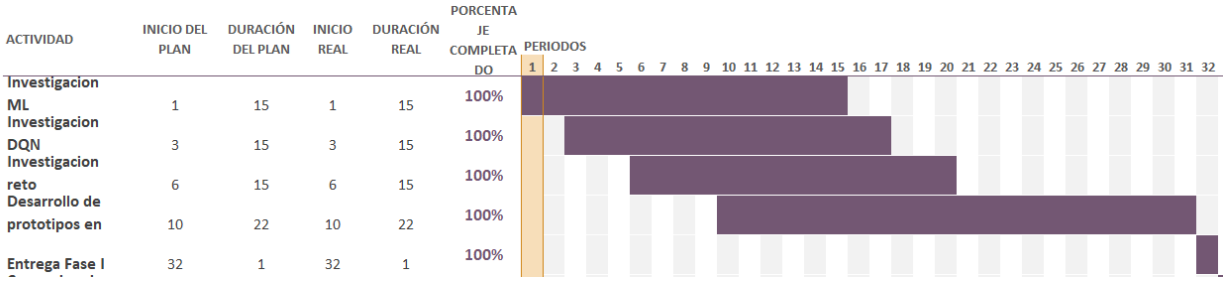
7. Desarrollo del **agente** con el que se obtiene los datos del experimento. Dicho agente implementa completamente **Rainbow DQN** y se entrena en diferentes niveles detallados en apartado de resultados.

8. Conclusiones del trabajo donde se muestran los resultados de los niveles de validación y la explicación de estos.

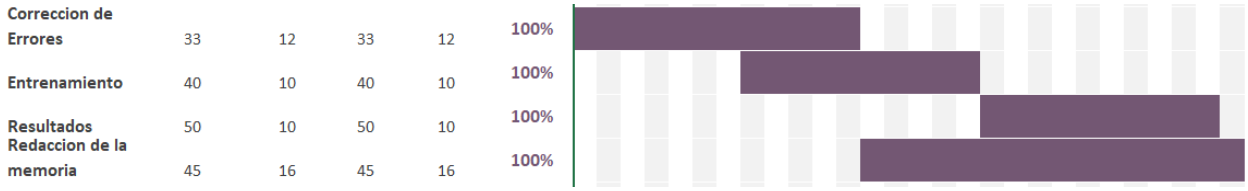
1.4 Planificación del Trabajo

16/03/2020 - 31/03/2020	Implementación del prototipo
1/04/2020 - 16/04/2020	Pruebas y correcciones
17/04/2020	Entrega Fase I
18/04/2020 – 10/05/2020	Entrenamiento del agente y obtención de resultados
10/05/2020 – 17/05/2020	Comparación de resultados
17/05/2020	Entrega Fase II
19/05/2020 – 02/06/2020	Redacción de la memoria
03/06/2020 – 10/06/2020	Elaboración de la presentación
11/06/2020 – 22/06/2020	Defensa pública

Fase I



Fase II



1.5 Breve sumario de productos obtenidos

- Repositorio GitHub con algunas implementaciones.
- Una Memoria del TFG.
- Un video de presentación y defensa del TFG.

1.6 Breve descripción de los otros capítulos de la memoria

- Capítulo 2: Explicación teórica donde se repasan los aspectos más relevantes.
- Capítulo 3: Estado del arte en videojuegos utilizando IA
- Capítulo 4: Explicación de las condiciones del reto.
- Capítulo 5: Metodología seguida para realizar los experimentos.
- Capítulo 6: Experimentación y obtención de resultados
- Capítulo 7: Conclusiones y valoraciones finales.

2. Machine Learning

El aprendizaje automático (*Machine Learning*), es la rama de la inteligencia artificial destinada a técnicas que permitan a un software o máquina aprender automáticamente. Se pueden agrupar en tres grupos diferentes, aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

El aprendizaje supervisado (*Supervised Learning)* es una técnica que logra obtener una función a entrena a partir de un modelo etiquetado. La finalidad es la de que sea capaz de clasificar un conjunto de datos que no pertenece al conjunto de utilizado en la obtención de la función. El SL se suele usar en problemas de clasificación (identificación de dígitos, animales, etc.) y en problemas de regresión (problemas de predicciones). Los algoritmos más habituales son: Árboles de decisión, clasificación Naïve Bayes, SVM, etc.

El aprendizaje no supervisado (*Unsupervised Learning)* no posee un conocimiento a priori, no se tienen modelos etiquetados como ocurre en el aprendizaje supervisado. La finalidad es categorizarlo a partir de las observaciones de forma automática según sus características, regularidades, correlaciones y categorías. Son usados en problemas de *clustering*.

El aprendizaje por refuerzo (*Reinforcment Learning*) se basa en mejorar la respuesta del modelo usando retroalimentación. El algoritmo aprende observando el mundo y en respuesta a sus acciones recibe una recompensa. No podemos clasificarlo de aprendizaje supervisado porque no se basa en un conjunto de datos etiquetados, sino en las respuestas a las acciones. Tampoco es un aprendizaje no supervisado ya que la acción en un estado tiene una recompensa establecida.

Aprendizaje por Refuerzo

Un agente aprende a comportarse en base a la experiencia de un entorno, a partir de sus acciones y comprobando las consecuencias en base a una medida o noción de recompensa. La función de recompensa regula la calidad de la acción tomada por el agente.

Se compone principalmente de los siguientes elementos:

- **Agente**: Toma las decisiones (acciones) sobre el entorno.
- **Entorno**: Es el “mundo” donde se encuentra el agente.
- **Recompensa**: Es el valor de la acción tomada por un agente.
- **Estados**: Son todos los posibles lugares donde se puede encontrar al agente dentro del entorno.

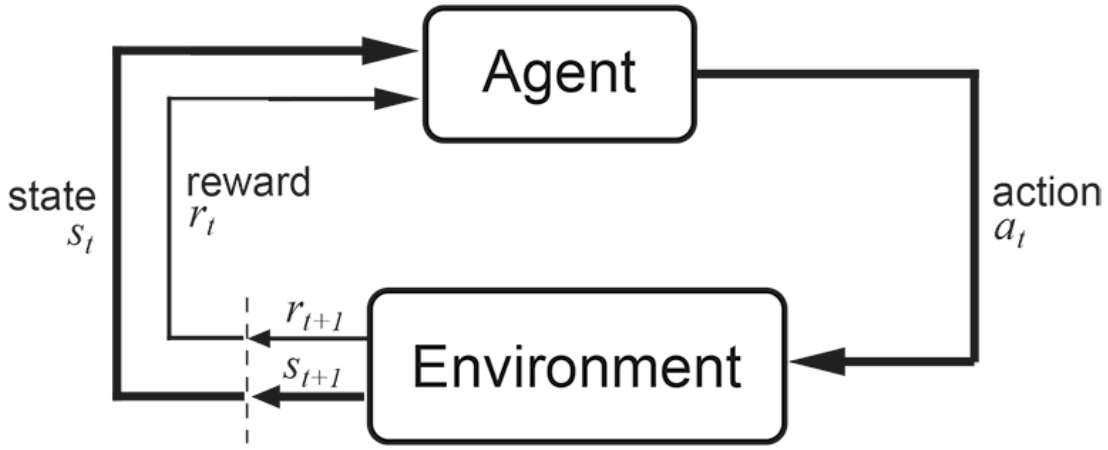


Figura 1: Interacción entre agente y entorno

Un agente toma una acción en el estado actual en instante de tiempo t. Para cada instante de tiempo t (*step*) el agente recibe una recompensa y una representación del entorno (observación). En este instante el agente escoge la acción, dentro de su conjunto de acciones posibles, que tenga una mayor probabilidad de maximizar la recompensa y se avanza un paso nuevamente en el tiempo. En la *figura 1* se observa gráficamente el proceso.

La función de recompensa es fundamental para el aprendizaje del agente ya que debe ajustarse al objetivo a alcanzar. Decide como de bueno o malo es tomar una acción en un determinado estado. Una mala función de recompensa generará comportamientos no deseados en el agente.

Las recompensas pueden variar en acciones a corto o largo plazo. Es decir, una acción inmediata puede generar una gran recompensa, pero no necesariamente será la acción más óptima a largo plazo. Para solucionar este problema se utiliza un factor de descuento gamma γ que multiplica a la suma de las recompensas. Gamma toma valores entre 0 y 1. Cuanto más cerca de cero tendrán más valor las recompensas cercanas, mientras que cuanto mas se acerque a 1 priorizará las recompensas futuras.

Existen dos métodos de procesar las recompensas. En el método **Monte Carlo** soló se tiene en cuenta la recompensa al final de cada episodio y en ese momento se modificará el modelo con dicha información. En el método de **Diferencia Temporal** (*Temporal Difference Learning, TD*) se tiene en cuenta la recompensa en cada paso actualizando el modelo y seleccionado la siguiente acción que maximice la recompensa esperada.

El resultado de maximizar la recompensa es establecer lo que se llama una **política** que corresponde a un mapeo entre los estados del agente y las

acciones. Actualizándose en función de las acciones y recompensas aprendidas de la observación del entorno.

2.1 Q-Learning

Los valores Q es una relación estado/acción, representada como $Q(s,a)$ siendo *s* un estado y *a* una acción, que retorna un valor esperado de tomar una acción en un estado concreto. Los valores Q se actualizan durante todo el proceso del algoritmo utilizando la experiencia adquirida para encontrar una política optima. **La política** se define como el argumento máximo de los valores Q de tomar una acción $\Pi^* = argmax_a Q^*(s,a)$

El algoritmo Q-Learning es ***off-policy***. Esto quiere decir que la política que sigue cuando explora y actualiza los valores Q no es la misma que realiza en el proceso final cuando los valores Q han sido establecidos. Se debe a que en la fase de exploración debe buscar otras alternativas que puedan mejorar los valores Q.

2.1.1 Exploración vs explotación

Una de las formas de resolver el problema de la exploración, ya que en un primer momento no tenemos valores Q, es tomar acciones aleatorias y actualizar dichos valores en función a la recompensa que obtenemos. Para garantizar que el agente no solo explora los estados más cercanos al inicio, se utiliza la técnica Epsilon-*Greedy.*

La técnica del *Epsilon-Greedy* consiste en tomar acciones aleatorias en lugar de obtener la acción a partir de la política. En los primos pasos la probabilidad de elegir una acción aleatoria es alta, lo que permite explorar el entorno y aprender sin explotar una política aprendida. A medida que avanza el algoritmo la aleatoriedad decrece utilizando finalmente la política que se obtiene de los valores Q aprendidos de la exploración.

$$\mathrm{C}_t(\mathrm{S}_t) = \begin{cases} \mathrm{a}_t & probaility\ (1-\varepsilon) \\ random & probability\ \varepsilon \end{cases}$$

2.1.2 Representación

Podemos representar los valores Q en una matriz Estado x Acciones como se puede observar en la *Tabla 1*.

Q	a_0	a_1	a_2
S_0	$Q(S_0, a_0)$	$Q(S_0, a_1)$	$Q(S_0, a_2)$
S_1	$Q(S_1, a_0)$	$Q(S_1, a_1)$	$Q(S_1, a_2)$

Tabla 1: Representación de los valores Q en una matriz estado x acciones

La tabla de valores Q se actualiza cada vez que el agente recibe una recompensa del entorno en base a una acción tomada. Esta representación puede modelizarse matemáticamente en un **proceso de decisión de Markov (MDPs).**

2.1.3 Procesos de decisiones de Markov

Es un formalismo matemático para la toma de decisiones en entornos con incertidumbre que cumplen la propiedad de **Markov**. Esta propiedad dictamina que la probabilidad de pasar al siguiente estado sólo depende del estado actual $P(X_{t+1}|X_t|) = P(X_{t+1}|X_1, \dots., X_t)$.

Los Procesos de decisión de Markov (*Markov decision process*) permiten formalizar los problemas de RL. Estós pueden ser representados por una tupla <S, A, P, R, γ > siendo S una lista de estados, A un conjunto de acciones $\{a_1 \dots a_n\}$, P una matriz de transición de estado tal que $P : A\ x\ S \rightarrow \Pi(s)$, R la función de recompensa y γ el factor de descuento.

Una política para un MDP se define como una asociación $\Pi : S \rightarrow A$ que selecciona una acción por cada estado dependiendo únicamente del estado actual y siendo independiente del tiempo, lo que se denomina estacionario.

La función estado-valor retorna la recompensa esperada a largo plazo denotada con E_Π desde un estado siguiendo la política Π.

$$V^{\Pi}(S) = E_{\Pi}\left\{\sum_{t=0} \gamma^t r_t \mid S_0 = S\right\}$$

La función acción-valor devuelve el valor esperado E_{Π} al ejecutar una acción *a* en un estado *s* que sigue la política Π : $q^{\Pi}(s\,,a) = E_{\Pi}[\,G_{\mathrm{t}}\,|S_{\mathrm{t}} = s, A_{\mathrm{t}} = a]$

La función óptima de estado-valor representa el mayor valor sobre todas las políticas $V^{\Pi}(S) = max_{\Pi}V^{\Pi}(s)$

La función óptima de la acción-valor es el máximo valor que puede obtener al haber tomado una acción *a* sobre todas las políticas, lo que permite elegir dentro del proceso de decisión de Markov aquellas acciones que son las más óptimas $Q(s,a) = max_{a}q^{\Pi}(s,a)$

2.1.4 Ecuación de Bellman (TD)

Finalmente, se aplica la ecuación de Bellman $V(s) = max_{a}\big(R(s,a) + \gamma V(s')\big)$ para encontrar las acciones más óptimas a tomar dentro del proceso de decisiones de **Markov**. Está es la ecuación principal para el aprendizaje por refuerzo [2]. Conociendo todos sus parámetros, esta ecuación sería suficiente para que un agente avanzara por un entorno con la máxima recompensa.

Nuestro problema es estocástico ya que para cada estado existen probabilidades de tomar diferentes acciones, lo que remplaza el valor por la suma de las probabilidades sobre todas las acciones y estados posibles. La ecuación toma la forma de $V(s) = max_{a}(R(s,a) + \gamma \sum_{s'} P(s,a,s') \cdot V(s')$

Esta ecuación nos garantiza convergencia. Usando el algoritmo **value iteration** esta ecuación se resuelve de forma iterativa eligiendo un valor inicial aleatorio.

Finalmente podemos dividir la fórmula en:

- El valor óptimo de un estado, lo que nos permite rellenar la tabla de valores Q: $V(s) = \ max_{a}Q(s,a)$

- La política óptima desde un estado S: $\Pi = \ argmax_{a}Q(s,a)$

2.2 Deep Reinforcement Learning

El algoritmo de **Q-Learning** es suficiente para solucionar problemas de aprendizaje por refuerzo donde existe con una cantidad de estados limitados en tamaño y complejidad, pero insuficiente cuando el espacio de estados y el

conjunto de acciones es muy grande. Como solución a este problema se utilizan redes neuronales profundas.

2.2.1 Redes Neuronales Artificiales

La neurona artificial es un modelo matemático que simula el comportamiento de la neurona biológica. La neurona artificial posee unas entradas *g* a las que se le asocia un peso *w* y una salida controlada por una función de activación *k* tal que $s = k(\sum_i w_i g_i)$

El valor neto de la entrada es la suma de cada entrada multiplicadas por el peso correspondiente. Este valor pasa a la función de activación. Está puede ser de muchos tipos: sigmoidal, tangente hiperbólica, Relu, etc, la cual nos devolverá un valor de salida.

El conjunto de varias neuronas artificiales es lo que llamamos una red neuronal artificial o perceptrón multicapa, siendo está la clase de red más simples. Se usan para problemas de clasificación.

Las redes neuronales convolucionales, **CNN,** son un tipo de red diseñadas para funcionar de forma muy similar a las neuronas de la corteza visual primaria del cerebro humano. Estas redes están formadas por capas de filtros convolucionales de una o más dimensiones. Su finalidad es la de extraer características de las imágenes.

2.2.2 Deep Q Network

El equipo de **DeepMind** utilizó redes neuronales para aproximar la función Q. Utilizan una imagen como entrada a la red que corresponde a la observación del entorno, y como salida las acciones que puede tomar el agente. Para la toma de una acción a partir de la observación soló necesitamos tomar el argumento máximo de la salida. Para calcular los valores Q usamos el máximo de la salida usando la ecuación de **Bellman**.

Las diferencias con **Q-Lerning** son las siguientes:

1. Uso de una arquitectura de redes neuronales, donde se usan unas capas de convoluciones seguida de una red para clasificación.

2. Utilizar lotes de experiencias para el entrenamiento en lugar de paso a paso.

Se puede extender DQN con diferentes métodos que permiten un mayor refuerzo en el aprendizaje. Estas mejoras se conocen como **Rainbow DQN** [19].

Rainbow DQN

DDQN (Double Deep Q Network)

Utiliza una red a la que llamaremos **target** con una frecuencia de actualización más lenta que la principal. Esta red será la que estime el valor Q del siguiente estado. Esto reduce la dependencia circular de la función Q y permite un entrenamiento más estable.

Dueling DQN

Desglosa la función *Q(s,a)* en dos funciones separadas: *V(s)* el valor del estado *s* y *A(s,a), que es* la ventaja de tomar una acción sobre todas las demás acciones posibles [5]. Para ello se separan en capas aisladas internas de la DQN y se suman ambas a la salida $Q(s,a) = V(s) + A(s,a)$

Piorized Experience Replay

En el algoritmo de DQN elige un buffer para entrenar aleatoriamente entre todas las experiencias guardadas en la memoria. Este mecanismo es ineficiente ya que muchos de los registros podrían ser inútiles para el entrenamiento.

Un mecanismo más adecuado sería elegir el buffer de entrenamiento en función a una prioridad [24]. La prioridad más común se basa en la pérdida de los valores Q.

NoisyNet

La exploración del entorno se realiza de forma aleatoria. Esta solución al ser probabilista dista mucho de ser óptima. La mejora propuesta es la eliminación de la probabilidad de elegir una acción al azar añadiendo ruido a las neuronas de la red. De este modo logramos, que si la red aún no conoce la solución a este estado se produzca un ruido que genere una acción aleatoria hasta que la red aprenda. Una vez la red determina un resultado seguro este ruido no afecta a la toma de la acción.

N-Step

Avanzamos n pasos a partir de un estado, haciendo el cálculo de la recompensa total por avanzar esos n estados y lo guardamos en la memoria de experiencias.

Distributional DQN (C51)

Se basa en utilizar una distribución multimodal para el cálculo de las recompensas en vez del valor esperado Q. Se usa una variable aleatoria *Z(s,a)* para remplazar a *Q(s,a).* Quedando la ecuación distributiva de **Bellman** como $Z(s,a) = R(s,a) + \gamma Z(S',A')$

El algoritmo gana su nombre por el número 51, que representa los valores discretos que parametrizan la distribución de valores *Z*. El 51 se trata como un número mágico ya que se obtiene empíricamente dando un rendimiento óptimo.

3. Estado del Arte

3.1 IA en videojuegos

La **IA** ha estado presente en el mundo de los videojuegos desde que estos nacieron, como se indica en la introducción de este proyecto, juegos como el Ajedrez o el Pong ya disponían de ciertos sistemas inteligentes.

En la época de los 70 hubo un gran avance para la inteligencia artificial en los videojuegos. Space Invaders contaba con enemigos más inteligentes y Pac-Man disponía de un sistema de búsqueda.

En la época de los 90 juegos como Starcraft o Age of Empires, juegos de genero RTS, eran capaces de crear tácticas o estrategias en función del jugador y gestionar recursos.

Otra clase de juegos como Metal Gear Solid o Half-Life implementaban en enemigos técnicas de combate conjuntas o sistema de alertas modificando la ruta del enemigo si esté escuchaba un ruido.

Posteriormente no solo se usó la **IA** para el comportamiento de enemigos, sino que se utilizaba para generar sistemas procedurales o diseño de misiones reduciendo así el trabajo de los diseñadores.

3.2 RL

En 2015, Google desarrollo un agente de **IA** capaz de vencer a un jugador profesional del juego de tablero Go. Inicialmente este agente entrenaba contra jugadores humanos. La evolución fue AlphaGo zero que se entrenó jugando

contra sí misma y logrando las capacidades del agente anterior en menos tiempo.

Blizzard y Deepmind han colaborado para realizar investigaciones de aprendizaje profundo utilizando el videojuego Starcraft. AlphaSta**r** fue el primer agente en poder vencer a jugadores profesionales sin ninguna ventaja.

OpenAI Five es un agente creado para el juego Dota 2 (juego MOBA) que fue capaz de vencer a los campeones del mundo de este juego en 2018. Este agente utiliza redes **LSTM**.

3.3 Resultados del Reto

El equipo de Dharmaraja quedo en primer lugar usando una variante del algoritmo **PPO** con algunas mejoras [33]. Utilizo imágenes RGB descartando la escala de grises. Un espacio de acciones un poco aumentado con combinación de botones más comunes y una finalmente una función de recompensa aumentada. Obtuvo una puntuación media de 4692.

El equipo Mistake, en segundo lugar, uso el algoritmo de **Rainbow DQN** haciendo algunas modificaciones para aumentar el rendimiento [35]. Las mejoras consistieron en un mejor valor de n para el n-step. La adición de una capa más convolucional a la red lo que ralentizo el entrenamiento pero reforzo el aprendizaje, y una actualización de la red target más bajo. Obtuvieron una puntuación media de 4446

Finalmente, en tercer lugar, quedó el Equipo Aborg. Utilizó una variante de **PPO** con muchas mejoras: mas niveles de entrenamiento utilizando otros niveles de otras videoconsolas, que seguían una estructura idéntica a los juegos propuestos. Una arquitectura de red diferente y un ajuste de hiperparámetros diseñados específicamente para el aprendizaje rápido. Obtuvo una clasificación de 4430

Se presentaron 229 soluciones del reto [36] pero obtuvieron peores puntuaciones en la generalización del aprendizaje. Para calcular las puntuaciones se utilizo un conjunto de pruebas de cinco niveles creados por un editor de niveles. De esta forma se aseguraron de que el agente no podía conocer de antemano el nivel.

4. Retro Contest

El concurso trataba de la transferencia de aprendizaje midiendo la capacidad de los algoritmos de RL para generalizar a partir de la experiencia previa. Típicamente los algoritmos de RL se prueban en el mismo entorno donde se entrenan, lo que favorece un buen sistema. Por ello se propuso experimentar como se comporta un agente entrenado en un entorno diferente con el mismo conjunto de acciones y la misma función de recompensa.

Se utilizó **Gym Retro** una plataforma para la investigación de aprendizaje por refuerzo que integra juegos de Atari y Sega. Para el concurso se utilizaron 3 juegos diferentes de la misma plataforma: *Sonic The Hedgehog*, Sonic The Hedgehog 2*, Sonic 3 & Knuckles*. Todos de la misma temática y sin mucha diferencia en las acciones que puede tomar el jugador, de los cuales se disponen de 47 niveles de entrenamiento y 11 niveles de validación elegidos para más compatibilidad con la función de recompensa. De esta forma se eliminan mucho de los objetivos de un jugador para centrarnos únicamente en el avance horizontal.

Existe similitud entre los niveles del mismo nombre donde solo varia el Act. El objetivo del jugador es llegar al final del nivel evitando caer en trampas, esquivando o matando enemigos. También están las anillas que permiten sobrevivir al daño si tienes acumuladas, soltándolas en dicho momento y pudiendo recuperar algunas en un breve periodo de tiempo antes de que desaparezcan. También sirven para ganar vidas extras. En los niveles Act3 hay que enfrentarse a un enemigo final para completar el nivel. Este enemigo ha sido eliminado de los mapas para simplificar el sistema de aprendizaje.

Gym Retro provee de un sistema donde en cada paso según la acción tomada, dentro del conjunto de acciones que también provee, nos devuelve:

- Una observación del entorno.
- Una recompensa.
- Una variable booleana que indica si el juego a finalizado.
- Un array de información sobre el juego (posición del agente, número de vidas, etc.) esta información varía según el juego. Se usa como mera información y no como parte del entrenamiento del agente.

4.1 Estados de Finalización

Existen tres formas de finalizar un episodio:

- El agente completa el nivel con éxito, para ello debe llegar al final del nivel (recorrer una distancia horizontalmente)

- El agente pierde una vida. En cada nivel existen enemigos o trampas donde si el agente cae pierde una vida.

- 4500 pasos que equivalen aproximadamente a 5 min del tiempo de juego real a 60 fps (18000 fotogramas).

4.2 Observaciones

En cada paso el entorno avanza 4 **frames** y retorna una observación en forma de imagen RGB de 24 bits de 320 píxeles de ancho y 224 pixeles de alto [224,320,3]. El entorno es estocástico porque posee un **sticky frameskip**.

El **frameskip** es una técnica que permite omitir fotogramas aumentando el rendimiento del sistema donde se ejecuta. En un videojuego normalmente el **frameskip** repite una acción tomada por el jugador n veces. En este entorno puede repetirse n + 1 veces con una probabilidad de 0.25 lo que en la siguiente observación la acción se repetirá 1 vez menos.

4.3 Conjunto de Acciones

Las acciones que puede tomar el agente son una combinación de botones codificadas en una vector booleano de longitud 12, donde cada elemento corresponde a un botón del mando de la MegaDrive *Ilustración 1*. Donde 1 significa presionado y 0 no presionado. Se ignoran las combinaciones no válidas, como acceder al menú o presionar arriba y abajo al mismo tiempo.

Ilustración 1: Mando MegaDrive

4.4 Función de recompensa

La recompensa que gana el agente es proporcional al desplazamiento en el eje horizontal, positivo cuanto más se acerque a la meta o negativo si se aleja. La suma total es de 9000. También se incluye un bonus de tiempo que comienza en 1000 y va disminuyendo, lo que premia que supere el nivel lo antes posible. La recompensa máxima que se puede obtener es de 10000.

5 Metodología

5.1 Entorno

Debido al tiempo que requiere el entrenamiento, se entrenará el agente en tres niveles diferentes y se estudiará el resultado en los niveles de validación propuestos en el reto como se ilustra en la *Tabla 2.*

Entrenamiento		**Validación**	
SonicTheHedgehog-Genesis	GreenHillZone.Act1	SonicTheHedgehog-Genesis	SpringYardZone.Act1
SonicTheHedgehog2-Genesis	MetropolisZone.Act1	SonicTheHedgehog-Genesis	GreenHillZone.Act2
SonicTheHedgehog-Genesis	LabyrinthZone.Act1	SonicTheHedgehog-Genesis	StarLightZone.Act3
		SonicTheHedgehog-Genesis	ScrapBrainZone.Act1
		SonicTheHedgehog2-Genesis	MetropolisZone.Act3
		SonicTheHedgehog2-Genesis	HillTopZone.Act2
		SonicTheHedgehog2-Genesis	CasinoNightZone.Act2
		SonicAndKnuckles3-Genesis	LavaReefZone.Act1
		SonicAndKnuckles3-Genesis	FlyingBatteryZone.Act2
		SonicAndKnuckles3-Genesis	HydrocityZone.Act1
		SonicAndKnuckles3-Genesis	AngelIslandZone.Act2

Tabla 2: Niveles usados en la experimentación

5.2 Episodios

Los episodios en sí no son relevantes para el diseño así, que únicamente se basará en los números de pasos totales.

5.3 Observaciones

Para aumentar el rendimiento del aprendizaje se escala la imagen a una proporción de [128,128,1] donde se transforma el canal RGB en escala de grises.

Finalmente se almacenan 4 **frames** (ya que corresponden a la misma acción) y se envían a la red quedando, finalmente la entrada de está como una tupla (1,128,128,4). El primer elemento indica el tamaño del **batch.**

5.4 Acciones

Trabajar directamente con el espacio de acciones original se vuelve complejo para el aprendizaje, por lo que se discretiza, quedando finalmente las siguientes acciones:

- LEFT
- RIGHT
- LEFT, DOWN
- RIGHT, DOWN
- DOWN
- DOWN, JUMP (B)
- JUMP (B)

5.5 Recompensa

No se realiza ninguna modificación sobre la recompensa.

5.6 Diseño e Hiperparámetros

Se realizarán tres experimentos diferentes pero relacionados secuencialmente. En el primer experimento se entrenará al agente sobre un nivel y se comprobara los resultados sobre ese mismo nivel y algunos de validación, comprobado cual es la recompensa media obtenida en ellos.

En el segundo experimento, con los pesos ya entrenados del experimento anterior, entrenaremos al agente en otro nivel y volveremos a realizar mediciones sobre las recompensas obtenidas tanto en los niveles entrenado como en algunos de validación.

Finalmente, en el tercer experimento, se utilizarán los pesos ya entrenados en de los niveles del experimento uno y el experimento dos para entrenar al agente en otro nivel diferente a los anteriores. Finalmente comprobaremos las recompensas medias que recibe en cada nivel de validación.

Como hiperparámetros para el algoritmo **Rainbow DQN** usaremos 3 pasos para **n-step**, la recomendación para este parámetro es entre 3 y 4 pasos. Se usa la experiencia priorizada utilizando el **TD error** como valor de priorización. Se utiliza una capacidad para **50000** experiencias y un **batch size** de 32 registros que se obtendrán de esas experiencias priorizadas para entrenar la red.

Como optimizador de la red se usa **Adam** con un *learning rate* de **1e^-4.** Este optimizador tiene la ventaja de adaptar su ratio de aprendizaje en función de la distribución de los parámetros.

6 Resultados

6.1 Primer estudio

Training

Juego: SonicTheHedgehog-Genesis
Nivel: GreenHillZone.Act1

Parámetros:
- Número de pasos: 4500000

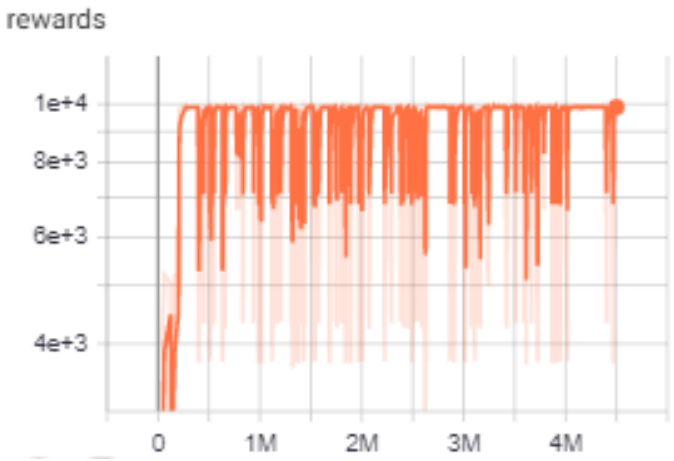


Figura 2: Recompensas obtenidas en el entrenamiento en GreenHillZone.Act1

Como se observa en la *Figura 2* sobre los **360.8K steps** el agente obtiene la recompensa máxima y el aprendizaje parece estabilizarse.

Validación

Nivel	**Recompensa media en 10 episodios**
GreenHillZone.Act1	9868.327
GreenHillZone.Act2	4016.612
MetropolisZone.Act3	450.468

Tabla 3: Recompensas medias en niveles de validación del experimento 1

Como se comprueba en la *Tabla 3* el entorno en el que fue entrenado *GreenHillZone.Act1* recibe cerca de la máxima recompensa (10000). En el

entorno con características similares *GreenHillZone.Act2* obtenemos también una recompensa elevada para un nivel nunca visto por el agente.

Por el contrario, se comprueba que en entornos diferentes donde no existe similitud con el entorno de entrenamiento *MetropolisZone.Act3,* se obtienen peores resultados. Es lógico pensar que la extracción de características de la red neuronal convolucional se ha adaptado a niveles con un estilo similar al entrenado.

6.2 Segundo Estudio

Training

Parámetros:
- Numero de pasos: 2000000

Juego: SonicTheHedgehog2-Genesis
Nivel: MetropolisZone.Act1

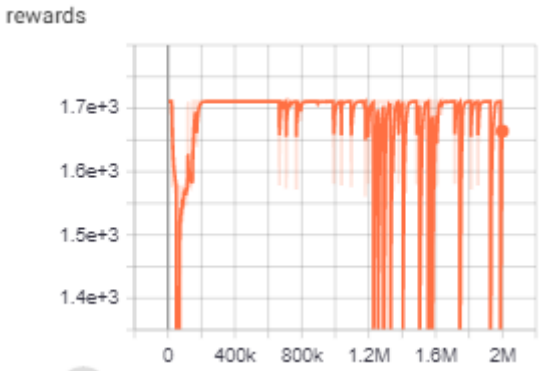


Figura 3: Recompensas obtenidas en el entrenamiento en MetropolisZone.Act1

Se observa en la *Figura 6* que el algoritmo no es capaz de obtener una recompensa superior aproximadamente de 1700. Lo que nos indica que debe existir un punto donde el agente no es capaz de tomar acciones capaces de maximizar mas allá de este valor. Esto puede deberse a la función de recompensa, que siempre premia las acciones de avance en el eje horizontal impidiendo que el agente pueda explorar otras opciones.

Validación

Nivel	Recompensa media en 10 episodios
GreenHillZone.Act1	1894.856
GreenHillZone.Act2	3324.635
MetropolisZone.Act3	1710.852
MetropolisZone.Act3	327.659

Tabla 4: Recompensas medias en niveles de validación del experimento 2

Se observa en la *Tabla 4* como el entrenamiento ha afectado al rendimiento en estos niveles en comparación con los pesos de la red del primer experimento. Esto es debido a que el algoritmo intenta generalizar el comportamiento perdiendo el sobreajuste que tenia sobre el nivel *GreenHillZone.Act1*y mejorando en *MetropolisZone.Act3*

6.3 Tercer estudio

Training

Parámetros:
- Numero de pasos: 2000000

Juego: SonicTheHedgehog-Genesis
Nivel: LabyrinthZone.Act1

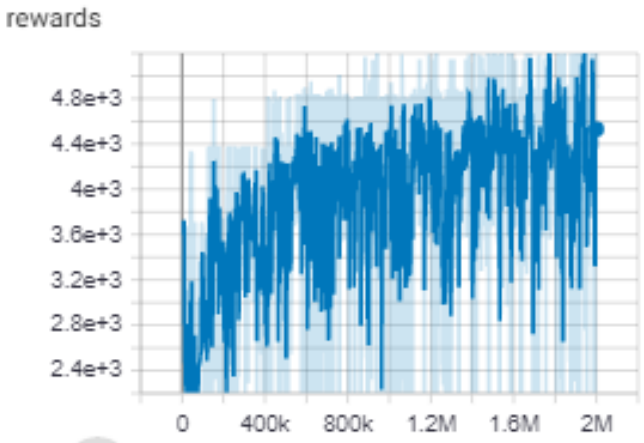


Figura 4: Recompensas obtenidas en el entrenamiento en LabyrinthZone.Act1

Se observa en la *Figura 11* como el algoritmo mejora su recompensa por episodio en la fase de entrenamiento. A diferencia del experimento anterior, que inmediatamente se establece en torno a un valor.

Al establecerse el máximo número de pasos a 2000000 es imposible determinar cuando el algoritmo se estabiliza sobre un valor de la recompensa.

Validación

Nivel	**Recompensa media en 10 episodios**
LabyrinthZone.Act1	4227.153
GreenHillZone.Act1	1105.649
MetropolisZone.Act3	1069.532

Tabla 5: Recompensas medias en niveles de validación del experimento 2

Como se observa en la *Figura 12*, hay un nivel de recompensa casi de un 50% del total, lo que significa que el agente ha logrado superar aproximadamente medio nivel. Como consecuencia, en mapas donde anteriormente fue entrenado como se ve en las *Figura 13 y Figura 14* la recompensa obtenida es muy baja llegando a superar un poco más de un 10%.

6.4 Resultados Finales

Nivel	Media de 10 episodios	Experimento 1	Experimento 2
SpringYardZone.Act1	423.723	-	-
GreenHillZone.Act2	1885.731	4016.612	3324.635
StarLightZone.Act3	1206.684	-	-
ScrapBrainZone.Act1	610.626	-	-
MetropolisZone.Act3	1412.766	450.468	327.659
HillTopZone.Act2	192.265	-	-
CasinoNightZone.Act2	1885.731	-	-
LavaReefZone.Act1	475.163	-	-
FlyingBatteryZone.Act2	865.255	-	-
HydrocityZone.Act1	589.910	-	-
AngelIslandZone.Act2	612.058	-	-

Tabla 6: Resultados y medias de 10 episodios en cada nivel de validación

7 Conclusiones

El entrenamiento sobre diferentes niveles tiende a generalizar al agente perdiendo ajustes sobre los niveles entrenados. Sin embargo, la recompensa media es muy baja en muchos niveles de validación como se observa en la *Tabla 3*, esto puede ser debido a varios factores, como los niveles de entrenamiento utilizados y las características de éstos a nivel visual.

En trabajos similares se comprueba un ligero aumento de las recompensas utilizando más niveles de entrenamientos diferentes. Aun así, los agentes nunca son capaces de superar un nivel que no conocen [33]. La máxima puntuación obtenida en el reto fue de **4692** utilizando el algoritmo **PPO** con algunas modificaciones.

El segundo puesto del reto fue una versión de **Rainbow DQN**, mismo algoritmo empleado en este proyecto, donde obtiene una puntuación de **4446**. Lograron estos resultados modificando los parámetros de actualización de la red target en 1024, añadiendo una capa mas a la red neuronal y establecieron el N-Step en 4.

El objetivo de crear un agente se ha logrado y ha sido capaz de superar el nivel inicial de entrenamiento. No obstante, no ha sido capaz de llegar a una puntuación parecida a otros trabajos similares. El tiempo de aprendizaje y de implementación llevó más tiempo de lo esperado por ello se dispuso de un tiempo menor para experimentación.

Utilizando **framework** de **RL** con los algoritmos ya implementados soló sería necesario centrarse en la experimentación y en los resultados, obteniendo de esta forma, mayores comparativas, aunque esto hubiera significado una menor comprensión de los algoritmos a nivel de desarrollo.

Como futuras líneas de investigación queda aún mucho por realizar. Se puede estudiar este mismo algoritmo modificando parámetros como número de pasos por entrenamiento, nuevos niveles e hiperparámetros, e incluso modificar la red neuronal para seguir analizando el comportamiento.

Igualmente existen otros algoritmos de RL por explorar, como **Asynchronous Advantage Actor-Critic Algorithm, Proximal Policy Optimization,** etc*.* que podrían implementarse para comparar resultados.

8 Glosario

- **IA** Inteligencia Artificial
- **ML** Machine Learning
- **CNN** Convolutional Neural Network
- **RL** Reinforcement Learning
- **DRL** Deep Reinforcement Learning
- **DQN** Deep Q Network
- **DDQN** Double Deep Q Network
- **MDP** Markov Decission Processes
- **TD** Temporal Differences
- **PPO** Proximal Policy Optimization

9 Bibliografía

[1] Stuart Russell, Peter Norvig. Artificial Intelligence: A Modern Approach

[2] Patrick Winston (1990) “Artificial Intelligence”, Addison-Wesley Iberoamericana, Argentina, 1994

[3] García Fernández, Luis Alberto (2004), “Uso y aplicaciones de la inteligencia artificial”, La Ciencia y el Hombre, septiembre-diciembre 2004 , no. 3, p. 29-32

[4] Turing, Alan (1948), «Machine Intelligence», Copeland, B. Jack, ed., The Essential Turing: The ideas that gave birth to the computer age, Oxford: Oxford University Press

[5] Turing, Alan (1952), «Can Automatic Calculating Machines be Said to Think?», Copeland, B. Jack, ed., The Essential Turing: The ideas that gave birth to the computer age, Oxford: Oxford University Press

[6] John D. Kelleher, Brain Mac Namee, y Aoife D’Arcy, Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studie

[7] Li Deng & Yang Liu 2016. Deep Learning in Natural Language Processing. Springer

[8] Bellman, Richard (1978). An introduction to artificial intelligence: can computers think? San Francisco: Boyd & Fraser Pub. Co

[9] Richard Bellman. Dynamic Programming. 1st ed. Princeton, NJ, USA: Princeton. University Press, 1957.

[10] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, Cambridge, 2017

[11] Y. Bengio, A. Courville, and P. Vincent., "Representation Learning: A Review and New Perspectives," IEEE Trans. PAMI, special issue Learning Deep Architectures, 2013

[12] Cho, K. (2014). Foundations and Advances in Deep Learning. PhD thesis, Aalto University School of Science.

[13] Coates, A., Huval, B., Wang, T., Wu, D. J., Ng, A. Y., and Catanzaro, B. (2013). Deep learning with COTS HPC systems. In Proc. International Conference on Machine learning (ICML’13).

[14] Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more, 2nd Edition

[15] Melrose Roderick, James MacGlashan, Stefanie Tellex, Implementing the Deep Q-Network

[16] Abounadi, J., Bertsekas, D., and Borkar, V. S. (2002). Learning algorithms for Markov decision processes with average cost. SIAM Journal on Control and Optimization, 40(3):681–698.

[17] Curso Udemy. Curso completo de Inteligencia Artificial con Python. Juan Gabriel Gomila Salas

[18] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: Proceedings of the IEEE 86.11 (1998), pp. 2278–2324.

[19] Rainbow: Combining Improvements in Deep Reinforcement Learning https://arxiv.org/pdf/1710.02298.pdf

[20] Playing Atari with Deep Reinforcement Learning https://arxiv.org/pdf/1312.5602.pdf

[21] Human-level control through deep reinforcement learning http://files.davidqiu.com//research/nature14236.pdf

[22] Deep Reinforcement Learning with Double Q-Learning https://arxiv.org/abs/1509.06461

[23] Dueling Network Architectures for Deep Reinforcemnt Learning https://arxiv.org/pdf/1511.06581.pdf

[24] Prioritized Experience Replay https://arxiv.org/abs/1511.05952

[25] Noisy Networks for Exploration https://arxiv.org/abs/1706.10295

[26] Faster Deep Q- leraning using Neural Episodic Control https://arxiv.org/pdf/1801.01968.pdf

[27] A Distributional Perspective on Reinforcement Learning https://arxiv.org/pdf/1707.06887.pdf

[28] https://flyyufelix.github.io/2017/10/24/distributional-bellman.html

[29] https://contest.openai.com/2018-1/details/

[30] https://gym.openai.com/

[31] Gotta Learn Fast: A New Benchmark for Generalization in RL

[32] Playing hard exploration games by watching YouTube

[33] Gotta Learn Fast: A New Benchmark for Generalization in RL https://arxiv.org/pdf/1804.03720.pdf

[34] Mnih, Volodymyr y col. (2016). «Asynchronous methods for deep reinforcement learning». En: International Conference on Machine Learning

[35] mistake-in-retro-contest-of-OpenA https://github.com/xupe/mistake-in-retro-contest-of-OpenAI

[36] https://openai.com/blog/first-retro-contest-retrospective/

10 Anexos

- Repositorio de GitHub de los prototipos de agente iniciales

  https://github.com/Belfor/Sonic-Deep-Q-Learning