UNIVERSIDAD CENTRAL DE CHILE
FACULTAD DE INGENIERÍA
ESCUELA DE COMPUTACIÓN E INFORMÁTICA

# Desarrollo de algoritmos continuos de aprendizaje por refuerzo interactivo en agentes móviles

Memoria para optar al título profesional de
Ingeniero en Computación.

Profesor Guía: **Francisco Cruz Naranjo**
Profesor Informante: **Alejandro Sanhueza Olave**
Profesor Informante: **Erwin Fischer Angulo**

**Angel Antonio Ayala Maldonado**

Santiago, Chile
2018

*Esta tesis está dedicada a mi familia*
*quien me ha permitido seguir aprendiendo.*
*Angel Ayala Maldonado.*

# Resumen

Las investigaciones en el área de sistemas inteligentes han permitido encontrar varios métodos para que una máquina pueda obtener conocimiento, uno de éstos corresponde a aprendizaje por refuerzo. El problema de este método es el tiempo que requiere para aprender a resolver un problema, por lo que este trabajo aborda el enfoque de aprendizaje por refuerzo interactivo como una manera de solución para el entrenamiento de agentes robóticos. Por otra parte este trabajo aborda los problemas desde una representación continua además del enfoque interactivo. Para esto se experimenta con entornos simulados que poseen diferentes representación en su vector de estado para mostrar la eficiencia de este enfoque bajo una determinada probabilidad de interacción. Los resultados obtenidos evidencian una convergencia mayor del aprendizaje en términos de la recompensa acumulada por los agentes, demostrando la eficiencia del enfoque presentado.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

El presente trabajo es el resultado de la recopilación de información y el estudio sobre sistemas de aprendizaje autónomo, tecnología que se ha investigado para su implementación en diversas áreas que requieren de la automatización para solucionar alguna problemática. Trabajos como los mostrados en (Williams, 1992; Thrun, 1992; Sutton, 1984), entre otros, se han llevado a cabo desde la primera máquina propuesta por Alan Turing (Turing, 1948), *pleasure-pain system*, cuyo diseño es un sistema que funciona bajo el principio de *Law Effect* (Thorndike, 1911).

A continuación se expone la motivación del presente trabajo, la definición del problema que se va a tratar, los objetivos planteados para dar solución al problema, así como la metodología que se utilizará para llevar a cabo los objetivos y en lo que contribuye el trabajo. Además se nombra la estructura que posee este documento.

## 1.1. Motivación

En la actualidad la tecnología nos ha brindado la posibilidad de ejecutar tareas repetitivas y de gran esfuerzo gracias a mecanismos robóticos programados para realizar una actividad en específico, por ejemplo, en una fábrica de producción lineal existen una variedad de estructuras robóticas, ubicadas una después de la

otra, que complementan sus funciones para elaborar un producto específico. Sin embargo, existen actividades de mayor complejidad que los sistemas de producción en línea no son capaces de realizar bajo una metodología clásica de desarrollo de algoritmos (Cruz et al., 2014), y se hace necesaria la implementación de técnicas de inteligencia artificial, que proporcione el conocimiento necesario al sistema para la ejecución de tareas (Marsland, 2009).

## 1.2. Definición del problema

La definición de adquisición de conocimiento o aprendizaje desde una perspectiva biológica, se puede establecer como la interacción de un individuo con su entorno, donde éste identifica el cambio generado frente a la acción realizada. Estas interacciones, cuyo origen permite influenciar un determinado comportamiento a su entorno, proveen una vasta fuente de información (Thorndike, 1911).

Los algoritmos de aprendizaje por refuerzo o *Reinforcement Learning* (RL), son un acercamiento computacional de aprendizaje desde la interacción, el cual está enfocado en aprendizajes mediante interacción orientados por objetivos (Sutton and Barto, 1998). Estos algoritmos brindan la capacidad para que los robots (o sistemas automatizados) puedan ejecutar tareas de mayor complejidad debiendo, tal como los humanos, aprender a ejecutarla para conseguir el objetivo. Sin embargo, estos algoritmos poseen un problema de rendimiento, los cuáles pueden derivar en una implementación costosa en la cantidad de tiempo empleado para alcanzar el aprendizaje (Cruz et al., 2014).

Una alternativa ya existente, que reduce el tiempo necesario en el aprendizaje, es *Interactive Reinforcement Learning* (IRL) para un vector de estado con dominio discreto (Cruz et al., 2018). En caso que un problema posea un vector de estado con dominio continuo, se hace necesario un preprocesamiento del estado que sea capaz de discretizarlo. Por esto en este trabajo se implementa el enfoque interactivo que sea capaz de trabajar con un vector de estado continuo.

## 1.3. Objetivos

### 1.3.1. Objetivo general

Estudiar, diseñar, desarrollar y comparar algoritmos de aprendizaje por refuerzo y algoritmos continuos de aprendizaje por refuerzo interactivo para optimizar el aprendizaje en robots móviles en entornos simulados.

### 1.3.2. Objetivos específicos

En el presente trabajo, se han definido los siguientes objetivos específicos:

- Estudiar sobre algoritmos de aprendizaje por refuerzo interactivo discretos y continuos, y sus aplicaciones en aprendizaje de agentes móviles.
- Definir escenario robótico simulado y alcances para la optimización del aprendizaje.
- Desarrollar algoritmos de aprendizaje por refuerzo y refuerzo interactivo, tanto discretos como continuos.
- Ejecutar pruebas de algoritmos de aprendizaje en escenario robótico simulado.
- Analizar los resultados obtenidos y comparar entre los algoritmos la velocidad del aprendizaje.

## 1.4. Metodología de trabajo

Dado que el proyecto es una investigación, se quiere demostrar que el enfoque interactivo mediante una representación continua posee mejor convergencia en el aprendizaje, correspondiente al tiempo necesario para esto. Por lo mencionado anteriormente se hace uso del método científico que permite ejecutar en fases el proyecto, logrando el planteamiento, la ejecución y la obtención de resultados a interpretar. Esta metodología se puede dividir en cinco pasos (Nola and Sankey,

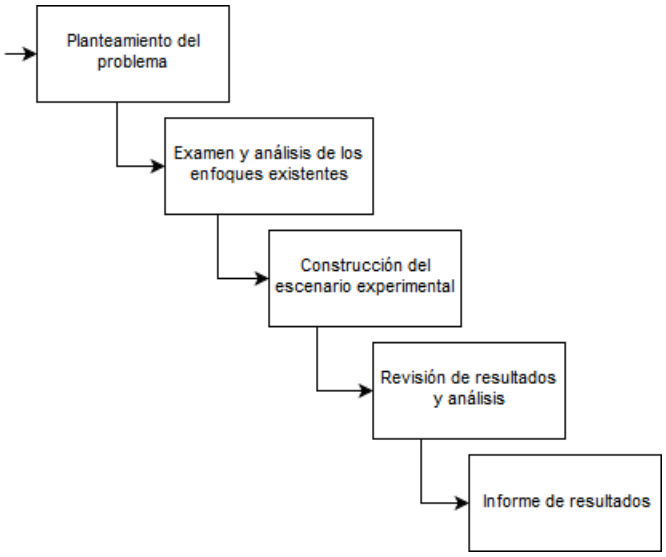


**Figura 1.1:** Los cinco pasos llevados a cabo en el método científico. Adaptada desde (Nola and Sankey, 2007).

2007), tal como se observa en la Figura 1.1 cada una de las fases descritas a continuación.

- **Planteamiento del problema:** RL requiere tiempo excesivo para encontrar una política apropiada.

- **Examen y análisis de los enfoques existentes:** Se realizará una revisión exhaustiva del marco teórico y de las investigaciones recientes, así obtener una visión detallada de los enfoques útiles.

- **Construcción del escenario experimental:** Los métodos encontrados en el paso anterior se integrarán en un escenario robótico simulado, incluyendo consejos de un entrenador para acelerar la adquisición del conocimiento sobre cómo mantener una vara en posición vertical y otro entorno simulado correspondiente a un video juego. En este sentido, se ha implementado un agente que ha obtenido la mejor recompensa para entrenar otros agentes mediante el enfoque interactivo.

- **Revisión de resultados y análisis:** Los resultados sobre el logro del objetivo de completar la tarea se evaluarán en esta etapa. Con este fin, se utilizan

las recompensas de los diferentes agentes aprendices para evaluar el punto de convergencia y la velocidad de la convergencia.

- **Resultados del informe:** Todos los resultados obtenidos serán reportados a través del desarrollo de esta tesis. Además se postuló para la publicación de un artículo de índole académico que ha sido aprobado para su presentación.

## 1.5. Resultados obtenidos y contribución del trabajo

Dado el tiempo que requiere RL para converger en el aprendizaje para desarrollar una tarea, es que requiere de métodos que optimicen este tiempo de convergencia. En este trabajo se desarrolla un algoritmo como extensión del enfoque antes mencionado, que permita la interacción con otro sistema automatizado llamado aprendizaje por refuerzo interactivo (IRL) propuesto para optimizar el rendimiento del algoritmo RL utilizando un vector de estados con dominio continuo.

Con este trabajo se ha logrado demostrar que el enfoque interactivo para RL presenta mejor convergencia de aprendizaje al utilizar un vector de espacio continuo, o más general, la utilización del vector de estado bajo la representación original que posee el entorno.

## 1.6. Estructura del documento

A continuación se expone una descripción simple y resumen de las diferentes partes, capítulos y secciones del trabajo expuesto.

1. Introducción: En este capítulo se describe el motivo de esta tesis, el planteamiento del problema, los objetivos de la solución, así como la metodología para desarrollar la solución. Además se indica las principales novedades y contribuciones.

2. Marco teórico y estado del arte: Este capítulo menciona el avance actual en investigaciones sobre RL así como técnicas, ya desarrolladas, para resolver este tipo de problemas. Aquí se describe además el estado del arte para IRL.

3. Escenarios experimentales: Aquí se describen las características que poseen los diferentes entornos en los que se ejecutan los agentes para comparar resultados.

4. Diseño e implementación de los agentes: Este capítulo define el comportamiento de los agentes para las diferentes representaciones que presentan los entornos.

5. Comparación y análisis de resultados: Aquí se muestran y comparan los resultados obtenidos del entrenamiendo de los agentes en las diferentes representaciones.

6. Conclusiones: Este capítulo resume las principales ideas del trabajo realizado, discute los resultados y propone futuras mejoras.

# Capítulo 2

# Marco teórico y estado del arte

Para comenzar el desarrollo de la investigación, en el presente capítulo, se expone la documentación base para comenzar el trabajo del tema abordado, definiendo su origen, elementos y el actual estado del arte que se ha de complementar con la contribución del presente trabajo. Por lo que, en primera instancia, se define el concepto de Aprendizaje por refuerzo o *Reinforcement learning* (RL) (Sutton and Barto, 1998) que en sus inicios posee dos aristas principales, una hace referencia a la psicología animal definiendo el aprendizaje como el resultado del ensayo y error, y el segundo, a la solución del problema de control óptimo mediante funciones de valor y programación dinámica. Sin embargo, aparece luego una tercera arista que involucra métodos de diferenciación temporal, que da inicio al campo moderno de aprendizaje por refuerzo.

El método de aprendizaje por refuerzo es cualquier manera efectiva de resolver problemas de *Markov decision process* (MDP), que poseen una estrecha relación con los problemas de control óptimo, donde la programación dinámica permite acercar el aprendizaje mediante la iteración sucesiva de aproximaciones a la respuesta correcta. La definición del aprendizaje por refuerzo está basado en el principio de *Law Effect* (Ley de Efecto) (Thorndike, 1911), el cual establece: de muchas respuestas efectuadas frente a la misma situación, aquellas que conllevan una satisfacción ma-

yor para el animal, permitirá un fortalecimiento entre la acción y la situación, lo que aumenta la probabilidad en que éste repita dicha acción para esa situación, así su contraparte, si su respuesta conlleva malestar, se debilita este lazo entre acción y situación, disminuyendo la probabilidad en que vuelva a ocurrir. Este principio además da inicio a teorías de aprendizaje por influencia.

Basándose en el contexto de aprendizaje animal, el término refuerzo aparece posterior al principio de Thorndike de Ley de Efecto, que establece el fortalecimiento de patrones de conducta como el resultado de un animal que recibe un estímulo en una relación temporal con otro estímulo o respuesta, donde algunos psicólogos amplían el concepto con el debilitamiento de estos patrones, que generan un cambio en el comportamiento (Sutton and Barto, 1998).

La idea de ensayo y error, bajo un acercamiento computacional, aparece por primera vez en un reporte de 1948, en el que Alan Turing describe un diseño de "pleasue-pain system" (sistema de placer dolor) que funciona bajo el principio de la Ley de Efecto: "Cuando se alcanza una configuración para la que la acción es indeterminada, se realiza una selección aleatoria de los datos faltantes y se realiza la entrada apropiada en la descripción, tentativamente, y se aplica. Cuando el estímulo del dolor ocurre, todas las entradas tentativas se cancelan y cuando ocurre un estímulo del placer, se hacen permanentes.", traducido (Turing, 1948).

## 2.1. Aprendizaje por refuerzo

RL es un método de aprendizaje que define la interacción entre un agente y su ambiente en términos de estados, acciones y recompensas, representando de manera simple los aspectos esenciales del problema de inteligencia artificial como se puede apreciar en la Figura 2.1, y está compuesto por los siguientes elementos (Sutton and Barto, 1998):

- Una **política**, que define la manera en que el agente selecciona una acción con la finalidad de mejorar la señal de recompensa obtenida.

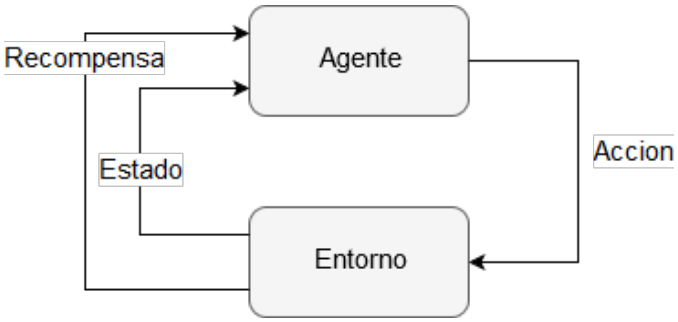


**Figura 2.1:** Diagrama de Aprendizaje por refuerzo.

- Una **señal de recompensa**, que establece el objetivo del problema de aprendizaje por refuerzo mediante la definición de eventos positivos (o buenos) y eventos negativos (o malos).
- Una **función de valor**, que especifica que tan buena es la señal de recompensa en el tiempo, a partir de un estado o un par estado-acción.
- Y opcionalmente, un **modelo del entorno**, que permite inferir cuál será el siguiente estado y recompensa, dado un estado y acción.

### 2.1.1. Proceso de decisión Markoviano

Para que exista un proceso de decisión Markoviano o *Markov decision process* (MDP) el vector de estado obtenido desde el entorno debe proveer toda la información necesaria del estado para tomar una decisión (Sutton and Barto, 1998), como las mediciones de los sensores, pero puede contener mucho más que esto. Esta representación de estado puede ser una estructura compleja compuesta por una secuencia de sensaciones. A esto se le llama *Markov property*.

Cuando el vector de estados y acciones pertenecen a un dominio discreto, se habla de un MDP finito, que particularmente está definido por la dinámica de su entorno donde la probabilidad del estado siguiente ($s'$) y la recompensa ($r$) dado un estado ($s$) y acción ($a$) se presenta en la ecuación (2.1)

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = t, A_t = a\} \tag{2.1}$$

Con esta representación discreta de estados se puede implementar una interfaz de agente-ambiente, en donde el actor o agente interactúa con su entorno (Sutton and Barto, 1998) para llevar a cabo una tarea. Esta interacción se efectúa mediante una función de transición que para determinada acción ejecutada por el agente, modifica el entorno observable que a su vez le entrega una señal de recompensa que dada la ecuación (2.1) se puede calcular la recompensa esperada dado un par estado-acción de acuerdo a la ecuación (2.2) y le permite visualizar el estado en el que este se encuentra.

$$r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a) \tag{2.2}$$

Con esto se establece que para resolver un MDP mediante RL este debe poseer un espacio de estados, acciones, una función de transición cuya probabilidad está denotada por la ecuación (2.3) y una función de recompensa.

$$p(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a) \tag{2.3}$$

### 2.1.2. Métodos de predicción de diferenciación temporal

La metodología de aprendizaje de diferenciación temporal o *Temporal-Difference learning* (TD) permite buscar solución al problema de predicción o política de evaluación, siendo el resultado de la combinación de ideas sobre la Metodología de Monte Carlo y de Programación Dinámica (Sutton and Barto, 1998), donde ambos métodos aprenden directamente desde una experiencia directa sin un modelo del entorno, estos métodos actualizan la estimación de la función de valor basados en el aprendizaje de otras estimaciones sin la necesidad de alcanzar previamente un episodio final.

La programación dinámica, para un determinado modelo de entorno basado en un MDP, se encarga de evaluar la función de valor para una determinada política y optimizarla mediante una función de aproximación. El método de Monte Carlo

a diferencia de la programación dinámica (Sutton and Barto, 1998) no requiere conocer el modelo del entorno, sin embargo, necesita de una muestra con secuencias de estados, acciones y recompensas de la interacción con el entorno o experiencia, presentando una gran ventaja para su implementación con la finalidad de explorar en una cantidad suficiente para converger en el aprendizaje.

La idea de solución para este problema es encontrar una predicción lo más certera al retorno esperado de la función de valor de una determinada política denominada $\pi$, por esto el problema se aborda como dos procesos donde el primero se encarga de la comparación y aproximación de este valor obtenido y el otro debe permitir una cantidad de exploración suficiente para elegir acciones que pueden corresponder a una buena decisión para optimizar la aproximación de la función de valor.

El acercamiento más simple para TD corresponde a TD(0), donde la actualización de la función de valor ($V$) se genera mediante dos parámetros constituidos por la tasa de aprendizaje ($\alpha$) y el factor de descuento para los valores futuros ($\gamma$) como se muestra en la ecuación (2.4).

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.4}$$

Estos métodos de predicción se pueden clasificar principalmente en dos grupos, el primero con una actualización *on-policy* y el segundo con una actualización *off-policy*.

#### 2.1.2.1. Método de predicción on-policy

Para estos métodos se consideran las transiciones del par estado-acción a otro par estado-acción (Singh and Sutton, 1996) definido como SARSA, que obtiene su nombre de la tupla de $<$ estado$_t$, acción$_t$, recompensa$_t$, estado$_{t+1}$, acción$_{t+1}$ $>$ ($< S_t, A_t, R_t, S_{t+1}, A_{t+1} >$), este método presenta una aproximación influenciable donde la selección de una acción influye en el valor de la función de valor para la actualización de la política. Este método como extensión de TD(0) (ecuación (2.4))

actualiza la función de valor ($Q$) para un determinado par estado-acción como se muestra en la ecuación (2.5).

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})) - Q(S_t, A_t)] \tag{2.5}$$

#### 2.1.2.2. Método de predicción off-policy

El método de Q-learning (Watkins, 1989) debido a que el aprendizaje de la política de selección de acción es independiente de las acciones realizadas por el agente, generando con esto una aproximación más directa al finalizar cada episodio. Este método como extensión de TD(0) (ecuación (2.4)) actualiza la función de valor ($Q$) para un determinado par estado-acción donde no considera la política implementada como se muestra en la ecuación (2.6), generando una convergencia a la política óptima $q*$.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.6}$$

## 2.2. Aprendizaje interactivo

Este método de aprendizaje interactivo o *Interactive reinforcement learning* (IRL) es una extensión del aprendizaje por refuerzo, donde se incluye un entrenador externo que ofrece instrucciones para optimizar la toma de decisiones (Cruz et al., 2018), éste actuará como guía para el aprendiz mediante la estrategia de retroalimentación (Thomaz and Breazeal, 2006), que puede ser a través de la configuración en la política o la modificación en la señal de recompensa.

### 2.2.1. Configuración de política

En la estrategia de retroalimentación bajo la configuración de la política (véase Figure 2.2), la acción propuesta por el aprendiz, puede ser reemplazada por una acción mejor, elegida por el entrenador externo, antes de ser ejecutada (Cederborg

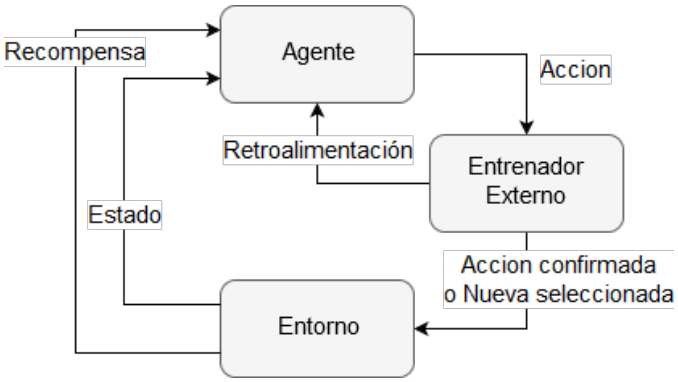


**Figura 2.2:** Diagrama de Aprendizaje por Refuerzo Interactivo mediante configuración de política.

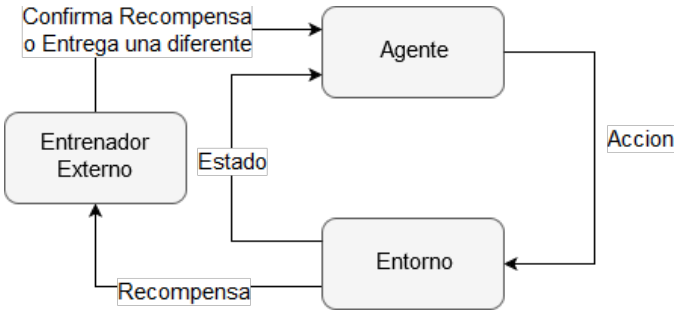


**Figura 2.3:** Diagrama de Aprendizaje por Refuerzo Interactivo mediante configuración de recompensa.

et al., 2015).

### 2.2.2. Modificación de la recompensa

En la modificación de la señal de recompensa (véase Figura 2.3), el entrenador es capaz de evaluar que tan buena o mala fueron las acciones ejecutadas por el agente aprendiz (Knox and Stone, 2012).

## 2.3. Aprendizaje por refuerzo continuo

En RL convencional solo son considerados MDP con vectores de estados y acción bajo un dominio discreto (van Hasselt and Wiering, 2007), sin embargo, en muchas aplicaciones del mundo real la discretización del espacio no es muy útil, ya que la generalización se hace más difícil de experiencias pasadas y el aprendizaje se

vuelve lento. Una manera para aprender desde un dominio continuo es Q-learning (Watkins and Dayan, 1992), que provee al agente la capacidad de aprender a actuar de manera óptima mediante la experiencia de las consecuencia de las acciones ejecutadas. Es un método incremental de programación dinámica que mediante sucesivas actualizaciones mejora la calidad de determinadas acciones para determinados estados.

Un algoritmo capaz de manejar un espacio continuo de estados y acción es el Continuos Actor-Critic Automaton (CACLA) que posee la habilidad de encontrar soluciones reales continuas, una mejor generalización de las propiedades y una rápida selección de acción como se muestra en (Zhong et al., 2012).

## 2.4. Discusión

De lo mencionado anteriormente se conoce que RL es un paradigma de aprendizaje de máquina en el que un agente debe ser capaz de resolver un MDP finito para aprender a ejecutar una tarea mediante prueba y error. Para esto el agente debe ser capaz de percibir su entorno mediante el vector de estado, para así tomar una decisión en la acción a ejecutar. Esta decisión está basada en una política $\pi$, la cual se ha de optimizar en cada iteración durante el entrenamiento mediante el método TD aplicado (*off-policy* u *on-policy*).

Un enfoque que permite optimizar el tiempo necesario para el aprendizaje corresponde a IRL, en donde aparece un segundo agente como entrenador, que dada una determinada probabilidad de interacción $\mathcal{L}$ este entrenador puede reemplazar la acción seleccionada por el agente aprendiz o modificar la recompensa que debe obtener.

Para el entrenamiento de los agentes se debe definir el escenario o entorno donde va a interactuar el agente. Muchos de los casos no presentan un vector de estado discreto sino que poseen un vector de estado continuo, por lo que debe considerar una función de aproximación para trabajar con estos estados de manera directa.

# Capítulo 3

# Escenarios experimentales

En este capítulo se definen los diferentes escenarios utilizados para el entrenamiento de los diferentes agentes desarrollados, los que deben ser capaces de resolver el respectivo proceso de decisión Markoviano (MDP) y optimizar la recompensa obtenida. La ejecución de los agentes será en dos ambientes simulados de distinta complejidad, el primero pertenece a un problema de rendimiento conocido como péndulo invertido (Brownlee, 2005), y el segundo a un entorno arcade (Bellemare et al., 2013) para generar una comparación de resultados en diferentes problemas, y la eficiencia que presenta el enfoque interactivo en cada uno de ellos.

Para estos ambientes simulados se implementa la librería de OpenAI, Gym (Brockman et al., 2016), que es una plataforma de evaluación que posee una amplia cantidad de entornos simulados con vectores de estado en diferentes representaciones.

## 3.1. Péndulo invertido

Este escenario estará basado en el péndulo invertido (véase Figura 3.1), que requiere un controlador basado en retroalimentación de circuito cerrado, cuyo comportamiento debe balancear un poste conectado a un carro dirigido por motores. El movimiento del carro está restringido a un movimiento de un eje horizontal

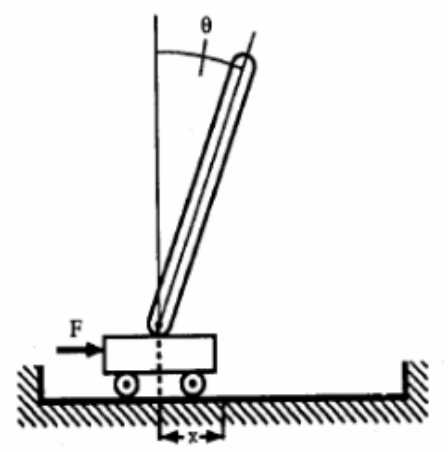


**Figura 3.1:** Ilustración del péndulo invertido, desde (Brownlee, 2005).

mediante una pista, y el poste es libre de moverse en este eje a través de un pivote.

### 3.1.1. Definición del sistema en una dimensión

Para la simulación del escenario experimental se utilizarán los parámetros para el sistema del péndulo invertido (Brownlee, 2005) definidas en la Tabla 3.1, correspondientes a las entradas, salidas y las variables físicas involucradas para el movimiento del poste y carro, necesarias para la aplicación en un ambiente simulado. Esta simulación corresponde al entorno *CartPole-v1* del la librería Gym, donde ya se encuentra implementado todos los aspectos mencionados que involucran el escenario.

### 3.1.2. Restricciones

La simulación finalizará cuando la inclinación de la vara supere el ángulo de falla, o el carro traspase los límites de la pista. El objetivo consiste en obtener un controlador que mantenga la vara en pie para una determinada cantidad de tiempo en la simulación, este controlador debe ejercer una fuerza constante a la vara en cualquier dirección. Los valores iniciales del sistema deben especificar la posición inicial del carro en la pista y el ángulo inicial del poste, los cuales se establecen en el centro de la pista y en el centro del carro respectivamente.

**Tabla 3.1:** Variables del sistema del péndulo invertido, adaptada desde (Brownlee, 2005).

| **Var.** | **Nombre** | **Descripción** |
|---|---|---|
| $\theta$ | Ángulo del poste | El ángulo del poste en radianes |
| $\theta'$ | Velocidad del poste | La velocidad angular del poste en radianes/segundos |
| $\theta''$ | Aceleración del poste | Aceleración del poste en radianes/segundos$^2$ |
| $x$ | Posición del carro | Medición de la diferencia respecto al centro de la pista en metros (0.0) |
| $x'$ | Velocidad del carro | Velocidad del carro en metros/segundos |
| $x''$ | Aceleración del carro | Aceleración del carro en metros/segundos$^2$ |
| $g$ | Aceleración de gravedad | Aceleración de gravedad $-9{,}81$ metros/segundos$^2$ (hacia arriba es positivo) |
| $m_c$ | Masa del carro | 1.0 Kilogramos |
| $m_p$ | Masa del poste | 0.1 Kilogramos |
| $l$ | Largo del poste | La distancia del pivote al centro de masa del poste 0.5 metros (ya que el largo de la vara es de $2l = 1{,}0$ metros) |
| $t$ | Tiempo | Medido en segundos |
| $F$ | Fuerza | La magnitud de fuerza aplicada al centro de masa del carro en el momento $t$. Típicamente una constante de $\pm 10$ newtons o $\pm 1$ newtons |
| $h$ | Largo de la pista | $\pm 2{,}4$ metros desde el centro de la pista (ya que $2h = 4{,}8m$) |
| $r$ | Ángulo de falla del poste | $\pm 12°$ desde $0°$ (aproximadamente $\pm 0{,}209$ radianes) |
| $\tau$ | Unidad de tiempo | El paso de tiempo de integración discreta para la simulación, convencionalmente $\triangle = 0,002$ segundos (50hertz) |

### 3.1.3. Aplicaciones

Este problema se utiliza como un caso de estudio simple en teoría de control (Brownlee, 2005) para investigaciones sobre controladores para problemas inestables no lineales como la recuperación de un sistema con malas condiciones iniciales. Este representa la idealización de un prototipo para laboratorio de un sistema mecánico inestable. Otras aplicaciones de este problema se ha utilizado para el

estudio como sistema de control dinámico en (Zhou and Liu, 2014; Bainbridge, 2010).

### 3.1.4. Péndulo invertido MDP

Para la implementación de los algoritmos de RL se definen los siguientes elementos según las definición de su MDP:

- **Estados**: El vector de estado posee una representación continua conformado por $< x, x', \theta, \theta' >$ correspondientes a la posición del carro respecto al centro de la pista, la velocidad del carro, el ángulo de inclinación del poste con el carro, y la velocidad angular del poste, respectivamente de acuerdo a (Brownlee, 2005).
- **Acciones**: Las acciones poseen una representación discreta en dos posibilidades que corresponden a la dirección en que se debe mover el carro en la pista, izquierda que ejerce la fuerza en un sentido o derecha que ejerce una fuerza en sentido contrario.
- **Función de Transición**: Para obtener una variación del vector de estado, se debe aplicar una fuerza al carro de magnitud constante obtenida de una ecuación que posee las cuatro variables de entradas al sistema, esta variación se obtiene mediante el uso de ecuaciones diferenciales propias del modelo físico del entorno (Brownlee, 2005).
- **Función de Recompensa**: Mientras el agente mantenga el poste en sentido vertical se le entregará una recompensa igual a 1, y si a éste se le cae, o traspasa los límites de la pista, la recompensa es igual a 0.

## 3.2. Arcade: Space Invaders

Este entorno corresponde a un juego de máquina llamado *Space Invaders*, el cual está basado en la plataforma de *Arcade Learning environments* (ALE) (Bellemare


**Figura 3.2:** Ilustración del entorno de *Space Invaders* (Bellemare et al., 2013).

et al., 2013) mediante la librería de Gym. Esta plataforma presenta un vector de estados con una representación visual que puede ser obtenida mediante la lectura de la imagen en bruto, correspondiente a un fotograma de 210x160 píxeles con profundidad de colores en RGB, como se puede apreciar en la Figura 3.2 o mediante la lectura directa de la información almacenada en 128 bytes en RAM, utilizada por ALE para la representación interna del estado del juego (Desai and Banerjee, 2017).

Este entorno presenta un vector de estado más complejo que el mencionado anteriormente del péndulo invertido, dado que el estado está compuesto por una mayor cantidad de variables e incluso dimensiones. Por ejemplo, su representación en RGB posee 100.800 elementos así como su representación en memoria RAM posee 128 elementos.

El objetivo para este entorno es conseguir la mayor recompensa posible, mediante la selección iterativa de las acciones disponibles desde lo percibido desde el vector de estado.

### 3.2.1. Space Invaders MDP

Al igual que el entorno anterior se identifican los elementos del MDP con los que se ha de implementar el agente:

- **Estados**: El vector de estado corresponde a una imagen de 210x160 píxeles de tamaño con una profundidad de color en RGB, mostrada cuadro a cuadro en la simulación correspondiente. Que es utilizada como entrada para la red neuronal.
- **Acciones**: Las acciones disponibles para el agente se encuentran en una representación discreta donde de acuerdo a su valor es la correspondiente acción a ejecutar, teniendo la posibilidad de elegir entre 6 diferentes acciones donde se puede mover (izquierda o derecha), o no moverse, más las mismas acciones que incluyen la acción de disparar.
- **Función de Transición**: La transición de estado está compuesta por el motor Arcade (Bellemare et al., 2013), donde presenta cuadro a cuadro la respectiva imagen con la acción ejecutada en la simulación.
- **Función de Recompensa**: . La recompensa para este escenario está dada por las naves enemigas que sea capaz de destruir el agente, en donde las naves del nivel inferior presentan un puntaje de 15 puntos, sumando 5 puntos a cada nave enemiga, de acuerdo a la fila que corresponda llegando a un máximo de 40 puntos. Dentro del escenario además se encuentra una nave extra que entrega un puntaje de 200 puntos, la cual aparece en momentos aleatorios en la parte superior del cuadro de imagen.

## 3.3. Discusión

Se han definido dos escenarios con diferentes complejidades, el primero corresponde al péndulo invertido, y el segundo a un video juego *Space Invaders*. Para cada uno de estos se definen sus respectivos MDP con los estados, acciones, función de transición y función de recompensa. Estos escenarios son entornos simulados, en donde se deben implementar los agentes.

En el siguiente capítulo, se especifican las característica que tendrán los agentes para que puedan resolver el MDP de los diferente entornos mencionados anteriormente. Dada las distintas complejidad que poseen en su representación, es que se desarrollan diversos agentes que sean capaces de trabajar con sus respectivos vector de estado.

# Capítulo 4

# Diseño e implementación de los agentes

En el presente capítulo se describen los diferentes algoritmos o agentes que han de interactuar con los diferentes entornos simulados definidos en el capítulo anterior. Para el caso del péndulo invertido, se implementan dos agentes, uno capaz de interactuar mediante un vector de estados discreto, y otro mediante un vector de estados continuo. Para el entorno de *Space Invaders* se implementan dos agentes que deben percibir desde una representación visual toda la información que describe el estado actual del juego (Desai and Banerjee, 2017). El primero debe tratar esta representación en tres dimensiones desde la imagen en bruto. El segundo debe observar desde los 128 bytes almacenados en la RAM la información del entorno.

Para los diferentes agentes es necesario diseñar la manera en que se decide la acción a efectuar en su entorno donde se requiere un balance óptimo entre explotación y exploración del espacio de acciones, para lo cual la información disponible para esta decisión depende de las acciones tomadas con anterioridad, con la que el agente debe explorar el espacio de acción compensando las buenas acciones ya exploradas con otras que nunca intentó (Marsland, 2009). Para la solución de esta compensación entre explotación y exploración se implementa el método $\epsilon$-greedy

que posee un factor de exploración $\epsilon$, esta estrategia en general converge en una mayor recompensa sin quedar atrapado en un mínimo local (Sutton and Barto, 1998).

Para un primer acercamiento a este algoritmo se implementa el método *off-policy* Q-learning (Watkins, 1989), este método de aprendizaje le permite al agente la capacidad de actuar de manera óptima, mediante la experiencia de la consecuencia de las acciones sin la necesidad de construir un mapa del dominio Markoviano.

Para las representaciones no discretas, continua y visual, se implementa una técnica llamada *Experience Replay* (Adam et al., 2012), donde en una memoria $\mathcal{D}$ se almacenan experiencias u observaciones hechas por el agente, de donde se extrae una cantidad de muestras para la optimización de la red neuronal implementada en el agente.

## 4.1. Enfoque interactivo

Un primer acercamiento para este enfoque es mediante la implementación de un agente externo, el entrenador, que retroalimenta con una selección de acción según su configuración de política, a un segundo agente dada una determinada probabilidad $\mathcal{L}$. Dicho lo anterior, el desarrollo del algoritmo extiende el agente RL, modificando la forma en que selecciona la acción involucrando la política del entrenador como se muestra en el algoritmo 4.1. Esta implementación se efectúa para los agentes de los dos entornos en sus respectivas representaciones.

**Algorithm 4.1.** Agente IRL

```
function SELECCIONARACCION(s_t)
    if valorAleatorio < L then
        a_t ← función de valor del agente entrenador para s_t
    else
        a_t ← selecciona acción según política propia para s_t
    end if
    return a_t
end function
```

## 4.2. Representación discreta

Un primer acercamiento para la implementación del algoritmo para el péndulo invertido es abordar su representación discreta, para simplificar la complejidad en encontrar una política óptima de aprendizaje. Dado que el vector de estado para el sistema corresponde a valores continuos, se debe discretizar en rangos con límites superior e inferior. El enfoque BOXES (Miche and Chambers, 1968), divide el espacio de estados del sistema en regiones discretas llamadas particiones o boxes que reduce la complejidad del problema para el entrenamiento del agente, la dinámica para este tipo de algoritmo es actualizar la acción para cada partición (box).

En la implementación de Q-learning (Watkins, 1989) para este tipo de representación, se genera una matriz Q que almacena la recompensa obtenida para cada par de estado acción, permitiendo la simplificación en la estimación de la función de valor, validando que "tan buena" es la acción tomada dado un determinado estado, esta definición de "bueno" está basado en términos de la recompensa o retorno esperado para una determinada política (Sutton and Barto, 1998), la implementación del agente discreto se puede observar en el algoritmo 4.2.

**Algorithm 4.2.** Agente Discreto

```
Inicializar matriz Q con valores uniformes [-1, 1]
for Episodio = 1, M do
    Observar s_0 desde el entorno
    Discretización de s_0 con BOXES
    while s_t no sea terminal do
        Seleccionar acción a_t según política ϵ-greedy
        Observar s_{t+1}, r_t
        Discretización de s_{t+1} con BOXES
        Actualizar matriz Q para el par s_t, a_t
        s_t ← s_{t+1}
    end while
    Reducir valores de ϵ y α.
end for
```

## 4.3. Representación continua

Dado que la representación continua pertenece al conjunto de los números reales, es impracticable almacenar en memoria todos los posibles pares estado-acción, es por esto que para la solución en espacios continuos se utiliza una función de aproximación (van Hasselt and Wiering, 2007), la cual permite estimar el valor Q a partir del vector de estado sin necesidad de almacenarlo como un valor dentro de la memoria, sino como una función Q.

Para la implementación de red neuronal en esta representación, se utilizan dos redes neuronales del tipo perceptrón multicapa, una se utilizará para el entrenamiento del agente y la segunda para el cálculo del valor esperado para la función Q. Además se implementa una optimización de Temporal-Difference learning en donde se almacenan en memoria una tupla de $<$ estado$_t$, acción$_t$, recompensa$_t$, estado$_{t+1}$ $>$ la cual se utilizará para la obtención del conjunto para el entrenamiento de las redes neuronales (Sutton and Barto, 1998), que les permita encontrar la solución al sistema propuesto en el sistema (Brownlee, 2005).

Cada red neuronal posee una arquitectura de 4x24x24x2, la cual posee 4 entradas correspondientes al vector de estado, 2 capas ocultas con 24 neuronas en cada una, y 2 salidas que pertenecen a las dos acciones que puede seleccionar el agente que poseen una función de activación lineal, los pesos iniciales son definidos aleatoriamente desde una distribución normal y el error del entrenamiento corresponde al error cuadrático medio. Este método corresponde a una implementación incremental de Q-learning (Watkins and Dayan, 1992) implementado en el algoritmo 4.3.

## 4.4. Representación visual

Para la implementación del agente que corresponde al entorno de *Space Invaders*, se implementa el método de deep Q-network (Roderick et al., 2017), se puede observar en el algoritmo 4.4. Este método se diferencia del anterior Q-learning

**Algorithm 4.3.** Agente Continuo

```
Inicializar memoria 𝒟 con largo N
Inicializar función Q con pesos uniformes [-1, 1]
Inicializar aproximador Q con pesos uniformes [-1, 1]
for episodio = 1, M do
    Observar s_0 desde el entorno
    while s_t no sea terminal do
        Seleccionar acción a_t según política ϵ-greedy
        Observar s_{t+1}, r_t
        Almacenar en 𝒟 la tupla (s_t, a_t, r_t, s_{t+1}, esTerminal)
        Actualizar función Q desde 𝒟
        s_t ← s_{t+1}
    end while
    Asignar pesos de función Q al aproximador Q
    Reducir valor de ϵ.
end for
```

(Watkins and Dayan, 1992) dado que la red neuronal se reemplaza por una del tipo convolucional para la estimación del valor Q. La implementación para este enfoque dada la magnitud de la imagen y su costo computacional se hace necesario un preprocesamiento de la imagen (Mnih et al., 2015) en donde esta se reduce en tamaño, así como su profundidad, dejando una imagen de 84x84 píxeles en escala de grises.

Para trabajar con esta representación se implementa una red con una arquitectura de acuerdo a lo recomendado en (Mnih et al., 2015), con la siguiente disposición: la entrada corresponde a una imagen de 84x84x4, la primera capa oculta contiene 32 filtros con una ventana de 8x8 con pasos de 4 píxeles y una función de rectificación no lineal, la segunda capa oculta posee 64 filtros con ventana de 4x4 con pasos de 2 píxeles y rectificación no lineal, una tercera capa convolucional con 64 filtros con ventana de 3x3 con con pasos de 1 píxel. La última capa totalmente conectada, posee 512 unidades de rectificación, y la capa de salida es totalmente conectada y lineal con 6 neuronas que representan las 6 posibles acciones. Los pesos iniciales son definidos aleatoriamente desde una distribución normal y el error del entrenamiento corresponde a RMSprop (Ruder, 2016) que divide la velocidad de aprendizaje por un promedio que decae exponencialmente en gradientes cuadrados.

Una técnica necesaria para los entornos ALE es *frame skipping* (Braylan et al., 2015). Esta técnica establece que se debe repetir la acción cada $m$ fotogramas, lo cuál ha demostrado que de acuerdo a una determinada cantidad de fotogramas se puede obtener resultados de acuerdo al estado del arte en el rendimiento del algoritmo.

Otra manera de abordar este tipo de representación es mediante la lectura de la información de los 128 bytes en RAM que son utilizados para la representación interna del estado del juego (Desai and Banerjee, 2017). Para esta representación del estado se implementa una red neuronal del tipo perceptrón multicapa, cuya arquitectura, similar a la red neuronal implementada anteriormente, presenta solamente las capas con neuronas totalmente conectadas. Esta red neuronal presenta una capa de entrada para el vector de 128 números correspondiente a la imagen, una capa oculta con 512 neuronas y una función de rectificación no lineal, una segunda capa oculta con 128 neuronas y una función de rectificación no lineal, para la capa de salida son 6 neuronas correspondientes a cada una de las diferentes acciones disponibles para el agente. Para esta representación se utiliza el mismo algoritmo 4.4 en donde solamente se cambia la red neuronal implementada.

**Algorithm 4.4.** Agente SpaceInvaders

Inicializar memoria $\mathcal{D}$ con largo $N$
Inicializar función Q con pesos uniformes [-1, 1]
Inicializar aproximador Q con pesos uniformes [-1, 1]
**for** episodio = 1, M **do**
    Observar $s_0$ desde el entorno y preprocesar
    **while** $s_t$ no sea terminal **do**
        Seleccionar acción $a_t$ según política $\epsilon$-greedy para $m$ fotogramas
        Observar $s_{t+1}, r_t$
        Preprocesar $s_{t+1}$
        Almacenar en $\mathcal{D}$ la tupla $(s_t, a_t, r_t, s_{t+1}, \text{esTerminal})$
        Actualizar función Q desde $\mathcal{D}$
        $s_t \leftarrow s_{t+1}$
        Asignar pesos de función Q al aproximador Q cada $k$ fotogramas
        Reducir valor de $\epsilon$.
    **end while**
**end for**

## 4.5. Discusión

El diseño de los agentes con sus diferentes representaciones, consideran una política $\epsilon$-greedy y una optimización de la política mediante el método *off-policy* Q-learning. Esto le brinda al agente balancear su capacidad de exploración y explotación dada una probabilidad $\epsilon$, y además, converger en una política óptima de $q*$.

Para todos los agentes se implementa el enfoque interactivo en donde un segundo agente entrenador selecciona la acción que ha de ejecutar el primer agente dado una determinada probabilidad de interacción $\mathcal{L}$.

Para el agente de representación discreta del péndulo invertido, dado que la naturalidad del problema posee un vector de estado con dominio continuo, se requiere separar en regiones las diferentes variables del vector de estado para así discretizarlo, de tal forma que pueda ser manejado por el agente, el que almacena los valores Q para cada par estado-acción. Por otra parte, el agente para la representación continua observa el vector de estado directamente desde el entorno, pero a diferencia del agente discreto, este implementa una función de aproximación mediante una red neuronal del tipo perceptrón multicapa, para el valor Q de cada acción dado un determinado estado.

El agente que trabaja con la representación visual, al ser un entorno con una representación más compleja, funciona de manera similar que el agente continuo mediante una función de aproximación, pero en primera instancia se implementa una red neuronal del tipo convolucional para el tratamiendo de imagen.

En el siguiente capítulo, luego de lo menciondo anteriormente, se muestran los resultados obtenidos en el entrenamiento de los agentes mencionados con sus distintas representaciones, en los diferentes entornos simulados para comparar su rendimiento.

# Capítulo 5

# Comparación y análisis de resultados

En este capítulo se exponen los resultados de los diferentes agentes que fueron desarrollados, para su implementación en los distintos escenarios experimentales simulados. Cada uno de los agentes dada la diferencia en su estructura y manera de trabajo para las diferentes representaciones del vector de estado que presentan los diferentes entornos.

Para las diferentes representaciones en la ejecución de los algoritmos, se utilizaron dos entornos desde la librería Gym (Brockman et al., 2016). Para la representación discreta y continua se utilizó el entorno *CartPole-v1* que define la duración máxima de cada episodio en 500 unidades de tiempo, siendo este el valor máximo que se puede obtener como recompensa; un valor mínimo de 475 de recompensa indica que la tarea se ejecutó de manera exitosa para el episodio durante el entrenamiento del agente. Además se considera que la tarea está aprendida si completa satisfactoriamente la ejecución en 50 episodios consecutivos.

Para la representación visual se utiliza en primera instancia el entorno *SpaceInvaders-v0* que presenta como en su vector de estado la imagen en bruto con una extensión de 210x160 píxeles con colores en RGB. Dado la alta necesidad de cálculo para el

**Tabla 5.1:** Parámetros de discretización del vector de estado para el entorno de *CartPole-v1*.

| **Variable** | **Limites** | ***boxes*** |
|---|---|---|
| Posición del carro ($x$) | $[-4{,}8; 4{,}8]$ | 1 |
| Velocidad del carro($x'$) | $[-0{,}5; 0{,}5]$ | 1 |
| Ángulo del poste ($\theta$) | $[-0{,}419; 0{,}419]$ | 6 |
| Velocidad angular del poste ($\theta'$) | $[-0{,}873; 0{,}873]$ | 3 |

tratamiendo de imágenes, con los recursos económicos y de tiempo disponible para el proyecto, no es posible llevar a cabo el entrenamiento, ya que se requiere de un equipo que posea una unidad de procesamiento gráfico (GPU) para optimizar el tiempo requerido para el tratamiento de las imágenes. En caso contrario el tiempo necesario para el entrenamiento es aún mayor. Es por esto que se utiliza una variación del mismo entorno llamado *SpaceInvaders-ram-v0*, donde la información del estado se encuentra almacenado en RAM, disminuyendo con esto la complejidad del entorno, como el tiempo necesario para el aprendizaje.

## 5.1. Resultados representación discreta

Los resultados obtenidos en representación discreta corresponden a la implementación del algoritmo 4.2 como el agente, el que debe aprender a resolver el problema del péndulo invertido mediante el entorno *CartPole-v1*. El entrenamiento se ejecutó en 500 episodios, de los cuales en los primeros 350 muestra aprendizaje de la tarea. Para la discretización del espacio de estado, se definen límites y cantidad de *boxes* para cada una de las variables que componen el vector de estado como se muestran en la Tabla 5.1.

Los parámetros utilizados en el agente, exceptuando el factor de descuento, varían en relación con el tiempo por lo que se establece un valor máximo y un valor mínimo, tal como se muestran en la Tabla 5.2, la variación de los valores está dada por la ecuación (5.1).

**Tabla 5.2:** Parámetros para el agente con representación discreta.

| **Parámetro** | **Valor** |
|---|---|
| Factor de descuento ($\gamma$) | 0.99 |
| Tasa de exploración ($\epsilon$) | [1, 0.01] |
| Tasa de aprendizaje ($\alpha$) | [0.5, 0.1] |

**Tabla 5.3:** Índices de resultados entrenamiento RL con representación discreta.

| **Índice** | **Autónomo** | **Interactivo** | **Diferencia** | **Métrica** |
|---|---|---|---|---|
| Promedio | 423,02 | 438,23 | 15,21 | pasos |
| Máximo | 500 | 500 | 0 | pasos |
| Mínimo | 18 | 45 | 27 | pasos |
| Recomp. Ep. 50 | 22 | 65 | 43 | pasos |
| Recomp. Ep. 96 | 41 | 129 | 88 | pasos |
| Ep. Recomp. 450 | 196 | 173 | 23 | episodio |
| Ep. Recomp. 500 | 219 | 204 | 15 | episodio |

$$MAX(param_{max}, MIN(param_{min}, param - \log_{10}(\frac{\text{ep}+1}{25}))) \tag{5.1}$$

De los resultados obtenidos, se calculan índices que muestra la Tabla 5.3, donde expone las diferencias entre el método de RL con entrenamiento autónomo, contra el entrenamiento interactivo (IRL). Aquí se demuestra, que el enfoque interactivo, es capaz de conseguir una mayor cantidad de recompensa, logrando mantener por mayor cantidad de tiempo (pasos) el poste en sentido vertical, así como también, una mayor recompensa en menos cantidad de episodios.

Con estos parámetros, y tal como se aprecia en la Figura 5.1, se obtiene convergencia de aprendizaje en el episodio 150, pero no es capaz de mantener su comportamiento en el tiempo para los episodios consecutivos. Sin embargo el agente aprende a ejecutar la tarea satisfactoriamente desde el episodio 204, desde el cual es capaz de ejecutarla satisfactoriamente por más de 50 episodios consecutivos.

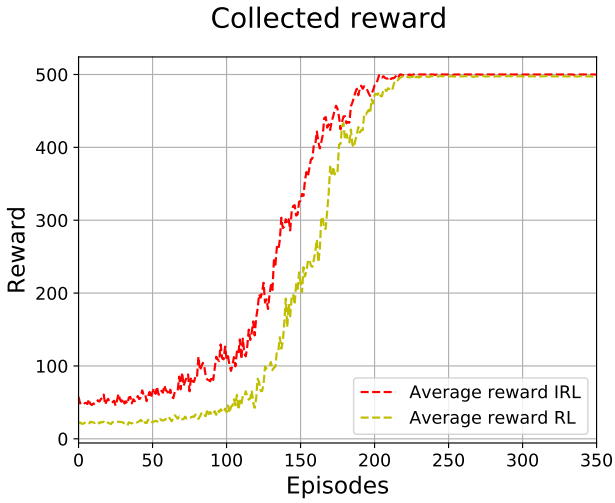


**Figura 5.1:** Resultado del entrenamiento de 50 agentes para el entorno *CartPole-v1* con discretización del vector de estado con BOXES (Miche and Chambers, 1968), y una probabilidad de feedback $\mathcal{L}$ de 0.3.

**Tabla 5.4:** Parámetros para el agente con representación continua.

| **Parámetro** | **Valor** |
|---|---|
| Factor de descuento ($\gamma$) | 0.99 |
| Tasa de exploración ($\epsilon$) | [1, 0.01] |
| Tasa de aprendizaje ($\alpha$) | 0.001 |

## 5.2. Resultados representación continua

Para esta representación se utiliza el entorno de *CartPole-v1*, donde el agente, implementado en el algoritmo 4.3, percibe un vector de estado con dominio continuo del péndulo invertido, el cual debe aprender a resolver. La ejecución del entrenamiento duró 500 episodios, de los cuales en los primeros 350 muestra aprendizaje de la tarea. Aunque el algoritmo requiere optimización para alcanzar la recompensa máxima y estable, el agente es capaz de aprender a ejecutar la tarea. Para esta implementación se utilizó una memoria de repetición $\mathcal{D}$ de largo 2000 para almacenar experiencias pasadas, desde la cual se obtiene un lote de 64 experiencias para la optimización de la política. De los parámetros utilizados para el agente, sólo la tasa de exploración presenta variación con el tiempo dada la ecuación (5.2), véase la Tabla 5.4.

**Tabla 5.5:** Índices de resultados entrenamiento RL con representación continua.

| **Índice** | **Autónomo** | **Interactivo** | **Diferencia** | **Métrica** |
|---|---|---|---|---|
| Promedio | 330,42 | 349,46 | 19,04 | pasos |
| Máximo | 462,14 | 450,36 | 11,78 | pasos |
| Mínimo | 16,12 | 26 | 9,88 | pasos |
| Recomp. Ep. 50 | 22 | 87 | 65 | pasos |
| Recomp. Ep. 96 | 163 | 440 | 277 | pasos |
| Ep. Recomp. 450 | 208 | 153 | 55 | episodio |

$$param_{t+1} = param_t * 0{,}999 \tag{5.2}$$

De los resultados obtenidos, se calculan índices que muestra la Tabla 5.5 con las diferencias entre el método de RL con entrenamiento autónomo, contra el entrenamiento interactivo (IRL). Para este caso, al igual que en la representación discreta, se obtienen datos que indican que el enfoque interactivo presenta mejores resultados, tanto en la cantidad de recompensa obtenida, como el tiempo que requiere para obtener una alta recompensa (450 pasos). A diferencia, del método discreto, es que los valores obtenidos son aún mayor, consiguiendo una alta recompensa en menos cantidad de episodios (como el episodio 96).

De estos parámetros se obtiene convergencia de aprendizaje en el episodio 139, pero no es capaz de mantener su comportamiento estable en los episodios siguientes, no obstante desde el episodio 163, el agente es capaz de mantener estable el poste durante más episodios consecutivos, alcanzando los 50 episodios necesarios para finalizar el entrenamiento, esto se puede observar en la Figura 5.2.

## 5.3. Resultados representación visual

La primera implementación para esta representación es en el entorno de *SpaceInvaders-v0*, donde se ejecuta el algoritmo 4.4 con una red neuronal convolucional, la que permite percibir y representar el estado desde entornos visuales (Krizhevsky et al.,

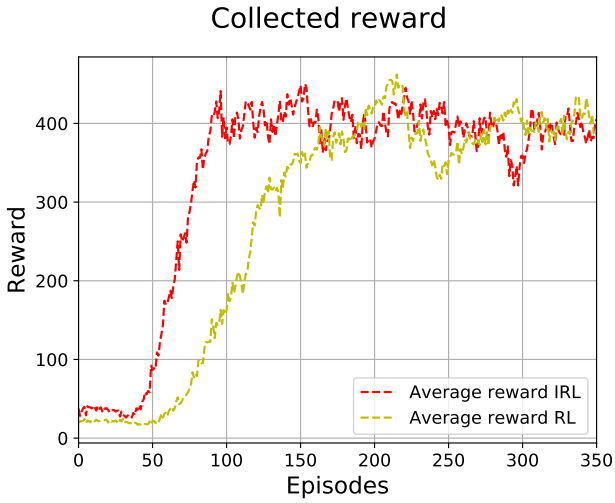


**Figura 5.2:** Resultados del entrenamiento de 50 agentes para entorno *CartPole-v1* con representación continua y una probabilidad de feedback $\mathcal{L}$ de 0.3.

2012) donde la entrada de la red corresponde a la imagen en bruto de 210x160 píxeles y sus salidas a las diferentes acciones que es capaz de ejecutar el agente. Para este agente no se concluyó el entrenamiento, dado que no se obtuvieron los resultados esperados en su ejecución de manera autónoma, lo que no permite completar el entrenamiento bajo la implementación del enfoque interactivo. Para el estado del arte en esta representación (Mnih et al., 2015), se encontró una limitante de tiempo para la ejecución de los agente con la arquitectura de red neuronal implementada, así como una limitante de hardware. La limitante principal es la utilización de procesadores gráficos del tipo CUDA (Sanders and Kandrot, 2010), para optimizar el tiempo de cómputo.

Dado lo anterior, conllevó a implementar el entorno de *SpaceInvaders-ram-v0*, que posee una representación del estado almacenado en 128 bytes de RAM, para lo que se implementó el algoritmo 4.4, pero esta vez con una red del tipo perceptrón multicapa. Este cambio se determinó durante el proyecto dado los inconvenientes encontrados en la ejecución del experimento del agente sobre este entorno.

Para el entrenamiento de este último agente, se utilizó una red neuronal que en comparación con la representación anterior, su arquitectura solo considera capas totalmente conectadas entre sus neuronas. Los parámetros para este agente se

**Tabla 5.6:** Parámetros para el agente con representación visual.

| **Parámetro** | **Valor** |
|---|---|
| Factor de descuento ($\gamma$) | 0.99 |
| Tasa de exploración ($\epsilon$) | [1, 0.01] |
| Tasa de aprendizaje ($\alpha$) | 0.0002 |

pueden observar en la Tabla 5.6, en donde la tasa de exploración presenta un decremento lineal en un 1M de fotogramas, comenzando en 1 hasta 0,01.

La ejecución del entrenamiento duró 1500 episodios, siendo aproximadamente 1.5M de fotogramas Para esta implementación se utilizó una memoria de repetición $\mathcal{D}$ de largo 1M para almacenar experiencias pasadas, desde la cual se obtiene un lote de 32 experiencias para la optimización de la política. Así como se repitieron las acciones cada 4 fotogramas.

De los resultados obtenidos mostrados en la Figura 5.3 corresponde al entrenamiento de 10 agentes en el entorno *SpaceInvaders-ram-v0* y una convolución al promedio de 50 episodios, se puede observar que el rendimiento del algoritmo para esta representación posee una tendencia de aumento en el puntaje para este entorno desde el episodio 600, presentando una recompensa promedio para los 1500 episodios de 170,15 puntos para el algoritmo autónomo, y 165,13 puntos para el algoritmo IRL. Aún así el enfoque interactivo presenta puntajes $\in [78; 319{,}5]$ en comparación con RL que tiene puntajes $\in [65{,}5; 362{,}5]$. Los resultados obtenidos como tál, no permiten dicernir mejoras, por lo que se ejecutan otras pruebas con un valor menor de probabilidad de interacción $\mathcal{L}$; y otra con el mismo valor de $\mathcal{L}$ pero con otro entrenador con una recompensa diferente. Para estas pruebas se mantienen los valores de los parámetros para RL.

Debido a que los resultados obtenidos en primera instancias no son lo esperado, se entrenan otros agentes, esta vez reduciendo el valor de la probabilidad de feedback $\mathcal{L}$, a un valor de 0,1 en donde, el entrenador le entrega consejo al aprendiz el 10 % del tiempo. Este entrenador es el mismo para los resultados de la Figura 5.3. Estos resultados se reflejan en la Figura 5.4, en donde se puede observar que el

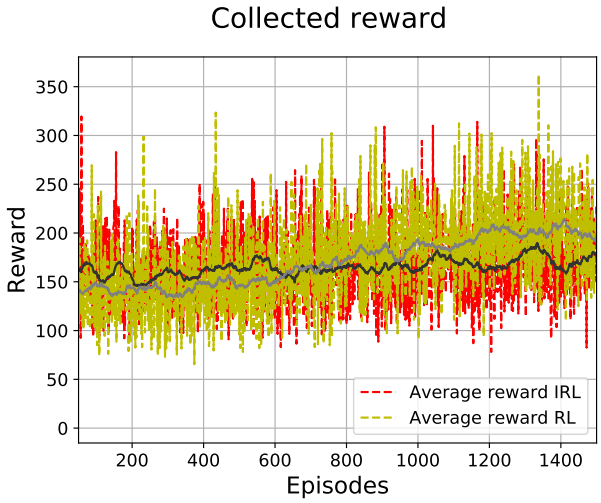


**Figura 5.3:** Resultados del entrenamiento de 10 agentes con promedio convolucional de 50 episodios para el entorno *SpaceInvaders-ram-v0* con representación visual y una probabilidad de feedback $\mathcal{L}$ de 0.3 desde un agente con recompensa promedio de 163 pasos.

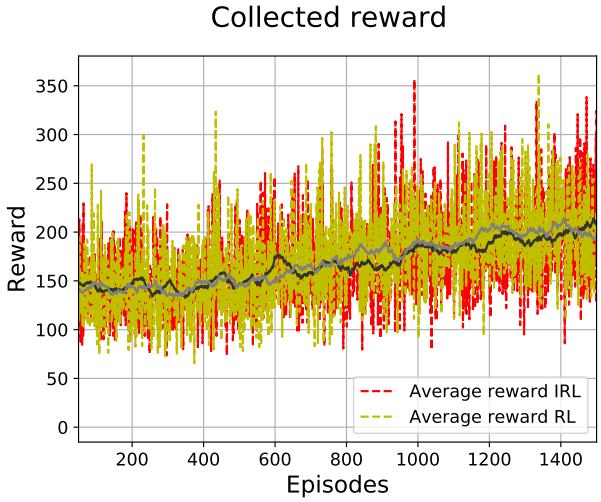


**Figura 5.4:** Resultados del entrenamiento de 10 agentes con promedio convolucional de 50 episodios para el entorno *SpaceInvaders-ram-v0* con representación visual y una probabilidad de feedback $\mathcal{L}$ de 0.1 desde un agente con recompensa promedio de 163 pasos.

comportamiento entre ambos métodos (RL e IRL) es similar indicando una leve mejora en la recompensa obtenida. Para estos resultados se obtienen diferentes índices cuantitativos, cómo en la Tabla 5.7, para cotejar los resultados en una segunda forma de visualización.

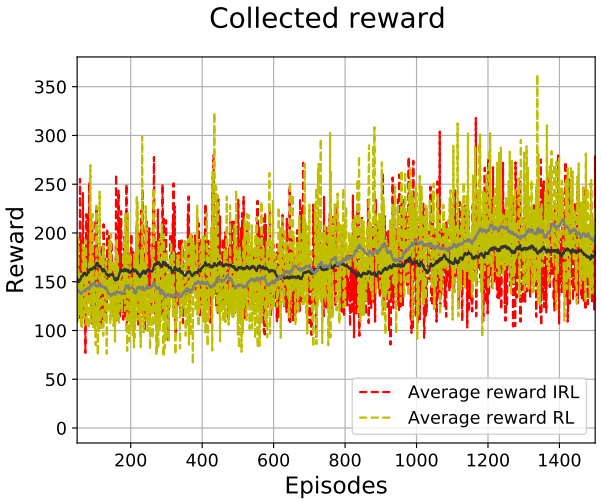


**Figura 5.5:** Resultados del entrenamiento de 10 agentes con promedio convolucional de 50 episodios para el entorno *SpaceInvaders-ram-v0* con representación visual y una probabilidad de feedback $\mathcal{L}$ de 0.3 desde un agente con recompensa promedio de 180 pasos.

Con los resultados obtenidos hasta aquí, las diferencias encontradas, no presentan datos que permitan dicernir un buen resultado, por esto, se ejecuta otro experimento en donde se cambia el entrenador por otro con una recompensa mayor, y manteniendo el valor de la probabilidad de feedback $\mathcal{L}$ del primer experimento. Estos resultados se pueden observar en la Figura 5.5, en donde se observa una recompensa similar al enfoque autónomo, pero no alcanza buenos resultados.

En la Tabla 5.7, se visualizan los índices obtenidos para el enfoque RL autónomo, y además los valores obtenidos para las tres diferentes ejecuciones del experimento, en donde se presentan los valores del agente entrenador que obtuvo una recompensa promedio de 163 pasos con una probabilidad de feedback $\mathcal{L}$ de 0,1 (**R163F01**), con una probabilidad de 0,3 (**R163F03**) y un entrenador con recompensa promedio de 180 pasos y una probabilidad de 0,3 (**R180F03**). Estos dos últimos casos, se entrenó un aprendiz durante 2000 episodios, pero se presentan los primero 1500 episodios.

**Tabla 5.7:** Comparación de resultados entrenamiento RL interactivo con representación visual.

| **Índice** | **Autónomo** | **R163F01** | **R163F03** | **R180F03** |
|---|---|---|---|---|
| Promedio | 170,15 | 176,70 | 165,13 | 171,66 |
| Máximo | 362,50 | 365,50 | 319,50 | 326,00 |
| Mínimo | 65,50 | 73,50 | 78,00 | 77,00 |

## 5.4. Discusión

En este capítulo se analizaron los resultados obtenidos en las diferentes representaciones, donde se entrenaron 50 agentes para el entorno del péndulo invertido, tanto para el método de RL autónomo como el método de RL interactivo (IRL). El valor usado para la probabilidad de interacción $\mathcal{L}$ es relativamente bajo, con un valor de 0,3, por lo que 30 % del tiempo el agente entrenador le indica al aprendiz qué acción debe ejecutar. Con lo mencionado anteriormente se observa que bajo el enfoque interactivo la convergencia de aprendizaje es mejor que para en enfoque autónomo. Además, con el mismo entorno, el agente que trabaja directamente con el vector de estado continuo, la convergencia de aprendizaje es aún mayor. Aún así, aunque en la representación discreta se obtiene una recompensa mayor y estable, esta discretización no es posible implementarse en muchos escenarios reales.

Para el caso del agente que interactúa mediante una representación visual del entorno, se ejecutaron tres experimentos diferentes, en donde se presentan diferentes parámetros utilizados, entre estos el valor de la probabilidad de interacción utilizada, como el agente entrenador. Dado que los resultados obtenidos no permiten diferenciar claramente la optimización del aprendizaje, se puede concluir que para este tipo de representación al presentar una complejidad aún mayor, el agente entrenador, se debe entrenar una mayor cantidad de tiempo, para que sea capaz de llegar a un valor mayor de recompensa. Esto dado que durante la ejecución de los experimentos se observó que el agente, a medida avanzan los episodios, aumenta gradualmente la recompensa promedio obtenida. Con esto se puede obtener además un entrenador que haya explorado mucho más su espacio de estados.

# Capítulo 6

# Conclusiones

Los algoritmos de aprendizaje por refuerzo se han aplicado para el desarrollo de diversos problemas, mucha de las veces, para un primer acercamiento a la solución propuesta requiere de pruebas, las que se pueden implementar mediante un simulador de entornos, mediante diversas herramientas o librerías de lenguajes para su utilización. El uso del entorno *CartPole-v1*, simulado de la librería Gym, permite comenzar el estudio para este tipo de algoritmos y sus métodos de optimización.

Como se puede apreciar en este trabajo, el agente que percibe un vector de estados con dominio continuo, como es la naturalidad del problema, obtiene una convergencia del aprendizaje en una menor cantidad de episodios comparado con la representación discreta de estados para el mismo problema.

Si bien se puede observar que la representación discreta obtiene mejores resultados en la recompensa, este no es lo óptimo dado que la discretización del espacio no es capaz de implementarse para muchos de los problemas en el mundo real.

Complementario a lo anterior, el enfoque interactivo para el agente de aprendizaje por refuerzo, permite una convergencia aún mejor respecto a su contraparte con enfoque autónomo para ambas representaciones, bajo la misma probabilidad de retroalimentación de la política.

Por otra parte, para la representación visual, dada a su complejidad no se pudo llevar a cabo el entrenamiento del agente con una red convolucional, lo que hubiese permitido un agente entrenador apto para el enfoque interactivo dado el actual estado del arte. Esto produce un cambio en la ejecución del proyecto, no obstante, se utiliza una variación del entorno que posee una representación en RAM del estado. Los resultados obtenidos con esta implementación, no presentan grandes variaciones en los puntajes por episodios, con lo cual, de acuerdo a los datos estadísticos de promedio, máximo y mínimo de puntajes, el enfoque interactivo presenta valores mayores en el máximo y el mínimo lo que indica que las recompensas obtenidas tienden a fluctuar en mejor resultados que el enfoque autónomo. Con una mayor cantidad de agentes entrenados es probable que esta diferencia se haga notar aún más.

Para trabajos futuros se contempla la implementación del agente IRL para espacios de acción continuo. Esto debiese significar una mejora en el tiempo requerido para converger el aprendizaje, basado en el algoritmo Continuos Actor-Critic Learning Automaton (van Hasselt and Wiering, 2007) que permite entrenar un agente con estados y acciones continuas.

# Apéndice A

# Lista de Acrónimos

RL – Reinforcement Learning.

IRL – Interactive Reinforcement Learning.

MDP – Markov Decision Process.

$\mathbb{E}$ – Valor esperado de una variable $X$

$Pr\{X = x\}$ – Probabilidad que la variable $X$ tome un valor $x$

TD – Temporal-Difference learning

SARSA – State, Action, Reward, State, Action.

CACLA – Continuos Actor-Critic Automaton

ALE – Arcade Learning Environments

RAM – Random Access Memory

# Apéndice B

# Agradecimientos

Para finalizar este trabajo, agradezco a mis profesores quienes me han brindado el conocimiento necesario y la motivación para desempeñar de manera exitosa, mis labores académicas.

Agradezco a mis compañeros de universidad, con quienes se comparten gratos momentos, se comentan experiencias y distribuye el conocimiento.

A quienes más le agradezco, es a mi familia quienes me han apoyado desde el comienzo de esta carrera, y siempre me han permitido saber que siempre cuento con ellos, queriendo así llegar aún más lejos.

Finalmente, agradezco al proyecto de investigación CIP201703 por el financiamiento parcial de este trabajo.

# Bibliografía

Adam, S., Busoniu, L., and Babuska, R. (2012). Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):201–212.

Bainbridge, C. J. (2010). Digital control networks for virtual creatures.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Braylan, A., Hollenbeck, M., Meyerson, E., and Miikkulainen, R. (2015). Frame skip is a powerful parameter for learning to play atari. In *AAAI-15 Workshop on Learning for General Competency in Video Games*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *ArXiv e-prints*.

Brownlee, J. (2005). The pole balancing problem: A benchmark control theory problem. Technical Report 7-01, Swinburne University of Technology, Melbourne, Victoria, Australia.

Cederborg, T., Grover, I., Isbell, C. L., and Thomaz, A. L. (2015). Policy shaping with human teachers. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3366–3372. AAAI Press.

Cruz, F., Magg, S., Nagai, Y., and Wermter, S. (2018). Improving interactive reinforcement learning: What makes a good teacher? *Connection Science*, 30(3):306–325.

Cruz, F., Magg, S., Weber, C., and Wermter, S. (2014). Improving reinforcement learning with interactive feedback and affordances. In *4th International Conference on Development and Learning and on Epigenetic Robotics*, pages 165–170.

Desai, N. and Banerjee, A. (2017). Deep reinforcement learning to play space invaders.

Knox, W. B. and Stone, P. (2012). Reinforcement learning from human reward: Discounting in episodic tasks. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 878–885.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Marsland, S. (2009). *Machine Learning: An Algorithmic Perspective.* Chapman and Hall/CRC, New York, 1st edition.

Miche, D. and Chambers, R. A. (1968). 9 boxes: An experiment in adaptive control.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Nola, R. and Sankey, H. (2007). *Theories of Scientific Method: An Introduction.* Acumen Publishing.

Roderick, M., MacGlashan, J., and Tellex, S. (2017). Implementing the Deep Q-Network. *ArXiv e-prints*.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv e-prints*, page arXiv:1609.04747.

Sanders, J. and Kandrot, E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Mach. Learn.*, 22(1-3):123–158.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. PhD thesis. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Última actualización - 2016-05-11.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Thomaz, A. L. and Breazeal, C. (2006). Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 1000–1005. AAAI Press.

Thorndike, E. L. (1911.). *Animal intelligence; experimental studies,*. New York,The Macmillan company,.

Thrun, S. B. (1992). Efficient exploration in reinforcement learning. Technical report.

Turing, A. M. (1948). Turing, intelligent machinery, a heretical theory. *Alan M. Turing*, pages 128–134.

van Hasselt, H. and Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279.

Watkins, C. J. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

Zhong, J., Weber, C., and Wermter, S. (2012). A predictive network architecture for a robust and smooth robot docking behavior. *Paladyn*, 3:172–180.

Zhou, X. and Liu, X. (2014). Learning and controling dynamic systems using gaussian process.